The Use of Proof Planning in Normalisation

Renato Busatto

Ph.D.
University of Edinburgh
1995

Abstract

Theorem proving in undecidable theories has to resort to *semi-decision procedures*, i.e. partial computable functions that halt when the input formula is a theorem, but may not terminate otherwise. Their performance can be improved once heuristic mechanisms are introduced to guide search. Heuristic functions, however, sometimes assume the properties of a particular subclass of formulae to be valid in a larger domain, and may as a result fail to recognise a theorem due to the loss of completeness.

Efficiency may be also improved when the decidability of subclasses of formulae is explored. *Decision procedures* establish, after a finite amount of time and computation, whether a formula of a class is a theorem or not. Their main advantage is their total computability, i.e. computation terminates for every formula of the domain, whether it is a theorem or not. Their reduced scope of application and, in some cases, their complexity are, nonetheless, the main limitations.

Evidence suggests that the development of efficient mechanical theorem provers requires the integration of both heuristic modules and decision procedures inside hybrid systems. Integration is achieved through at least three strategies,

(i) *juxtaposition*, where each component of the system operates independently from the others, and there is no communication between them,

(ii) *cooperation*, where the behaviour of a component influences others, and communication is direct, and

(iii) *interfacing*, where transformation or simplification steps take place before a formula is delivered to a component of the system, in which case communication amongst modules is mediated.

From the decision procedure viewpoint, the main effect of its integration into a prover is the enlargement of the decidable domain, whereas, for the heuristic component, there is a reduction in the number of subproblems it has to address.

Proof planning for normalisation provides a possible basis for the development of rewrite systems for interfacing decision procedures to heuristic provers. It includes a language for the description of recursive classes of formulae and a set of primitive operators or *normalisation tactics* for the construction of complex rewriters. Each tactic performs a specific syntactic task, such as the removal of occurrences of a symbol, the stratification of occurrences of a symbol over others, or the reorganisation of their occurrences. Rewrite rules are selected along these guidelines, and additional control mechanisms ensure that rewriting always terminates. Whenever an effective rewriter has a decidable subclass as range, the resulting extended class, i.e. the domain of the rewriter, is decidable. The introduction of new rules deductively strengthens the interface and enlarges the extended class, even though the full language in which the theory is defined is never encompassed.

Two alternative mechanisms are available under the proof planning approach. A specialised normaliser may be built by a *planner* to suit a particular description, given in terms of properties of a decidable class and its targeted extension. Primitive tactics are combined based on the partial specification provided by *methods*, until the desired reduction mechanism is obtained, whenever possible. Another solution involves the use of *general-purpose proof plans*, families of normalisers whose parameters are

instantiated in individual contexts to generate the desired system.

Since the development of interfaces relies entirely on the domain of a decision procedure, rather than on any of its computational features, the approach is highly modular. It is also general, given that any theory admits decidable subclasses. Empirical results concerning the application of general-purpose proof plans to arithmetical verification conditions indicate that they effectively widen the role of decision procedures in the verification of program correctness.

Acknowledgements

I would like to thank my supervisors, Alan Bundy, Helen Lowe and Toby Walsh, for their assistance during my period of studies in Edinburgh. Alan Bundy provided essential guidance through the large amount of material related to this dissertation. He is also the main responsible for arousing my interest in the empirical aspects of proof planning. Helen Lowe introduced me to the PRESS system and helped me at the critical stage of thesis writing. Toby Walsh gave valuable feedback in the early stages of the work, particularly during the preparation of the thesis proposal and the implementation of conditional methodicals for *Clam*.

Other members of the Mathematical Reasoning Group also contributed to the completion of this work. Alan Smaill advised me on the use of implication rewrite rules and the role of normalisation in constructive logics. Jane Hesketh introduced me to the Oyster proof system. Andrew Ireland helped me with the implementation of normalisation tactics. Ian Green issued pertinent criticism at the stage of formalisation of general-purpose proof plans. Alessandro Armando, from the University of Genova, currently visiting the group, introduced me to the GETFOL system, explained to me his work on decision procedures, and guided me in the revision of the chapter of the thesis that describes this system.

Special thanks are due to Rolando C. Sanchez, with whom I had an invaluable exchange of ideas over the last four years. His feedback was particularly helpful in areas such as logic programming, software engineering and theoretical computer science. He was also a constructive critic in the course of the implementation of general-purpose proof plans.

Finally, I am grateful to CAPES (Federal Agency for Postgraduate Education, Ministry of Education, Brazil) for having sponsored this research.

Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

Renato Busatto
Edinburgh
January 8, 1995

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The verification of program correctness is one amongst several applications of mechanical theorem proving to software development. Evidence suggests that efficient provers for program verification must contain both heuristic modules and decision procedures integrated inside a unified system. Proof planning for normalisation provides a framework for the creation of interfaces for module connection. It includes a language for the description of recursive classes of formulae and a set of atomic operators for the construction of complex rewriting interfaces.

Program verification is briefly discussed in section 1.1. Section 1.2 describes the integration problem for theorem provers, followed by a review of some of its solutions in section 1.3. Sections 1.4 and 1.5 present the main elements that compose the proof planning approach to normalisation. An outline of the remaining chapters of the dissertation in section 1.6 concludes the introduction.

1.1 Program Verification

The syntactic correctness of a program depends on the alphabet and the formation rules of the language in which it is written. Since programs provide a representation for mathematical objects such as sets, functions or relations, semantic correctness requires comparing properties of such objects with their representation. Given a set $\mathcal{S}$ and a (linguistic) representation $R(\mathcal{S})$ for it, $R(\mathcal{S})$ is *correct* if and only if every object that belongs to $R(\mathcal{S})$ also belongs to $\mathcal{S}$, and is *complete* if and only if every object of $\mathcal{S}$ also

belongs to $R(\mathcal{S})$. Whereas completeness is a desirable property for a representation, correctness is usually regarded as essential, in the sense that it must be guaranteed that none of the objects computed by $R(\mathcal{S})$ lies outside $\mathcal{S}$.

Informal correctness proofs frequently rely on the generation of a finite sample by a program, and the comparison of this sample with the mathematical object the program represents. For infinite objects, however, this approach is unsatisfactory. Rigorous correctness proofs require two distinct representations for the same object, the program itself and a *description* for the program (also expressed in a formal language), which are then compared extensionally. In the case of programs that compute functions, a description for it may have the form of a pair of formulae, $\langle \phi, \phi' \rangle$, where ϕ establishes properties of the domain (the *preconditions*), and ϕ', of the image (the *postconditions*). The partial correctness of a program P with respect to this description involves proving that, whenever an object x satisfies property ϕ and P computes an output y for x, y satisfies property ϕ', i.e.

$$(\forall x)((\phi[x] \wedge P(x) = y) \supset \phi'[y]) \quad (*)$$

An additional problem involves determining whether P halts for every element of the domain specified by ϕ,

$$(\forall x)(\phi[x] \supset (\exists y)(P(x) = y)) \quad (**)$$

When $(*)$ and $(**)$ are valid, P is correct with respect to $\langle \phi, \phi' \rangle$: for every input that satisfies ϕ, the program computes an output that satisfies the postconditions[1].

Condition $(*)$ can be established from the analysis of the flowchart of P. The input and output of P are associated with the description formulae, ϕ and ϕ', and intermediate nodes are linked to formulae that state properties of relevant variables. A *verification condition*,

$$\psi_i \supset \psi_j$$

[1] A description of the form $\langle \phi, \phi' \rangle$ actually identifies the family of functions that have $\{x \mid \phi(x)\}$ as domain and $\{y \mid \phi'(y)\}$ as image. Therefore, a program P is correct w.r.t. $\langle \phi, \phi' \rangle$ if and only if P computes an element of the family represented by the description. The notion of completeness is not applicable in this context, since P computes a single function.

is built for every path of the flowchart that links two nodes identified by formulae ψ_i and ψ_j. Once all of them have been proved, the correctness of P w.r.t. $\langle \phi, \phi' \rangle$ follows[2].

1.2 Decision Procedures & Heuristic Provers

Automated provers are mechanisms that have the set of formulae of a language as domain and a binary set, $\{1, 0\}$, as image, where 1 is associated with theorems of a particular theory, and 0 is reserved for non-theorems. Formal proofs may be derived as a side effect of the computation. A prover is *complete* when it identifies all the theorems of the theory. For undecidable axiomatisable theories, every complete prover is a *semi-decision procedure*, i.e. a partial function that halts when the input formula is a theorem, but may not terminate otherwise. Termination is arbitrarily obtained for all the domain once time or space constraints are imposed.

The performance of a prover may be improved, with or without loss of completeness, with the help of heuristic search mechanisms. Heuristic functions generalise partial information or knowledge available about subclasses of formulae of the domain. They may nonetheless fail to provide either the shortest proof for a conjecture, or a proof at all, due to loss of completeness. Empirical results, on the other hand, support the claim that domain knowledge effectively reduces search in the average case[3].

Improvements in efficiency also take place when the decidability of certain classes of formulae in undecidable theories is explored. Some theories admit effective algorithms to determine, for instance, whether a quantifier-free formula is a theorem or not. The main advantage of decision procedures is their total computability, i.e. computation terminates for all formulae of their domain, whether it is a theorem or not. The limitations are their reduced scope of application and, sometimes, their inefficiency. Heuristic functions and decision procedures are both important components in the construction of efficient provers for undecidable domains.

> "Decision procedures, alone or in co-operation with other decision procedures, are fast and predictable but often too limited to be of general use. On

[2] See [Anderson 79], p. 21-7, 80-2 and [Loeckx & Sieber 84], p. 113-5.

[3] See [Shapiro 92], vol. 1, p. 611-15.

> the other hand, today's heuristic theorem provers are capable of producing proofs of fairly deep theorems, but are generally so slow and unpredictable that few users have the patience and knowledge to use them effectively. It is generally agreed that when practical theorem provers are finally available they will contain both heuristic components and many decision procedures." ([Boyer & Moore 88], p. 84)

The effective use of decision procedures, therefore, requires hybrid provers capable of exploring limited decidable subdomains in as many ways as possible.

1.2.1 Exploring Decision Procedures

Besides the construction of hybrid systems, studies of the application of decision procedures to theorem proving have been traditionally concentrated in two other areas[4]. One of them concerns the complexity of decision procedures, in their full domains or in particular subclasses of formulae. The search for less complex procedures follows whenever required by a particular application. Theories targeted by such studies include Presburger arithmetic (PrA), derived from Peano arithmetic (PA) after the axioms involving multiplication are eliminated, and *strictly multiplicative arithmetic* (SMA), which corresponds to the theory of multiplication of natural numbers, without sum or successor.

The second area deals with the recognition of decidable subtheories and special decidable classes. It has mainly targeted theories relevant to the representation of programs and data structures, such as the theory of lists and the theory of arrays. New results are obtained either through the direct inspection of a theory or subclass, or by their reduction to another domain already known to be decidable: this is the case, for instance, for the extension of certain decidable quantifier-free theories by the introduction of undefined non-logical symbols. Subclasses selected for analysis are essentially those generated by context-free grammars, with emphasis on prefix classes (i.e. sets of formulae in prenex normal form whose prefixes satisfy certain restrictions, e.g. absence of universal quantifiers).

[4] See [Plaisted 90], p. 304-306.

The most relevant area for the present study, however, concerns the integration of decision procedures into (complete) theorem provers, which can be achieved through at least three loosely defined strategies. The plain *juxtaposition* of a decision procedure and a heuristic module lacks any form of communication between them, since both components operate independently. The resulting system checks whether a conjecture belongs to the decidable domain, directing it either to the corresponding procedure or to the heuristic module, as indicated in figure 1.1. The limitation of juxtaposition is its inability to stretch the use of a decision procedure beyond its domain. The application of theorem proving to program verification shows that this solution is beneficial only for a reduced number of representative verification conditions which happen to fall in the original decidable domain[5].

In the case of *cooperation*, the selection of any module in the first place does not prevent the use of the other in subsequent steps of the process, as shown in figure 1.2. Communication then is direct. As a result, the process ends up expanding the original decidable domain: in certain cases, the inference module plays an auxiliary role with respect to the decision procedures.

Example 1.2.1 *Let a hybrid prover contain two decision procedures, respectively for PrA and SMA, and a sequent calculus-based heuristic prover.*

i. Given the conjecture

$$x^2 \times y^3 = y \times (x^2 \times y^2) \ \wedge \ 2x + s(y+3) < w^2 \times z$$

since it does not belong to the domain of any of the decision procedures, it is supplied to the heuristic module. After it is decomposed into two new subproblems by the (backward) application of the $\wedge$-right rule,

$$x^2 \times y^3 = y \times (x^2 \times y^2) \qquad 2x + s(y+3) < w^2 \times z$$

the left formula can be reduced by the procedure for SMA to $\top$, thus completing this branch of the proof. Only one subproblem is left to the inference module; in

[5] See [Boyer & Moore 88].

this case, the decision procedure has an auxiliary role with respect to the heuristic component of the system.

ii. For another conjecture,

$$x^2 \times y^3 = y \times (x^2 \times y^2) \ \wedge \ 1 < 2x + s(y + 3)$$

which does not belong to a decidable domain either, the subproblems derived by the application of the $\wedge$-right rule,

$$x^2 \times y^3 = y \times (x^2 \times y^2) \qquad 1 < 2x + s(y + 3)$$

can be both reduced to $\top$ respectively by the decision procedure for SMA and PrA. The application of inference rules in this example transforms the conjecture till it can be entirely handled by the decision procedures. □

Finally, modules can be connected through *interfacing*, as in figure 1.3. Interfaces may perform a range of transformation or simplification steps before a formula is delivered to any of the components of the system. Communication between modules therefore is mediated. There is a clear link between the extension of decidable domains through interfacing and certain decision procedures which first require the reduction of a conjecture to a given normal form, as in the case of quantifier elimination.

Example 1.2.2 *Let the prover of example 1.2.1 have a simplifier based on the application of rewrite rules, and let*

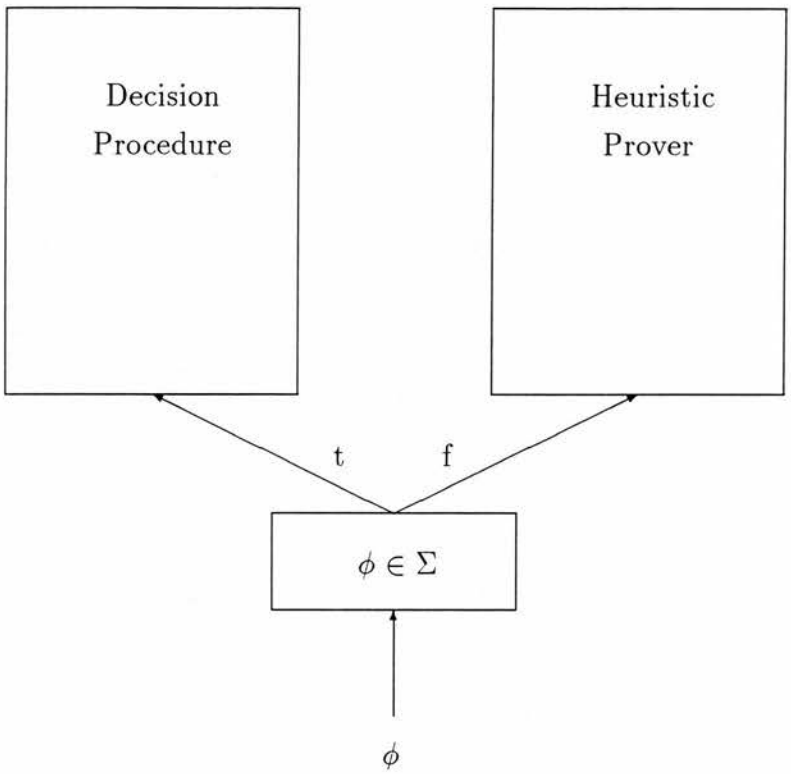
$$(x)(\exists y)(z)(w)(x^2 + (y \times z^3) = x^2 + w)$$

be a conjecture. Since it does not belong to the domain of any of the decision procedures, the simplifier is called, and, in the event that the rewrite rule

$$v_1 + v_2 = v_1 + v_3 \quad \Rightarrow \quad v_2 = v_3$$

is available, the conjecture is simplified to

$$(x)(\exists y)(z)(w)(y \times z^3 = w)$$



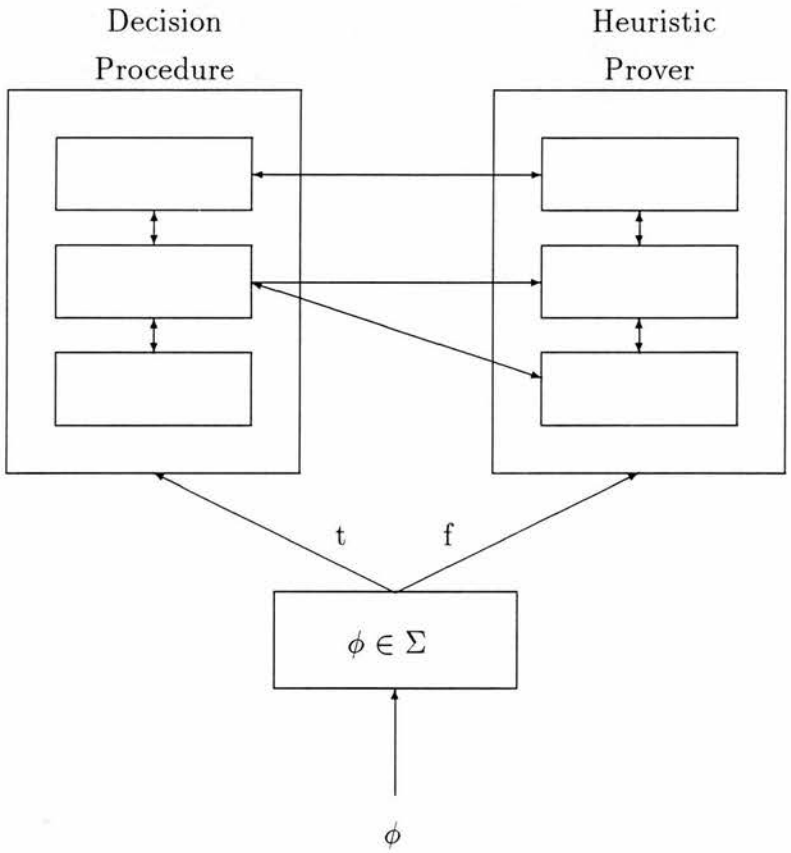
Once a formula ϕ is supplied to the hybrid system, a test is performed to establish whether it belongs to the decidable class Σ. When this is the case, ϕ is then directed to the corresponding decision procedure.

Figure 1.1: Integration of DP & HP (I) — Juxtaposition

which falls in the domain of the procedure for SMA. It is worth noting that no cooperation took place in this solution, since no inference rule of the sequent calculus was invoked. □

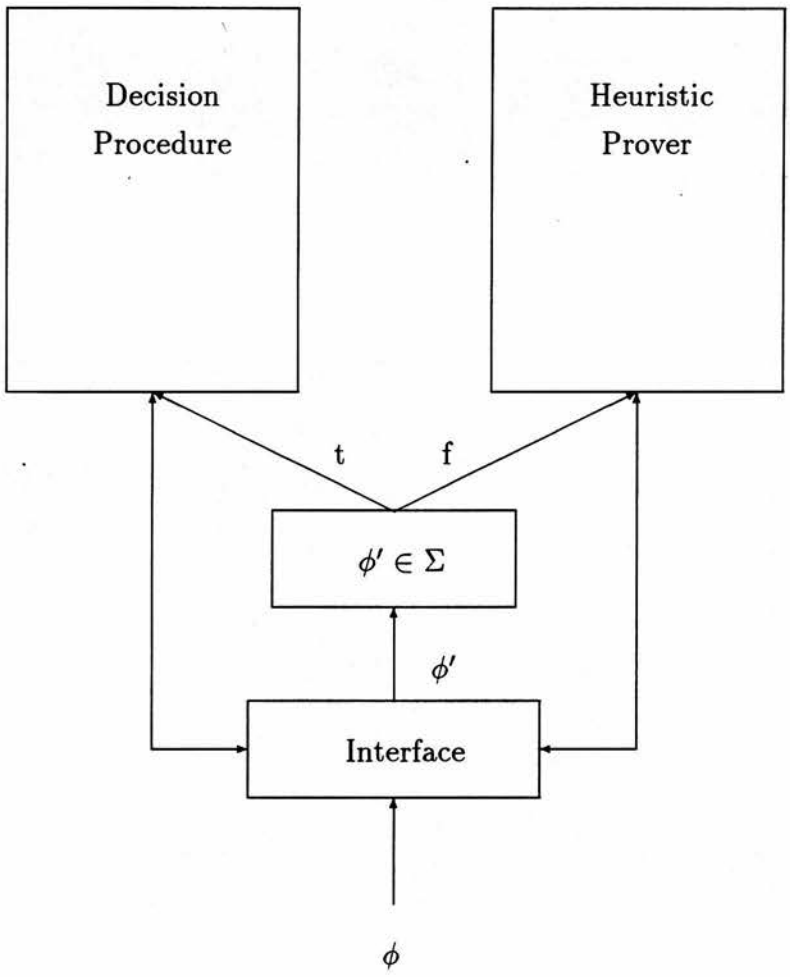
1.2.2 Extending Decidable Subclasses

From the viewpoint of a decision procedure, the main effect of its integration into a prover is the enlargement of its domain. From the viewpoint of the heuristic component, integration reduces the number of subproblems or subtasks it has to address. Both cooperation and interfacing may therefore lead to the desired expansion of a decidable domain. Cooperation can also improve the overall efficiency of the resulting system,



Σ decidable class

Figure 1.2: Integration of DP & HP (II) — Cooperation



Σ decidable class

Figure 1.3: Integration of DP & HP (III) — Interfacing

at the cost of the development of tailor-made links which requires knowledge about internal components of both heuristic prover and decision procedure. No component is dealt with as a black box. The approach usually lacks generality, since one solution is not necessarily translatable to other contexts.

The development of interfaces, on the other hand, depends only on the domain of a decision procedure, and none of its computational features has to be disclosed. Procedures therefore may be replaced without any effect on the integration mechanism, which amounts strictly to the extension of decidable classes. The main difference with respect to other studies of decidable subclass generation lies in the direction of the process: traditionally, a subclass is first chosen (e.g. the set of quantifier-free formulae, or the formulae that do not contain a particular function symbol of the underlying language) and then a procedure for its reduction to a decidable subclass is sought, whereas the new approach starts with a reduction procedure (e.g. a set of rewrite rules), and the newly delimited class, which is a fragment of the domain of the procedure, is determined thereafter.

A decision procedure for the extended class results from the combination of the initial procedure and the interface that *reduces* formulae to the original decidable domain. The approach, therefore, is *modular*: rather than requiring the complete development of new procedures, it explores existing ones, even though efficiency may suffer in some cases. A general solution for the extension of families of classes that satisfy certain syntactic properties can be formulated in the form of parameterised interfaces.

1.3 Integrated Systems

Various theorem provers operate in association with one or more decision procedures. Cooperation, for instance, takes place in the *Prototype Verification System* (*PVS*). GETFOL and the JOVIAL *program verifier*, on the other hand, have interfaces linking decision procedures to other modules. A combined approach is present in three other systems, the *Stanford Pascal verifier*, *Talzelwurm* and *Nqthm*[6].

[6] *PVS* is a system that operates both with program specification and verification, as described in [Owre *et al* 92]. GETFOL, an interactive theorem prover based on Prawitz's natural deduction, is discussed in [Giunchiglia & Traverso 91]. The JOVIAL *program verifier* was developed to prove

Two decision procedures, for ground equalities and linear inequalities, have mainly a subsidiary role in the *PVS* system. As described in section 1.2.1, cooperation can lead either to the simplification of a conjecture or to its reduction to one or more decidable subclasses. In GETFOL, an interface based on rewriting is linked to a decision procedure for a prefix class of the first-order predicate calculus. A second interface has the purpose of identifying the propositional form of conjectures, and is connected to a decision procedure for the propositional calculus. In the JOVIAL *program verifier*, the interface is responsible for the expansion of the original conjecture by the introduction of additional hypotheses, with the purpose of transforming it into an element of a decidable subclass.

The *Stanford Pascal verifier* is one of the systems where cooperation and interfacing coexist. The interface decomposes every input conjecture into subformulae, which are distributed among a series of decision procedures. Additional links are established between pairs of decision procedures for the exchange of information, when required. Cooperation also takes place in *Talzelwurm*, where decision procedures take part of the solution of subproblems. It incorporates the interface present in the *Stanford Pascal verifier* as well, and the interaction between inference rules and decision procedures is bidirectional. *Nqthm*, on the other hand, employs a decision procedure that cooperates with a *simplifier* and a *rewriter*, whereas a complex interface is responsible for the introduction of additional hypotheses. Individual components of the decision procedure are linked to individual routines of the rewriter and the simplifier. Since the final mechanism depends on the chosen procedure both extensionally and intensionally, the procedure cannot be replaced with any other, even if it applies to the same class.

None of these systems addresses the problem of integration from a global or abstract perspective. Even the solution incorporated in *Nqthm* by Boyer and Moore, who presented the integration problem in a general setting, is particular for a single context, linear arithmetic and its extensions[7], given that it relies on specific properties of these the-

---

verification conditions for programs implemented in JOVIAL, whereas the *Stanford Pascal verifier* is an interactive system for reasoning about Pascal programs: they are respectively mentioned in [Shostak 79] and [Nelson & Oppen 79]. *Talzelwurm*, whose main features are briefly examined in [Käufl 89], is a prover based on the analytic tableau method. *Nqthm*, the Boyer and Moore theorem prover, described in [Boyer & Moore 79], is a heuristic system specialised in inductive proofs.

[7] Although the solution was initially presented by Shostak, Boyer and Moore were responsible for the development of strategies to control the search present in Shostak's proposal.

ories. Other mechanisms for the extension of decidable subclasses have similar limitations, except for the interface of GETFOL, which could be applied to any theory where decidable classes have been identified[8]. The approach, however, lacks the necessary control structures that guarantee effectiveness and overall efficiency for the resulting system, since it is based on the exhaustive application of rewrite rules.

1.4 The Proof Planning Approach

Interfaces between mechanical provers and decision procedures can be abstractly modelled as effectively computable total functions whose domain and range are respectively the expanded and the original decidable classes. Provided that the transformed formula is equivalent to the original one, each of them is a theorem if and only if the other one is a theorem as well. Considering that the function is effective, the combined procedure represents a decision procedure for the extended class. The interface operates as a *normaliser*, since it transforms a formula into an equivalent expression of a pre-established class.

As mentioned in section 1.2, decidable subclasses can be extended by the application of rewrite rules. The construction of rewriting interfaces involves identifying suitable rules and control mechanisms to ensure termination and improve efficiency. Once termination is ensured for a finite set of rules, the expanded class (i.e. the domain of the rewrite system) is decidable. The continuous introduction of rules can deductively strengthen the interface and enlarge the extended class, even though the full language in which the theory is defined is never encompassed. *Proof plans for normalisation* provide the necessary tools for the construction of rewriting interface or normalisers. Plans are made up of methods and tactics, originally defined to handle inference rules. All three notions are examined in the next section.

1.4.1 Tactics

Theorems of axiomatic theories can be generated by the application of *inference rules* to axioms. Certain inference systems are both correct and complete for the representation

[8] See [Armando & Giunchiglia 93], p. 500.

of theories, i.e.

(i) any formula derived from the axioms is a theorem, and

(ii) every theorem can be generated from the axioms by the application of rules.

When the theory is undecidable, an inference system provides a semi-decision procedure for it.

Rules can be applied backwards to a conjecture until a list of axioms is obtained, whenever the formula is a theorem. Both forward and backward deduction may involve a considerable number of inference steps, which lead to large proof trees. *Tactics* allow the performance of arbitrarily large steps composed of primitive rules. Each tactic is a function that maps a formula ϕ into a list of new formulae, such that the application of inference rules to them results in ϕ. The combination of tactics defines proof structures, some of which are complete strategies for the recognition of classes of theorems (i.e. the application of a complete tactic to a formula of the class reduces it to a finite list of axioms).

The construction of a complete strategy for a conjecture involves search in a set of tactics. The task of combining them, however, can be lifted to the specification level. The specification for a tactic provided by *methods* includes properties of its domain, under the form of preconditions, and properties of the image, in the postconditions. Once the task of combining tactics is replaced by the combination of methods, there is, under certain circumstances, a contraction of the original search space. A composite method that converts a formula into a list of axioms of a theory represents a *proof plan.* Since each method specifies a tactic, a complete tactic is mechanically derivable from each plan.

1.4.2 Normalisation Planners

Under the proof planning approach, rewriting interfaces or normalisers are built from the combination of elementary normalisers or *primitive normalisation tactics.* A specification language for the description of both primitive and composite tactics simplifies the task of combining them. Elementary normalisers correspond to the basic operations that take place in the transformation of the members of a context-free generated class

into members of a similar class. They include *elimination* operators, capable of removing occurrences of symbols from the formulae of a class, and *reorganisation* operators, which impose restrictions upon relative positions of occurrences of symbols. Each operation requires a special set of rules, selected from the rewrite rule base available to the system. *Remove* rules are suitable for the elimination of occurrences of selected symbols, whereas *stratify* rules move occurrences of a symbol beneath others. These and other rules are mechanically recognisable as a result of their syntactic properties[9].

The specification language is adequate for the description of context-free generated classes. Its predicates allow the symbols that can occur in a formula of a class to be determined, together with the permissible relative positions of their occurrences. The description for a normaliser is provided by a pair of classes, its domain and image. Both elementary and composite normalisers are usually only *partially* specified, in the sense that their descriptions determine functions only partially defined in the chosen domain. As a result, any normaliser for the expansion of a decidable subclass Σ can be initially described as the partial function

$$\mathcal{N}: Fml_{\mathcal{L}} \rightsquigarrow \Sigma$$

where the extended decidable class, Σ', is a subset of $Fml_{\mathcal{L}}$, the set of formulae of the underlying language. A *planner* generates the normaliser through the gradual transformation of the initial class, $Fml_{\mathcal{L}}$, into the target class, Σ', by the composition of elementary normalisers. Given that this problem involves search in a universe of primitive operators, searching strategies have to be made available to the planner whenever efficiency has to be taken into account.

The construction of a normaliser, however, does not have to be necessarily handled by a planner. An alternative solution, still in the scope of proof planning, employs parameterised tactics or *general-purpose proof plans*, which represent families of normalisers. A normaliser for a particular extension task can be generated by the instantiation of parameters of a general-purpose plan, as part of an operation that is considerably less complex than the entire development of a normaliser by a planner. Given the description $\langle \Sigma_1, \Sigma_2 \rangle$, whereas the problem a planner has to solve is

[9] See [Bundy 91].

$$(\exists \mathcal{N})(dom(\mathcal{N}) \subseteq \Sigma_1 \ \& \ rng(\mathcal{N}) = \Sigma_2)$$

in the presence of a general-purpose plan $\mathcal{N}(v_1, \ldots, v_n)$, it suffices to find a constructive proof for

$$(\exists v_1) \ldots (\exists v_n)(dom(\mathcal{N}(v_1, \ldots, v_n)) \subseteq \Sigma_1 \ \& \ rng(\mathcal{N}(v_1, \ldots, v_n)) = \Sigma_2)$$

The relevance of the second approach derives from its ability to address families of interfacing problems in a dynamic context. Under certain circumstances, if the set of function and predicate symbols of the underlying (object) language is expanded, the rule base has to be extended but no structural change to the plans is required. Even though other approaches, such as the hypothesis introduction mechanism of *Nqthm*, can also handle new symbols, proof plans are more general, since they are applicable to any theory that admits decidable subclasses.

1.5 Applications to Peano arithmetic

One of the families of normalisers defined by means of a general-purpose plan addresses the normalisation problem defined by a pair $\langle \Sigma_1, \Sigma_2 \rangle$ where every formula ϕ in Σ_1 contains at least an occurrence of one of the symbols $S_1, \ldots, S_n$, i.e.

$$occ\_sym(S_1, \phi) \ \vee \ \ldots \ \vee \ occ\_sym(S_n, \phi)$$

whereas formulae in Σ_2 are free from occurrences of such symbols, i.e.

$$\neg occ\_sym(S_1, \phi) \ \& \ \ldots \ \& \ \neg occ\_sym(S_n, \phi)$$

The corresponding general-purpose plan then handles any normalisation problem strictly based on the elimination of symbols which are absent from the target class Σ_2.

Peano arithmetic is a possible domain of application for this plan, since this theory admits two decidable subclasses which can be obtained from the original set of formulae by the *removal* of *deviant symbols*. One of the decidable classes, $\mathcal{L}_{PrA}$, consists of all formulae ϕ where $\times$ is absent, being representable therefore as

$$\neg occ\_sym(\phi, \times)$$

The other class, $\mathcal{L}_{SMA}$, consists of formulae ψ in which the successor function and sum are both absent,

$$\neg occ\_sym(\psi, \mathrm{s}) \ \& \ \neg occ\_sym(\psi, +)$$

The general-purpose plan implemented for this purpose is a *simplifier* that operates as an interface similar to that described in figure 1.3. The simplifier examines the set of rewrite rules available to the system, selecting those able to eliminate symbols that are deviant with respect to a chosen decidable subclass. The choice of a target decidable class is influenced by heuristic functions. For instance, given the arithmetical conjecture

$$(x)(\exists y)(x^3y^2 + 1 = \mathrm{s}(xy^4))$$

heuristic measures explore the fact that this formula could be reduced to the first class, $\mathcal{L}_{PrA}$, after the removal of eight occurrences of $\times$, whereas it would require the removal of only one occurrence of $+$ and one occurrence of s to reduce it to the second class, $\mathcal{L}_{SMA}$.

1.6 Dissertation Outline

The systems mentioned in section 1.3 are further examined in two chapters. Chapter 2 presents the properties upon which their integration strategies are built, including the separate normal form lemma and the additional hypothesis introduction lemma. Chapter 3 describes how the introduction of additional hypotheses as been incorporated in *Nqthm*. Control structures for hypothesis selection are analysed along with their application to linear arithmetic.

The proof planning approach to normalisation is described in chapter 4, where *tactics*, *methods* and *proof plans* are defined. After the description of generic normalisation processes, the three notions are extended to the domain of expression normalisation. Specialised normalisers may then be constructed from the composition of tactics, specified

by methods. Chapter 5 describes two types of proof plans for normalisation, *special-purpose plans*, which are complex tactics that perform specific normalisation tasks, and *general-purpose plans*, parameterised tactics that delineate families of normalisers. The representation of two decision procedures by special-purpose plans illustrates the strength and generality of proof planning.

Proof plans are applied to the extension of decidable subclasses in chapter 6. Attention is mainly devoted to the control of the extension of decidable sublanguages, i.e. decidable classes that correspond to the set of formulae of a sublanguage. Normalisation in this context is reducible to the removal of deviant symbols. Control structures for this task include heuristic functions that assist the plan at the stage of selecting decidable classes and *remove* rules.

The elimination of disagreements between rewrite rules and conjectures is examined in chapter 7, which reviews three inference systems for the predicate calculus with equality: *E-resolution*, *RUE-resolution* and *equality graph construction*. Properties of these procedures that are relevant from the point of view of disagreement elimination are highlighted. Chapter 8 describes the *rule generation mechanism* (RGM), derived from the above difference reduction systems, which deals more efficiently with disagreement elimination in the course of rule matching, with loss of completeness.

Chapter 9 presents a group of general-purpose plans, including two simplifiers, for the extension of sublanguages. Some of these plans are interfaced to RGM. Specific features of Peano arithmetic and some of its extensions, particularly those generated by the introduction of recursive functions, are then examined. *Remove* rules for the deviant symbols with respect to two arithmetical decidable sublanguages, $\mathcal{L}_{PrA^\bullet}$ and $\mathcal{L}_{SMA}$, are incorporated in the arithmetical rule base $\mathcal{R}_{PA^\bullet}$. The termination for the application of chain-reducing rules is then proven. Some limitative results concerning the enlargement of the rule base are derived from the essential undecidability of the theory.

Empirical results related to the application of general-purpose proof plans to arithmetical conjectures are described in chapter 10. Four series of experiments have been conducted. The first series assesses the effect of the heuristic components of the plans, whose performances are compared with rewriters where one or more of such compon-

ents are absent. The second series determines the relevance of proof planning in the domain of program verification, based on the application of the simplifiers to a set of representative verification conditions. The third series involves randomly generated arithmetical formulae, which can provide a statistical mapping for the expanded decidable classes. Finally, a group of quantifier-free formulae allows the comparison of the performances of proof planned simplifiers and *Nqthm*.

The study ends with a discussion about future work and some conclusions. Chapter 11 describes some of the applications of proof planning that have not been covered in the present study, including

(i) the introduction of general *remove* rules, implication rules and *remove* procedures in the rule base,

(ii) the construction of general-purpose plans based on operations other than the strict removal of symbols, and

(iii) the direct cooperation of proof plans with semidecision procedures.

Finally, chapter 12 discusses some of the consequences of the empirical and theoretical results.

A series of appendices provides auxiliary information about the main text. Appendix A lists all the symbols used in previous chapters. Appendix B is dedicated to basic concepts related to the decision problem for first-order theories, including the notion of *effective computability*, built upon the theory of recursive functions. The decision problem for validity is formally introduced after the syntax and semantics of first-order languages are defined. Methodologies for the establishment of the decidability of a theory, such as *quantifier elimination* and *model completeness*, are also discussed.

Appendix C examines undecidable theories and the existence of recursive classes of formulae for which decision procedures can be nonetheless exhibited. Three types of such classes are identified. Appendix D defines a series of additional concepts, such as rewrite systems and implication rewrite rules, whereas appendix E contains proofs for some of the results quoted in the main text. Appendices F, G and H respectively contain the PROLOG code for the general-purpose proof plans, the arithmetical conjectures employed in the empirical stage, and the sets of arithmetical *remove* rules and

elimination equations employed by the plans.

It has to be emphasised that the reading of the appendices is not a prerequisite for the understanding of the dissertation. Appendices B to E, in particular, do not contain any original contribution, and have been included with the sole purpose of sparing the reader the effort of looking for basic definitions and proofs in the literature.

Chapter 2

Enlarging Decidable Subclasses

Once a decidable subclass for a theory has been identified, there are mechanisms which allow its extension. The upper bound for this process, which coincides with the full set of formulae of the underlying language, is unreachable if the theory is undecidable. When effectively performed, the extension of decidable domains strengthens the role of decision procedures in mechanical theorem proving.

The concept of *reduction class*, which is relevant to the study of decidable subclass extension, is examined in section 2.1. Section 2.2 describes results upon which extension mechanisms can be built. Applications of these results include studies by Armando and Giunchiglia (section 2.3), Nelson and Oppen (section 2.4), Shostak (section 2.5), Käufl (section 2.6) and Owre *et al* (section 2.7). Some of the limitations of these mechanisms are summarised in section 2.8.

2.1 Reduction Classes

Given a decidable subclass Σ for an axiomatisable first-order theory T, it is always possible to exhibit a proper extension of Σ that is also decidable for T. The least upper bound (with respect to $\subset$) for such extension that is also a recursive class is the set of formulae of the underlying language[1].

The extension of a decidable subclass is representable, from a set-theoretical point of

[1] For a proof of this result, see appendix E. First-order languages, first-order theories, recursive functions and decision procedures are defined in appendix B. Decidable classes of formulae are examined in appendix C.

view, as the union of two sets,

$$\Delta_1 \cup \Delta_2 = \Delta$$

where Δ_1 corresponds to the initial class, Δ, to the extended class, and Δ_2 is non-empty. Given that set union preserves decidability, if Δ_2 is decidable, the same applies to Δ. A trivial or *inessential* extension involves a set Δ_2 whose decidability is established independently of any property of Δ_1. As a result, the union of Δ_1 and Δ_2 does not yield any additional information.

In another case, the decidability of Δ_2 is *reduced* to the decidability of Δ_1, i.e. only a conditional proof for Δ_2 in terms of Δ_1 is exhibited. Under these circumstances, extending Δ_1 by means of the union of both sets generates additional information, since, given the conditional decidability of Δ_2, if Δ_1 is decidable, the same applies to their union[2]. Therefore, there are two basic processes related to the use of decidable subclasses.

i. The *identification* of new subclasses, an unconditional process where the decidability of a particular formula does not rely upon the establishment of the same property for any class of formulae, and

ii. The *extension* of a subclass, which relies essentially on conditional proofs in which the decidability of a class depends upon others.

This distinction reflects the main approaches to the study of the decision problem for validity in the predicate calculus, which consist of *special* and *reduction cases*[3]. The

[2] A propositional representation to this argument includes an application of *modus ponens*.

$$\frac{\text{decidable}(\Delta_1) \;\&\; (\text{decidable}(\Delta_1) \supset \text{decidable}(\Delta_2))}{\frac{\text{decidable}(\Delta_1) \;\&\; \text{decidable}(\Delta_2)}{\text{decidable}(\Delta_1 \cup \Delta_2)}}$$

[3] According to Curry, in the decision problem for validity in the predicate calculus, "two kinds of results continue to be obtained: first, solutions of special cases (such as the case where only unary predicates are present or where, in the Skolem normal form, $r \leq 2$), and second, reductions of the general problem to cases where the proposition to be investigated is of a special form". See [Curry 76], p. 358.

informal notion of reduction of a decision problem is applicable to domains other than formal theories.

Definition 2.1.1 (Reduction classes)

Let $\mathcal{U}$ be a universe set, $\mathcal{A}$ and $\mathcal{B}$, two of its subsets, and $g_{\mathcal{U}}$, a Gödel function[4] *for $\mathcal{U}$.*

i. *A* reduction class *for $\mathcal{U}$ w.r.t. $\mathcal{A}$ is a proper subclass $\mathcal{U}' \subset \mathcal{U}$ such that there is a recursive function f where $f(\ulcorner \mathcal{U} \urcorner) \subseteq \ulcorner \mathcal{U}' \urcorner$ and, for every $u \in \mathcal{U}$, $u \in \mathcal{A}$ iff $f(\ulcorner u \urcorner) \in \ulcorner \mathcal{A} \urcorner$.*

ii. *A* reduction class *for $\mathcal{B}$ w.r.t. $\mathcal{A}$ (in $\mathcal{U}$) is a proper subclass $\mathcal{B}' \subset \mathcal{B}$ such that there is a recursive function f where $f(\ulcorner \mathcal{B} \urcorner) \subseteq \ulcorner \mathcal{B}' \urcorner$ and, for every $u \in \mathcal{B}$, $u \in \mathcal{A}$ iff $f(\ulcorner u \urcorner) \in \ulcorner \mathcal{A} \urcorner$.*

In both cases, f represents a reduction function *for $\mathcal{U}$ (or $\mathcal{B}$) w.r.t. $\mathcal{A}$. In the second case, $\mathcal{B}$ is the* extended class *for $\mathcal{B}'$ w.r.t. $\mathcal{A}$.*

The relevance of reduction classes in the domain of decidability stems from the fact that, when the decision problem for $\mathcal{A}$ w.r.t. a reduction class $\mathcal{U}' \subset \mathcal{U}$ has a positive solution, represented by a recursive function h, the decision problem for $\mathcal{A}$ w.r.t. $\mathcal{U}$ also has a positive solution, h', defined as

$$h'(\ulcorner u \urcorner) = h(f(\ulcorner u \urcorner))$$

where f is a reduction function for $\mathcal{U}$ w.r.t. $\mathcal{A}$. Similarly, with respect to the second case, whenever the decision problem for $\mathcal{A}$ w.r.t. a reduction $\mathcal{B}'$ in $\mathcal{U}$ has a positive solution, represented by a recursive function h, and $\ulcorner \mathcal{B} \urcorner$ is recursive, the decision problem for A w.r.t. $\mathcal{B}$ in $\mathcal{U}$ also has a positive solution, defined in the next lemma[5].

---

[4] Gödel functions are defined in appendix B.

[5] When $\ulcorner \mathcal{B} \urcorner$ is not recursive, the decidability of the reduction class alone does not guarantee the decidability of $\mathcal{B}$ for $\mathcal{A}$. For instance, let T be an undecidable theory in $\mathcal{L}$, and let f be a recursive function such that, for all $\phi \in Fml_{\mathcal{L}}$, $f(\ulcorner \phi \urcorner) = \ulcorner \top \urcorner$. Then $\{\top\}$ is a reduction class for T with respect to T itself, and f is a reduction function for T w.r.t. T, given that

(a) $f(\ulcorner T \urcorner) \subseteq \ulcorner \{\top\} \urcorner$, and
(b) for every $\phi \in T$, $\phi \in T$ iff $f(\ulcorner \phi \urcorner) \in \ulcorner T \urcorner$ (since $f(\ulcorner \phi \urcorner) = \ulcorner \top \urcorner$).

In spite of the fact that $\{\top\}$ is decidable for T in $Fml_{\mathcal{L}}$, the same does not apply to T, since it is by hypothesis undecidable.

Lemma 2.1.1 *Let $\mathcal{A}$ be a subset of $\mathcal{U}$, and let $\mathcal{B}$ and $\mathcal{B}'$ be two subsets of $\mathcal{U}$ such that $\mathcal{B}'$ is a reduction class for $\mathcal{B}$ w.r.t. $\mathcal{A}$. Let $g_{\mathcal{U}}$ be a Gödel function for $\mathcal{U}$.*

i. If f is a reduction function for $\mathcal{B}$ w.r.t. $\mathcal{A}$, then $f(\ulcorner\mathcal{B}\urcorner) \cap \ulcorner\mathcal{A}\urcorner = f(\ulcorner\mathcal{B} \cap \mathcal{A}\urcorner)$.

ii. If $\ulcorner\mathcal{B}\urcorner$ is recursive, and $\mathcal{A}$ is decidable in $g_{\mathcal{U}}^{-1} f(\ulcorner\mathcal{B}\urcorner)$, then $\mathcal{A}$ is also decidable in $\mathcal{B}$.

PROOF. For notational simplicity, $g_{\mathcal{U}}^{-1} \circ f \circ g_{\mathcal{U}}$ will be represented as f'.

i. $u' \in f'(\mathcal{B}) \cap \mathcal{A}$ iff $u' \in f'(\mathcal{B})$ & $u' \in \mathcal{A}$ iff there is $u \in \mathcal{B}$ such that $f'(u) = u'$ & $u' \in \mathcal{A}$ iff there is $u \in \mathcal{B} \cap \mathcal{A}$ such that $f'(u) = u'$ (since $u \in \mathcal{A}$ iff $f'(u) \in \mathcal{A}$) iff $u' \in \{f'(u) \mid u \in \mathcal{B} \cap \mathcal{A}\}$ iff $u' \in f'(\mathcal{B} \cap \mathcal{A})$. Hence $f'(\mathcal{B}) \cap \mathcal{A} = f'(\mathcal{B} \cap \mathcal{A})$.

ii. Let h be a decision procedure for $\mathcal{A}$ w.r.t. $f'(\mathcal{B})$.

$$h(\ulcorner u' \urcorner) = \begin{cases} 0 & u' \notin f'(\mathcal{B}) \\ 1 & u' \in f'(\mathcal{B}) - \mathcal{A} \\ 2 & u' \in f'(\mathcal{B}) \cap \mathcal{A} \end{cases}$$

Since for every $u' \in f'(\mathcal{B})$ there is a formula $u \in \mathcal{B}$ such that $u' = f'(u)$, and considering that $f'(\mathcal{B}) \cap \mathcal{A} = f'(\mathcal{B} \cap \mathcal{A})$, then

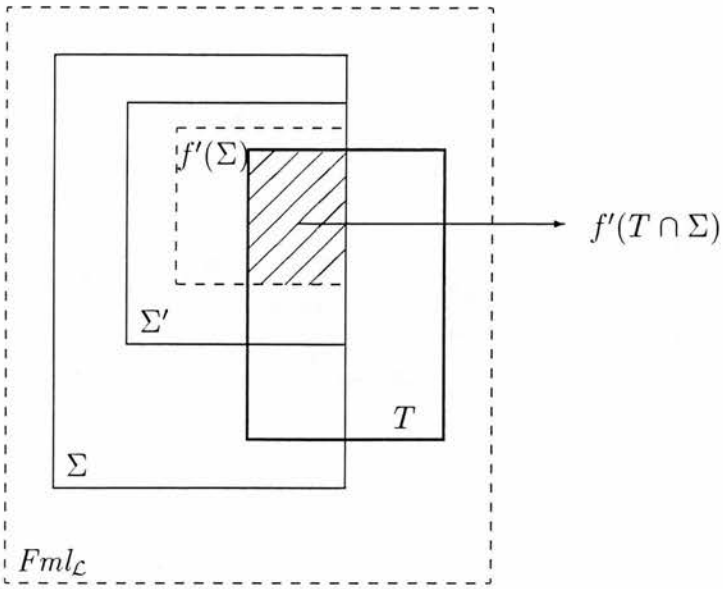
$$h(\ulcorner f'(u) \urcorner) = \begin{cases} 0 & f'(u) \notin f'(\mathcal{B}) \\ 1 & f'(u) \in f'(\mathcal{B}) - \mathcal{A} \equiv \\ & \equiv f'(u) \in f'(\mathcal{B}) - (f'(\mathcal{B}) \cap \mathcal{A}) \equiv \\ & \equiv f'(u) \in f'(\mathcal{B}) - f'(\mathcal{B} \cap \mathcal{A}) \equiv \\ & \equiv f'(u) \in f'(\mathcal{B} - \mathcal{A}) \\ 2 & f'(u) \in f'(\mathcal{B}) \cap \mathcal{A} \equiv \\ & \equiv f'(u) \in f'(\mathcal{B} \cap \mathcal{A}) \end{cases}$$

Given that $\ulcorner\mathcal{B}\urcorner$ is recursive, there is a recursive function $l_{\mathcal{B}}$ such that

$$l_{\mathcal{B}}(\ulcorner u \urcorner) = \begin{cases} 0 & u \notin \mathcal{B} \\ 1 & \text{otherwise} \end{cases}$$

Considering that $f'(u) \notin f'(\mathcal{B})$ implies $u \notin \mathcal{B}$, then

$$l_{\mathcal{B}}(\ulcorner u \urcorner) \times h(\ulcorner f'(u) \urcorner) = \begin{cases} 0 & u \notin \mathcal{B} \vee f'(u) \notin f'(\mathcal{B}) \equiv \\ & \equiv u \notin \mathcal{B} \\ 1 & u \in \mathcal{B} \wedge f'(u) \in f'(\mathcal{B} - \mathcal{A}) \equiv \\ & \equiv u \in \mathcal{B} - \mathcal{A} \\ 2 & u \in \mathcal{B} \wedge f'(u) \in f'(\mathcal{B} \cap \mathcal{A}) \equiv \\ & \equiv u \in \mathcal{B} \cap \mathcal{A} \end{cases}$$



Σ' reduction class for Σ
f reduction function
f' $g_{\mathcal{L}}^{-1} \circ f \circ g_{\mathcal{L}}$

$$f'(\Sigma) \cap T = f'(\Sigma \cap T)$$

Figure 2.1: Reduction Classes w.r.t. Theories

Hence $\mathcal{A}$ is decidable in $\mathcal{B}$, and $\lambda u(l_{\mathcal{B}}(\ulcorner u \urcorner) \times h(\ulcorner f'(u) \urcorner))$ is a decision procedure for $\mathcal{A}$ w.r.t. $\mathcal{B}$. ∎

Hence, it suffices to guarantee that $\mathcal{A}$ is decidable in $f'(\mathcal{B})$ (and not necessarily in $\mathcal{B}'$) to ensure that $\mathcal{A}$ is also decidable in $\mathcal{B}$. The application of this result to first-order theories is illustrated in figure 2.1. Some of the mechanisms that can be employed in the identification of reduction classes are discussed in the next section.

2.2 Reduction Transformations

When applied to the domain of decidable subclasses for theories, definition 2.1.1 unfolds into two cases. The first one involves reduction classes for $Fml_{\mathcal{L}}$ w.r.t. a theory T, as

well as reduction functions that generate, for every formula ϕ in $Fml_{\mathcal{L}}$, an element ϕ' of a reduction class such that ϕ and ϕ' are *equi-valid in* T, i.e.

$$\phi \in T \quad \text{iff} \quad \phi' \in T$$

In the second case, reduction classes are defined for a given subclass Σ w.r.t. T, where $\Sigma \subset Fml_{\mathcal{L}}$. A reduction function then corresponds to an effective mechanism, in the form of a recursive function, that generates, for every element $\phi \in \Sigma$, an element ϕ' of a reduction class such that ϕ and ϕ' are also equi-valid in T.

Example 2.2.1 *Let $\mathcal{U}$ be the set of formulae of a language $\mathcal{L}$.*

i. Let $T_0^{\mathcal{L}}$ be a subset $\mathcal{A}$ of the universe set.

(a) The class of prenex normal form formulae of $\mathcal{L}$ is a reduction class for $Fml_{\mathcal{L}}$ w.r.t. $T_0^{\mathcal{L}}$, considering that an element of $Fml_{\mathcal{L}}$ is valid (i.e. it is an element of $T_0^{\mathcal{L}}$) if and only if the corresponding reduced prenex formula is also valid. Any effective mechanism for generating a prenex normal form formula from an element of $Fml_{\mathcal{L}}$ is a reduction function for $Fml_{\mathcal{L}}$ w.r.t. $T_0^{\mathcal{L}}$.

(b) Let $\mathcal{B}$ be the class of prenex formulae of $\mathcal{L}$. Then the set of formulae in prenex conjunctive normal form *is a reduction class for $\mathcal{B}$ w.r.t. $T_0^{\mathcal{L}}$, for a formula in prenex form is valid (i.e. it belongs to $T_0^{\mathcal{L}}$) iff the corresponding formula in prenex conjunctive normal form is also valid. Any effective mechanism for generating a prenex conjunctive normal form formula from an element of $\mathcal{B}$ is a reduction function for $\mathcal{B}$ w.r.t. $T_0^{\mathcal{L}}$.*

ii. Given a decidable theory T in $\mathcal{L}$, if T is negation complete, $\{\top, \bot\}$ is a reduction class for $Stn_{\mathcal{L}}$ w.r.t. T, since any sentence of $\mathcal{L}$ is T-equivalent to either $\top$ or $\bot$. Any decision procedure for T is a reduction function for $Stn_{\mathcal{L}}$ w.r.t. T. □

These examples substantiate the claim that any reduction process in the context of formal languages is a special case of formula normalisation. They also make clear that the first reduction case is relevant only for decidable theories, since, once the decidability of a theory in the reduction class is established, its decidability in the full

language follows. For undecidable theories, not only there is no effective mechanism for such a reduction, but moreover there are formulae for which the reduction to a decidable class is impossible[6].

Considering the first reduction case, reduction classes can be exhibited, for instance, for any theory that admits quantifier elimination[7]: in particular, if the set of quantifier-free formulae is decidable for T, then T is decidable in $Fml_{\mathcal{L}}$. The second case, on the other hand, has a role in both decidable and undecidable theories: once the reduction class Σ' has been proven decidable, the initial class $\Sigma \subset Fml_{\mathcal{L}}$ becomes an *extended decidable class*. When reduction and extended classes are decidable, they have the same type, i.e. they must be both composed either solely of T-theorems, or of T-non-theorems, or of both theorems and non-theorems. This property can be justified by the fact that reduction preserves T-validity[8].

There are at least two classes of validity-preserving transformations, one based on equivalence and the other on strict equi-validity. There are also weaker forms of transformations which can generate certain reduction classes, provided that additional restrictions are met.

2.2.1 Equivalence

An effective formula transformation process in a theory T is *equivalence-based* iff

$$T \models \phi \equiv \phi'$$

In this case, ϕ and ϕ' are equi-valid in T. *Quantifier elimination* provides once again an example for this class of transformations, since every formula of the underlying language of a theory is linked to an equivalent quantifier-free formula. Another process that falls in this group consists of the application of *rewrite rules*. Since reduction functions have to be effective, the chosen set of rules has to be finite, noetherian and

[6] The connection between normalisation and formula reduction is examined in chapter 4. The existence of formulae that cannot be reduced into a decidable subclass is established in appendix E.

[7] *Elimination of quantifiers* is defined in appendix B.3.2.

[8] If Σ' has type II (i.e. it contains both theorems and non-theorems), as it is a subset of Σ, then Σ must also be of type II. If Σ' has type I or III, all formulae in Σ are respectively either T-valid or T-invalid, having therefore type I or III as well.

canonical, as defined below[9].

Definition 2.2.1 (Properties of Rewrite Sets)

Let $\mathcal{R}$ be a set of rewrite rules, and let $\epsilon_1, \ldots, \epsilon_4$ be expressions of a language $\mathcal{L}$.

i. ϵ_1 *and* ϵ_2 *are* similar under $\mathcal{R}$ *iff*

(a) $\epsilon_1 \overset{\mathcal{R}}{\Rightarrow} \epsilon_2$ *or* $\epsilon_2 \overset{\mathcal{R}}{\Rightarrow} \epsilon_1$, *or*

(b) there is an expression ϵ_3 *such that both pairs,* ϵ_1, ϵ_3 *and* ϵ_2, ϵ_3, *are similar under* $\mathcal{R}$.

ii. $\mathcal{R}$ *is* noetherian *iff every rewriting sequence for* $\mathcal{R}$ *is finite.*

iii. $\mathcal{R}$ *is* canonical *iff, whenever* ϵ_2 *and* ϵ_3 *are normal forms for* ϵ_1, *then* $\epsilon_2 \overset{s}{=} \epsilon_3$.

iv. $\mathcal{R}$ *is* Church-Rosser *iff, for every pair of similar expressions,* ϵ_1 *and* ϵ_2, *there is an expression* ϵ_3 *such that* $\epsilon_1 \overset{\mathcal{R}}{\Rightarrow} \epsilon_3$ *and* $\epsilon_2 \overset{\mathcal{R}}{\Rightarrow} \epsilon_3$.

v. $\mathcal{R}$ *is* locally confluent *iff, for any two rewriting sequences of the form* $\epsilon_1 \overset{R}{\Rightarrow} \epsilon_2$ *and* $\epsilon_1 \overset{R'}{\Rightarrow} \epsilon_3$, *where* $\epsilon_2 \overset{s}{\neq} \epsilon_3$ *and* $R, R' \in \mathcal{R}$, *there are additional sequences of the form* $\epsilon_2 \overset{\mathcal{R}}{\Rightarrow} \epsilon_4$ *and* $\epsilon_3 \overset{\mathcal{R}}{\Rightarrow} \epsilon_4$.

vi. $\mathcal{R}$ *is* confluent *iff, for any two rewriting sequences of the form* $\epsilon_1 \overset{\mathcal{R}}{\Rightarrow} \epsilon_2$ *and* $\epsilon_1 \overset{\mathcal{R}}{\Rightarrow} \epsilon_3$, *where* $\epsilon_2 \overset{s}{\neq} \epsilon_3$, *there are additional sequences of the form* $\epsilon_2 \overset{\mathcal{R}}{\Rightarrow} \epsilon_4$ *and* $\epsilon_3 \overset{\mathcal{R}}{\Rightarrow} \epsilon_4$.

Canonical sets compute a single expression for any element of the domain, and, as a result, denote (partial) functions. Finite and noetherian sets are guaranteed to halt for every expression of their domain, being therefore total (i.e. defined for every element of the domain). Hence a finite, noetherian and canonical set of rewrite rules represents a total (effective) function.

A final case is based on the expansion of a conjecture by the introduction of a T-valid antecedent, as justified in the lemma that follows.

[9] Rewrite rules are defined in appendix D. Properties of rewrite sets are discussed, for instance, in [Bundy 83], p. 153-5.

Lemma 2.2.1 (Additional Hypothesis Introduction)

Let T and T' be theories in $\mathcal{L}$ such that T' is an extension of T, and let $\{\phi_1, \ldots, \phi_n\} \subset T'$. If $\psi \in Fml_{\mathcal{L}}$, then

i. $T' \models \psi \equiv (\bigwedge_{i=1}^{n} \phi_i) \supset \psi$

ii. If $T \models (\bigwedge_{i=1}^{n} \phi_i) \supset \psi$, *then* $\psi \in T'$.

PROOF.

i. Since $\phi_i \in T'$, the original problem can be reduced to $T' \models \psi \equiv (\top \supset \psi)$, i.e. $T' \models \psi \equiv \psi$, which is obviously the case.

ii. If $T \models (\bigwedge_{i=1}^{n} \phi_i) \supset \psi$, then, since $T \subset T'$, $((\bigwedge_{i=1}^{n} \phi_i) \supset \psi) \in T'$. Given that $((\bigwedge_{i=1}^{n} \phi_i) \supset \psi) \equiv (\phi_1 \supset \cdots \supset \phi_n \supset \psi)$, after n applications of *modus ponens*, it follows that $\psi \in T'$. ∎

Given a theory T' in $\mathcal{L}$, a finite subset $\Phi = \{\phi_1, \ldots, \phi_n\}$ of T' and a decidable subtheory $T \subseteq T'$, the above lemma provides the justification for the construction of a class of formulae Σ that contains T, as follows:

(a) given a formula $\phi \in Fml_{\mathcal{L}}$, all the elements of Φ are conjoined to form the antecedent of a transformed formula, ϕ',

$$\phi' \stackrel{s}{=} ((\bigwedge_{i=1}^{n} \phi_i) \supset \phi)$$

and

(b) each formula ϕ' is supplied to a decision procedure for T; those which are elements of T are collected into Σ, which can then be defined as[10]

[10] A slightly modified version of the mechanism that builds Σ may clarify the principle behind its construction. Each formula ϕ of $\mathcal{L}$ is first supplied to a decision procedure for T. If it is a theorem, it is added to Σ, otherwise additional hypotheses are introduced as a new condition. The resulting formula, ϕ', is then submitted to the same decision procedure: if it belongs to T, ϕ is finally included in Σ, otherwise it is abandoned, and the process restarts from another element of $Fml_{\mathcal{L}}$. Considering that T is a subtheory of T', and in view of lemma 2.2.1, every element of Σ is a theorem of T'.

$$\Sigma \;=\; T \cup \{\phi \mid ((\bigwedge_{i=1}^{n} \phi_i) \supset \phi) \in T\}$$

Let g_Φ be a recursive function that represents the process of appending the conjunction of the formulae of Φ as antecedent to a formula ϕ, i.e.

$$g_\Phi(\ulcorner\phi\urcorner) \;=\; \ulcorner(\bigwedge_{i=1}^{n} \phi_i) \supset \phi\urcorner$$

and let h be a decision procedure for T. Then Σ can be also represented as

$$\Sigma \;=\; \{\phi \in Fml_{\mathcal{L}} \mid h(\ulcorner\phi\urcorner) = 1 \vee h(g_\Phi(\ulcorner\phi\urcorner)) = 1\}$$

It is then possible to show that (i) $\ulcorner\Sigma\urcorner$ is recursive and (ii) T is a reduction class for Σ (or Σ is an extended class for T) w.r.t. T'. Since T is a decidable subtheory of T', Σ then is a decidable class of type I for T'. A peculiar feature of the extended class is the fact that not only its decidability, but also its recursiveness (i.e. the fact that $\ulcorner\Sigma\urcorner$ is a recursive set of formulae of $\mathcal{L}$) depends upon the decidability of T, the initial class[11].

When Φ is an infinite set of T'-theorems, for each formula $\phi \in Fml_{\mathcal{L}}$, there is an infinite number of possible transformed formulae, generated by the introduction of every finite sequence of elements of Φ as antecedent. Two alternative definitions for g_Φ are then possible:

(a) a selection strategy that takes into account properties of ϕ could choose a finite subset of Φ, and a single transformed formula is obtained, or

(b) a finite set of transformed formulae is generated instead, in which case g_Φ links ϕ to a list, $[\phi^1, \ldots, \phi^m]$.

As discussed in chapter 3, the second case is more common in applications of this mechanism to theorem proving. Another use for the additional hypothesis introduction lemma is examined in section 2.5.

[11] Properties (i) and (ii) above are proven in appendix E. It has to be recalled that the possibility of reduction of a class Σ to a decidable subclass for a theory T does not necessarily amount to the decidability of Σ for T, for it is also necessary to establish that $\ulcorner\Sigma\urcorner$ is recursive.

2.2.2 Equi-validity

There is a group of formula transformation processes that are not equivalence-based but nonetheless preserve validity. Certain theorems guarantee their existence, particularly for theories with undefined symbols.

Definition 2.2.2 *Given an axiomatisable theory T in $\mathcal{L}$, a non-logical symbol S of $\mathcal{L}$ is* undefined *in T iff T admits an axiom set Δ such that S does not occur in Δ.*

Theorem 2.2.1 [Shostak 79] *Let T be a theory in $\mathcal{L}$, $\mathcal{L}'$, an expansion of $\mathcal{L}$, and T', an extension of T to $\mathcal{L}'$ such that T and T' admit a common axiom set.*

i. If ϕ is a quantifier-free formula of $\mathcal{L}'$, there is a quantifier-free formula $\widehat{\phi}$ in $\mathcal{L}$ such that

$$T' \models \phi \quad \textit{iff} \quad T' \models \widehat{\phi}$$

ii. If T is decidable in the class of quantifier-free formulae of $\mathcal{L}$, then T' is also decidable in the class of quantifier-free formulae of $\mathcal{L}'$.

PROOF. Let ϕ be a quantifier-free formula of $\mathcal{L}'$.

i. If $\phi \in Fml_{\mathcal{L}}$, then, since $Fml_{\mathcal{L}} \subset Fml_{\mathcal{L}'}$, $\widehat{\phi} \stackrel{s}{=} \phi$.

Otherwise ϕ must contain at least an occurrence of a non-logical symbol that does not belong to $\mathcal{L}$. It is possible to reduce ϕ to an equi-valid formula of $Fml_{\mathcal{L}}$ as follows.

(a) For each predicate symbol p in ϕ that does not belong to $\mathcal{L}$, let f_p be a function symbol that does not belong to $\mathcal{L}'$. Each atom $p(t_1, \ldots, t_n)$ in ϕ is replaced by the equation

$$f_p(t_1, \ldots, t_n) = c$$

where c is a constant symbol of $\mathcal{L}'$ (if none is available there, $\mathcal{L}'$ can be further expanded to include one).

(b) Let ϕ' be the formula freed from occurrences of predicate letters that do not belong to $\mathcal{L}$. For each function symbol f in ϕ that does not belong to $\mathcal{L}$, if f is the dominant symbol of n distinct subterms of ϕ, $n > 1$, for each pair $f(t_1, \ldots, t_m)$ and $f(u_1, \ldots, u_m)$ of such distinct subterms, let ψ_f^i be the formula

$$t_1 = u_1 \wedge \cdots \wedge t_m = u_m \quad \supset \quad f(t_1, \ldots, t_m) = f(u_1, \ldots, u_m)$$

where $1 \leq i \leq k, k = \frac{n(n-1)}{2}$. Each new formula is appended as antecedent to ϕ', so generating

$$\psi_f^1 \supset \cdots \supset \psi_f^k \supset \phi'$$

(c) If ϕ'' is the result of the above step, then ϕ''' is generated from ϕ'' by the replacement of each occurrence of a topmost composite term dominated by a function symbol not in $\mathcal{L}$ by a variable that does not occur in ϕ'' (i.e. only terms and subterms dominated by such function symbols are replaced by variables). This process is repeated until all the function symbols that do not belong to $\mathcal{L}$ have been eliminated.

(d) Each constant symbol c in ϕ''' that does not belong to $\mathcal{L}$ is replaced by a variable that does not occur in ϕ'''.

ii. Since T and T' share a common axiom set, according to lemma B.2.3 iii, T' is a conservative extension of T. As a result, if T is decidable in the class of quantifier-free formulae of $\mathcal{L}$, according to theorem C.5.2, T' is also decidable in the class of quantifier-free formulae of $\mathcal{L}$.

The mechanism described in theorem 2.2.1 above effectively generates, for every quantifier-free formula $\phi \in Fml_{\mathcal{L}'}$, a formula $\widehat{\phi} \in Fml_{\mathcal{L}}$ such that $\widehat{\phi}$ is valid in T' iff ϕ is valid in the same theory[12]. Then according to lemma 2.1.1, as T' is decidable in the quantifier-free class of $\mathcal{L}$, it must also be decidable in the quantifier-free class of $\mathcal{L}'$. ∎

[12] See [Shostak 79], p. 353-4.

A reduction function for the class of quantifier-free formulae of $\mathcal{L}'$ w.r.t. T' can be defined from the above theorem as $f(\ulcorner\phi\urcorner) = \ulcorner\widehat{\phi}\urcorner$. If $\phi \in Fml_{\mathcal{L}'}$ is quantifier-free, then $f(\ulcorner\phi\urcorner)$ is the Gödel number of a quantifier-free formula of $\mathcal{L}$. Also, $\phi \in T'$ iff $\ulcorner\widehat{\phi}\urcorner \in \ulcorner T'\urcorner$.

Example 2.2.2 *To illustrate the application of theorem 2.2.1 to Presburger arithmetic (PrA), let* ϕ

$$((p(z) \supset z = 1) \wedge g(y) = z + 4) \quad \supset \quad (f(g(y)) = f(3 + 2z) \vee \neg p(1))$$

be a formula of the expanded language $\mathcal{L}^*_{PrA} = \{0, 1, s, +, f, g, p\}$. *The generation of an equi-valid formula in PrA has to follow the steps described above*[13].

i. Elimination of new predicate symbols.

$$((f_p(z) = 0 \supset z = 1) \wedge g(y) = z + 4) \quad \supset \quad (f(g(y)) = f(3 + 2z) \vee f_p(1) \neq 0)$$

ii. Elimination of function symbols

(a) Identification of the new antecedent

$$\begin{array}{rcl} z = 1 & \supset & f_p(z) = f_p(1) \\ g(y) = 3 + 2z & \supset & f(g(y)) = f(3 + 2z) \end{array}$$

(b) Introduction of a new antecedent

$$\begin{array}{c} (z = 1 \supset f_p(z) = f_p(1)) \wedge (g(y) = 3 + 2z \supset f(g(y)) = f(3 + 2z)) \\ \supset \\ ((f_p(z) = 0 \supset z = 1) \wedge g(y) = z + 4) \supset (f(g(y)) = f(3 + 2z) \vee f_p(1) \neq 0) \end{array}$$

(c) Replacement of composite terms by new variables

$$\begin{array}{c} (z = 1 \supset x_1 = x_2) \wedge (x_3 = 3 + 2z \supset x_4 = x_5) \\ \supset \\ ((x_1 = 0 \supset z = 1) \wedge x_3 = z + 4) \supset (x_4 = x_5 \vee x_2 \neq 0) \end{array}$$

□

[13] This example has been extracted from [Shostak 79], p. 354. Presburger arithmetic is defined in appendix B.

Theorem 2.2.1 guarantees that, for every theory in $\mathcal{L}$ that admits as a decidable subclass the set of quantifier-free formulae of a sublanguage of $\mathcal{L}$, the decidable subclass can be expanded to include the undefined constant, function and predicate symbols of $\mathcal{L}$. Such extended decidable classes play an important role in the theorem prover examined in section 2.5. Another result that defines a validity-based transformation for the extension of combination of decidable subclasses is presented in section 2.4.

2.2.3 Weaker Transformations

Given that there are formula transformations where validity is not preserved, the decidability of the reduction class does not guarantee the decidability of the extended class. Nevertheless, it is still possible to effectively identify a partial extension of the original decidable subclass, whose boundaries depend on the logical status of the transformation performed. The notion of *weak reduction class* covers such cases.

Definition 2.2.3 (Weak reduction classes)

Let $\mathcal{U}$ be a universe set, and let $\mathcal{A}$ and $\mathcal{B}$ be two of its subsets.

i. *A* weak reduction class *for $\mathcal{U}$ w.r.t. $\mathcal{A}$ is a proper subclass $\mathcal{U}' \subset \mathcal{U}$ such that there is a recursive function f where $f(\ulcorner \mathcal{U} \urcorner) \subseteq \ulcorner \mathcal{U}' \urcorner$ and, for every $u \in \mathcal{U}$, if $f(\ulcorner u \urcorner) \in \ulcorner \mathcal{A} \urcorner$, then $u \in \mathcal{A}$.*

ii. *A* weak reduction class *for $\mathcal{B}$ w.r.t. $\mathcal{A}$ (in $\mathcal{U}$) is a proper subclass $\mathcal{B}' \subset \mathcal{B}$ such that there is a recursive function f where $f(\ulcorner \mathcal{B} \urcorner) \subseteq \ulcorner \mathcal{B}' \urcorner$ and, for every $u \in \mathcal{B}$, if $f(\ulcorner u \urcorner) \in \ulcorner \mathcal{A} \urcorner$, then $u \in \mathcal{A}$.*

In both cases, f represents a weak reduction function *for $\mathcal{U}$ (or $\mathcal{B}$) w.r.t. $\mathcal{A}$.*

When compared to the definition of reduction classes, its weak counterpart does not require that every element of $\mathcal{A}$ is transformed by the reduction function into another element of the same set: there is only the guarantee that, if an element u of the universe is reduced to $\mathcal{A}$, then u must also belong to $\mathcal{A}$. When $\mathcal{A}$ corresponds to a theory T, a

weak transformation does not preserve validity in T. If ϕ is the original formula and ϕ' is its transformed version, two conditions that do not amount to equi-validity are

i. $T \models \phi' \supset \phi$

ii. $T \models \phi'$ implies $T \models \phi$

The first case includes transformations based on the application of implication rewrite rules, and the second one, the use of inference rules[14]. The next sections examine five distinct systems that apply either strong or weak transformation mechanisms to the extension of decidable classes.

2.3 Subclass Extension in the Predicate Calculus

Armando and Giunchiglia incorporated in GETFOL a rewriting mechanism that extends the scope of application of decision procedures. GETFOL is an interactive theorem prover for the first-order predicate calculus with equality, based on the FOL system. It operates in any first-order language, using a variation of Prawitz's natural deduction as inference mechanism[15].

The extended decider in GETFOL is composed of a hierarchy of rewriters and decision procedures for subclasses of the first-order predicate calculus. It includes `ptaut`, a propositional decider built upon the principle of partial assignments, and `ptauteq`, which deals with the subclass of quantifier-free formulae with equality but without function symbols. Decision procedures for other subclasses are obtained once three rewriters are used to convert formulae to the domain of `ptaut` or `ptauteq`. The first one, `UE-dec`, computes the Herbrand expansion of a formula, in three stages. Negations are stratified over conjunctions and disjunctions by the application of de Morgan's laws, followed by the elimination of double negation,

[14] Even though property (i) implies property (ii), the converse is not valid. For instance, whereas for any theory T in $\mathcal{L}_{PA}$, $T \models x = 0$ implies $T \models y = 0$, it is not the case that $PA \models x = 0 \supset y = 0$, where PA stands for *Peano arithmetic*, a theory defined in appendix B. The use of implication rules in the extension of decidable subclasses is discussed in chapter 11. Regarding inference rules, see sections 2.3 and 2.6.

[15] The integration of decision procedures in GETFOL is described in [Armando & Giunchiglia 93]. FOL has been originally implemented by R. Weyhrauch, whereas GETFOL was developed by F. Giunchiglia; see [Weyhrauch 77] and [Giunchiglia & Traverso 91].

$$\begin{aligned} \neg(\phi \wedge \psi) &\equiv \neg\phi \vee \neg\psi \\ \neg(\phi \vee \psi) &\equiv \neg\phi \wedge \neg\psi \\ \neg\neg\phi &\equiv \phi \end{aligned}$$

Existential quantifiers are next removed by means of skolemisation. The final stage computes the partitioned Herbrand expansion for the class of UE-formulae in canonical form. The combination of `UE-dec` and the decision procedures results in a decider for the UE-subclass, which is made up of formulae where no existential quantifier $(\exists v)$ is applied to a subformula where v has a free occurrence in the scope of a universal quantifier.

Definition 2.3.1. (UE-formulae)

The class of UE-formulae (*or* formulae in UE-form) *of a language $\mathcal{L}$ is defined as follows.*

i. Any universal, existential or quantifier-free formula of $\mathcal{L}$ is in UE-form.

ii. If $\phi[\not{v}]$ is in UE-form, then $(\exists v)\phi[\not{v}]$ is also in UE-form.

iii. If ϕ_1 and ϕ_2 are in UE-form, so are $\neg\phi_1$, $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$, $\phi_1 \supset \phi_2$, $\phi_1 \equiv \phi_2$, $(v)\phi_1$.

The second rewriter, `tautren`, maps a conjecture into its propositional form (modulo bound variable renaming): the validity of the propositional form then entails the validity of the conjecture. Finally, `reduce` applies equivalence-based rewrite rules for UE-normal forming, thus enlarging the UE-subclass. The domain of `reduce`, for this reason, is not the full subjacent language, since the predicate calculus is undecidable. The interrelation of the five main modules is described in figure 2.2.

The set $\mathcal{R}$ of rewrite rules available to `reduce` is listed in table 2.1. The syntactic conditions imposed upon their application involve the notions of *top normalisable*, *normalised* and *minimal formula*, and guarantee that the set is noetherian and confluent[16]. As a result, the function

[16] See [Armando & Giunchiglia 93], p. 493-6. Normalised, top normalisable and minimal formulae are defined on p. 493.

	Rules			Conditions
R_1	$(Qv)\rho[\not v]$	$\Rightarrow$	$\rho[\not v]$	–
R_2	$(\exists v)(\phi \vee \psi)[v]$	$\Rightarrow$	$(\exists v)\phi \vee (\exists v)\psi$	–
R_3	$(\forall v)(\phi \wedge \psi)[v]$	$\Rightarrow$	$(\forall v)\phi \wedge (\forall v)\psi$	–
R_4	$(\exists v)(\rho[\not v] \wedge \phi[v])$	$\Rightarrow$	$\rho[\not v] \wedge (\exists v)\phi[v]$	–
R_5	$(\forall v)(\rho[\not v] \vee \phi[v])$	$\Rightarrow$	$\rho[\not v] \vee (\forall v)\phi[v]$	–
R_6	$\phi[v] \circ \rho[\not v]$	$\Rightarrow$	$\rho[\not v] \circ \phi[v]$	lhs top normalisable
R_7	$(\rho[\not v] \circ \phi[v]) \circ \psi[v]$	$\Rightarrow$	$\rho[\not v] \circ (\phi[v] \circ \psi[v])$	lhs top normalisable
R_8	$(\phi \vee \psi)[v] \wedge \gamma[v]$	$\Rightarrow$	$(\phi \wedge \gamma[v]) \vee (\psi \wedge \gamma[v])$	lhs top normalisable
R_9	$(\phi \wedge \psi)[v] \vee \gamma[v]$	$\Rightarrow$	$(\phi \vee \gamma[v]) \wedge (\psi \vee \gamma[v])$	lhs top normalisable
R_{10}	$\phi[v] \wedge (\rho[\not v] \wedge \psi[v])$	$\Rightarrow$	$\rho[\not v] \wedge (\phi[v] \wedge \psi[v])$	lhs top normalisable $\phi[v]$ minimal w.r.t. $\langle \exists, v \rangle$
R_{11}	$\phi[v] \vee (\rho[\not v] \vee \psi[v])$	$\Rightarrow$	$\rho[\not v] \vee (\phi[v] \vee \psi[v])$	lhs top normalisable $\phi[v]$ minimal w.r.t. $\langle \forall, v \rangle$
R_{12}	$\gamma[v] \wedge (\phi \vee \psi)[v]$	$\Rightarrow$	$(\gamma[v] \wedge \phi) \vee (\gamma[v] \wedge \psi)$	lhs top normalisable $\phi[v]$ minimal w.r.t. $\langle \exists, v \rangle$
R_{13}	$\gamma[v] \vee (\phi \wedge \psi)[v]$	$\Rightarrow$	$(\gamma[v] \vee \phi) \wedge (\gamma[v] \vee \psi)$	lhs top normalisable $\gamma[v]$ minimal w.r.t. $\langle \forall, v \rangle$

Q $\forall$ or $\exists$
$\circ$ $\vee$ or $\wedge$

Table 2.1: Rewrite Rules for *reduce*

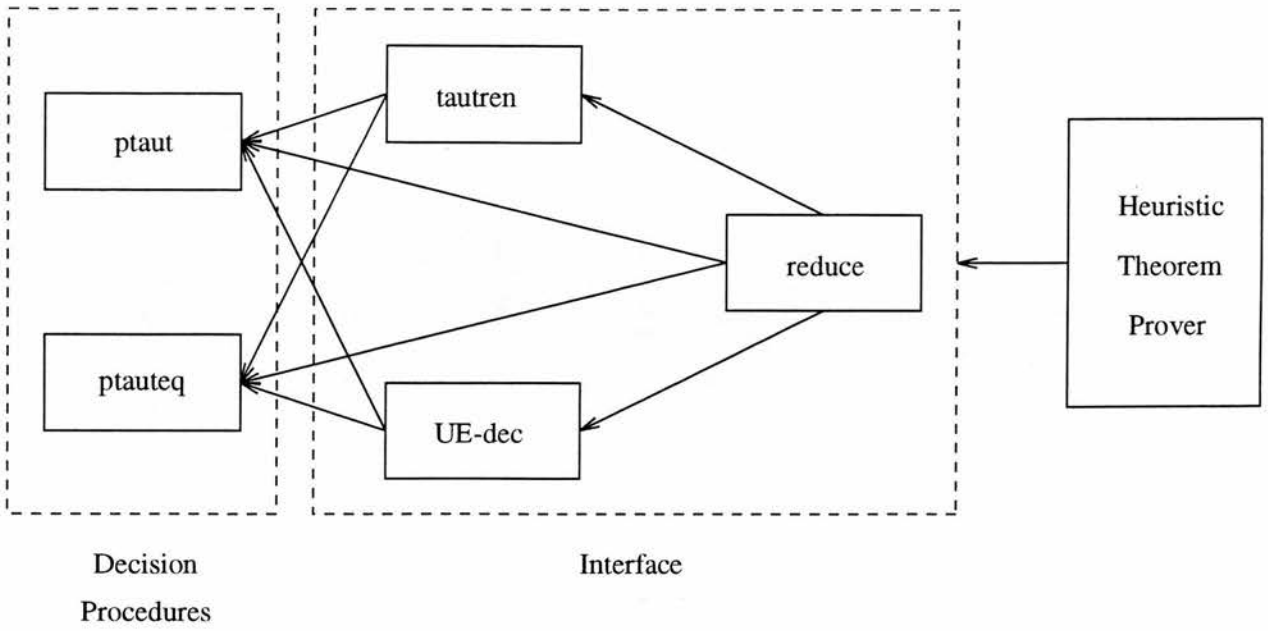


Figure 2.2: Decision Procedures in GETFOL

$$f(\ulcorner\phi\urcorner) = \begin{cases} \ulcorner\phi'\urcorner & \text{if } \phi \overset{\mathcal{R}}{\Rightarrow} \phi' \text{ and } \phi' \text{ is in UE-form} \\ 0 & \text{otherwise} \end{cases}$$

is recursive. These properties, in conjunction with the fact that the number of rules is small, make exhaustive rewriting feasible. The class

$$\Sigma \;=\; \{\phi \in Fml_{\mathcal{L}} \mid f(\ulcorner\phi\urcorner) \neq 0\}$$

is the extension of the UE-subclass, whereas f is a reduction function for Σ with respect to $T_0^{\mathcal{L}}$, considering that

(i) $f(\ulcorner\Sigma\urcorner)$ is contained in the UE-class, and

(ii) for every $\phi \in \Sigma$, $\phi \in T_0^{\mathcal{L}}$ iff $f(\ulcorner\phi\urcorner) \in \ulcorner T_0^{\mathcal{L}}\urcorner$, given that ϕ and its rewritten version are logically equivalent.

Since Σ is recursive, it is a decidable subclass for $T_0^{\mathcal{L}}$. In general, any noetherian and canonical set of rewrite rules implicitly defines a (not necessarily proper) extension of a decidable class.

Example 2.3.1

$$(\exists x)(y)(p(x) \vee q(y))$$

is not in UE-form, since there is a free occurrence of a variable in the scope of $(\forall y)$ *which is existentially quantified in the sentence. After the application of* R_5*, the new sentence,*

$$(\exists x)(p(x) \vee (y)q(y))$$

is in UE-form, and, as such, can be supplied to the decision procedure for the UE-subclass. □

The final enlarged subclass includes all the formulae in which

(a) every predicate symbol is monadic (i.e. has arity 1),

(b) every atom has at most one bound variable, and

(c) each atom either does not contain existentially quantified variables, or has a single existentially quantified variable but no other bound variable.

Since the integration strategy is based on properties of decidable classes, rather than on specific procedures or rewriters, it is highly modular: component decision procedures may be replaced and rule bases may be either expanded or contracted, while the integration mechanism remains unchanged. The same strategy can be applied to theories other than the predicate calculus, provided that suitable rules are added to the system[17]. Also, as $T_0^{\mathcal{L}}$ is undecidable, the rule base can be continuously augmented, in order to further enlarge the decidable domain.

Example 2.3.2

i. The conjecture

$$(\exists x)(y)((q(y) \wedge p(x,y)) \vee (q(y) \wedge \neg p(x,y)))$$

after the successive application of $R_9, R_3, R_{13}, R_3, R_{13}, R_3, R_7$ *and* R_4*, is reduced to*

[17] See [Armando & Giunchiglia 93], p. 492 and 500.

$$(\exists x)\overbrace{(y)(q(y) \vee q(y))}^{\phi_1}$$

$$\wedge$$

$$(\exists x)\underbrace{((y)(q(y) \vee \neg p(x,y)) \wedge ((y)(p(x,y) \vee q(y)) \wedge (y)(p(x,y) \vee \neg p(x,y))))}_{\phi_2}$$

where ϕ_1 and ϕ_2 are both normalised w.r.t. $(\exists x)$. As no other rule from the current rule base is applicable to it, the original conjecture cannot be reduced to the UE-class. However, the introduction of a new rule,

$$(\phi \wedge \psi) \vee (\phi \wedge \neg\psi) \quad \Rightarrow \quad \phi$$

allows the generation of $(\exists x)(y)q(y)$, which is in UE-form.

ii. Formulae containing equality, as for instance $(\exists x)(y)(p(x) \wedge (q(x,y) \wedge y = a))$, can also benefit from the introduction of new rules. The application of

$$\phi[v] \wedge v = u \quad \Rightarrow \quad \phi[^u/_v] \wedge v = u$$

to the above conjecture reduces it to

$$(\exists x)(p(x) \wedge (q(x,a) \wedge (y)(y = a)))$$

which is in UE-form. □

A substantial expansion of the rule base, however, would require the introduction of stronger control structures, since noetherianity may be lost in the process. Even when termination is ensured, additional control may be relevant to avoid exhaustive rewriting and improve efficiency.

2.4 Combining Decidable Subclasses

Nelson and Oppen devised a general mechanism for the expansion of subclasses that are decidable for a combination of theories. It consists of

(i) the transformation of a conjecture of the combined language into a

Boolean combination of formulae of the original languages,

(ii) the application of the original decision procedures to the new formulae, and

(iii) the use of the *equality propagation* for the transferral of relevant information about these formulae among the original decision procedures[18].

Their study is concerned with theories that are decidable for the class of quantifier-free formulae. While rewrite-based procedures for subclass extension operate with a single target decidable subclass, their approach reduces a conjecture into a collection of subformulae that belong to disjoint classes.

2.4.1 Combination of Theories

If $\mathcal{L}_1$ and $\mathcal{L}_2$ are languages that do not share non-logical symbols, the process of combining two theories, T_1 and T_2, respectively formulated in $\mathcal{L}_1$ and $\mathcal{L}_2$, preserves the decidability of the quantifier-free class of formulae.

Definition 2.4.1 (Combination of Theories)

Let $T_1, \ldots, T_n$ be theories respectively formulated in the languages $\mathcal{L}_1, \ldots, \mathcal{L}_n$.

i. $\mathcal{L}_1, \ldots, \mathcal{L}_n$ *are* mutually disjoint *iff* $Sym^*_{\mathcal{L}_i} \cap Sym^*_{\mathcal{L}_j} = \emptyset$, *for* $i \neq j$.

ii. $T_1, \ldots, T_n$ *are* mutually disjoint *iff* $\mathcal{L}_1, \ldots, \mathcal{L}_n$ *are mutually disjoint.*

iii. If $T_1, \ldots, T_n$ *are mutually disjoint, such that* Δ_i *is an axiom set for* T_i, *the* combination *of* $T_1, \ldots, T_n$ *is a theory* T_C *in* $\mathcal{L}_C$ *that admits* Δ_C *as axiom set, where*

$$Sym^*_{\mathcal{L}_C} = \bigcup_{i=1}^{n} Sym^*_{\mathcal{L}_i} \quad \textit{and} \quad \Delta_C = \bigcup_{i=1}^{n} \Delta_i.$$

A proof for the fact that the class of quantifier-free formulae of $\mathcal{L}_C$ is decidable for T_C iff the class of quantifier-free formulae of $\mathcal{L}_i$ is decidable for T_i, $1 \leq i \leq n$, can be based on the exhibition of a decision procedure for satisfiability (in T_C) w.r.t. the quantifier-free formulae of $\mathcal{L}_C$ (considering that a formula is *satisfiable in a theory* T iff it is satisfiable in at least one of the models of T). The first stage of the procedure, which puts expressions into *separate normal form*, consists of the following steps.

[18] See [Nelson & Oppen 79].

i. Given a quantifier-free formula ϕ in $\mathcal{L}_C$, it is first replaced by an equivalent formula ϕ' in disjunctive normal form

$$\phi' \stackrel{s}{=} \phi'_1 \vee \cdots \vee \phi'_m,$$

where

$$\phi'_i \stackrel{s}{=} \gamma_{i,1} \wedge \cdots \wedge \gamma_{i,k_i}, \quad 1 \leq i \leq m$$

ii. Each *non-homogeneous* literal $\gamma_{i,j}$ (i.e. a literal in which non-logical symbols of more than one of the component languages $\mathcal{L}_i$ occur) is split into a conjunction of homogeneous literals of the original languages.

(a) If a predicate symbol $p^{\mathcal{L}_a}$ (i.e. a symbol p that belongs to $\mathcal{L}_a$) occurs in the literal, the rewrite rules

$$\begin{array}{rcl} p^{\mathcal{L}_a}[f^{\mathcal{L}_b}(\mathbf{t})/v] & \Rightarrow & p^{\mathcal{L}_a}[v] \wedge v = f^{\mathcal{L}_b}(\mathbf{t}) \\ \neg p^{\mathcal{L}_a}[f^{\mathcal{L}_b}(\mathbf{t})/v] & \Rightarrow & \neg p^{\mathcal{L}_a}[v] \wedge v = f^{\mathcal{L}_b}(\mathbf{t}) \end{array}$$

where $\mathbf{t}$ is a list of terms, $a \neq b$ and v does not occur in ϕ (or in any of its rewritten versions prior to this step), are applied to $\gamma_{i,j}$ to remove every symbol $f^{\mathcal{L}_b}$ from the scope of $p^{\mathcal{L}_a}$. Since all the rules $\epsilon_1 \Rightarrow \epsilon_2$ employed at this and next stage correspond to *reverse implication rules*, i.e. $\models \epsilon_2 \supset \epsilon_1$, they cannot be applied to negative occurrences of subexpressions.

(b) If $\gamma_{i,j}$ is either an equation, or the negation of an equation, and a function symbol $f^{\mathcal{L}_a}$ dominates one of the sides of the equality, rules similar to those described above are applied to $\gamma_{i,j}$ to move every function symbol $g^{\mathcal{L}_b}$, $a \neq b$, to another atom

$$\begin{array}{rcl} f^{\mathcal{L}_a}(\mathbf{t}_1) \triangleleft g^{\mathcal{L}_b}(\mathbf{t}_2) & \Rightarrow & f^{\mathcal{L}_a}(\mathbf{t}_1) \triangleleft v \wedge v = g^{\mathcal{L}_b}(\mathbf{t}_2) \\ f^{\mathcal{L}_a}[g^{\mathcal{L}_b}(\mathbf{t})/v] \triangleleft u & \Rightarrow & f^{\mathcal{L}_a}[v] \triangleleft u \wedge v = g^{\mathcal{L}_b}(\mathbf{t}) \\ u \triangleleft f^{\mathcal{L}_a}[g^{\mathcal{L}_b}(\mathbf{t})/v] & \Rightarrow & u \triangleleft f^{\mathcal{L}_a}[v] \wedge v = g^{\mathcal{L}_b}(\mathbf{t}) \end{array}$$

where $\triangleleft$ stands for either $=$ or $\neq$ and $\boldsymbol{t}, \boldsymbol{t}_1, \boldsymbol{t}_2$ denote finite sequences of terms.

iii. Once each literal in ϕ'_i (including the equalities newly introduced by the application of rewrite rules) is transformed into expressions of the original languages, literals of the same language are attracted, and ϕ'_i is replaced by an expression ψ_i of the form

$$\psi_{i,1}^{\mathcal{L}_1} \wedge \cdots \wedge \psi_{i,n}^{\mathcal{L}_n},$$

where each $\psi_{i,j}^{\mathcal{L}_j}$ is a conjunction of literals of $Fml_{\mathcal{L}_j}$. The formula $\psi \stackrel{s}{=} \psi_1 \vee \cdots \vee \psi_m$ is the *separate normal form* of ϕ.

The following lemma provides the justification for the shift of attention from the original conjecture to its separate normal form.

Lemma 2.4.1 *Let $\mathcal{L}_C$ be a combination of disjoint languages $\mathcal{L}_1, \ldots, \mathcal{L}_n$ and let Δ_i be an axiom set for a theory T_i in $\mathcal{L}_i, 1 \leq i \leq n$. Let T_C be the combination of $T_1, \ldots, T_n$ in $\mathcal{L}_C$.*

i. T_C is a conservative extension of $T_i, 1 \leq i \leq n$.

ii. Given a quantifier-free formula ϕ in $\mathcal{L}_C$, if ψ represents its separate normal form, then ϕ and ψ are equi-satisfiable, i.e. ϕ is (un)satisfiable iff ψ is (un)satisfiable.

PROOF.

i. If $\phi \in T_i$, then $\phi \in T_C$ (since every axiom of T_i is also an axiom of T_C) and $\phi \in Fml_{\mathcal{L}_i}$ (since $T_i \in Fml_{\mathcal{L}_i}$), hence $\phi \in T_C \cap Fml_{\mathcal{L}_i}$ $(*)$.

If $\phi \in T_C \cap Fml_{\mathcal{L}_i}$, then $\Delta_C \models \phi$. Assume that $\phi \notin T_i$, i.e. $\Delta_i \not\models \phi$. Then there is a model $\mathfrak{A}^{\mathcal{L}_i}$ for T_i such that $\mathfrak{A}^{\mathcal{L}_i} \not\models \phi$. If Δ_C is consistent, there is an expansion $\mathfrak{A}^{\mathcal{L}_C}$ of $\mathfrak{A}^{\mathcal{L}_i}$ to $\mathcal{L}_C$ such that $\mathfrak{A}^{\mathcal{L}_C}$ is a model of Δ_C, given that none of the formulae of $\Delta_C - \Delta_i$ contains any non-logical symbol of $\mathcal{L}_i$. Assuming that it is possible to build a model for $\Delta_C - \Delta_i$ with the same universe as $\mathfrak{A}^{\mathcal{L}_i}$, then $\mathfrak{A}^{\mathcal{L}_C}$ would be a model of Δ_C but not of ϕ, thus contradicting the hypothesis that $\Delta_C \models \phi$. Therefore if $\phi \in T_C \cap Fml_{\mathcal{L}_i}$, then $\phi \in T_i$ $(**)$.

From $(*)$ and $(**)$, $T_C \cap Fml_{\mathcal{L}_i} = T_i$, and therefore T_C is a conservative extension of T_i, for all i, $1 \leq i \leq n$.

ii. (Informal) If ϕ is already in disjunctive normal form, the rewrite rules required to put it into a separate normal form formula are derived from logical equivalences of the form

$$\begin{aligned}
p^{\mathcal{L}_a}[f^{\mathcal{L}_b}(\mathbf{t})/v] &\equiv (\exists v)(p^{\mathcal{L}_a}[v] \wedge v = f^{\mathcal{L}_b}(\mathbf{t})) \\
\neg p^{\mathcal{L}_a}[f^{\mathcal{L}_b}(\mathbf{t})/v] &\equiv (\exists v)(\neg p^{\mathcal{L}_a}[v] \wedge v = f^{\mathcal{L}_b}(\mathbf{t})) \\
f^{\mathcal{L}_a}(\mathbf{t}_1) \lhd g^{\mathcal{L}_b}(\mathbf{t}_2) &\equiv (\exists v)(f^{\mathcal{L}_a}(\mathbf{t}_1) \lhd v \wedge v = g^{\mathcal{L}_b}(\mathbf{t}_2)) \\
f^{\mathcal{L}_a}[g^{\mathcal{L}_b}(\mathbf{t})/v] \lhd u &\equiv (\exists v)(f^{\mathcal{L}_a}[v] \lhd u \wedge v = g^{\mathcal{L}_b}(\mathbf{t})) \\
u \lhd f^{\mathcal{L}_a}[g^{\mathcal{L}_b}(\mathbf{t})/v] &\equiv (\exists v)(u \lhd f^{\mathcal{L}_a}[v] \wedge v = g^{\mathcal{L}_b}(\mathbf{t}))
\end{aligned}$$

where $a \neq b$, $\lhd$ denotes either $=$ or $\neq$, and $\boldsymbol{t}, \boldsymbol{t}_1, \boldsymbol{t}_2$ denote finite sequences of terms. After ϕ is rewritten by the exhaustive application of the equivalences above to (positive) occurrences of literals, existential quantifiers can be collected into a prefix, provided that each of the newly introduced variables does not occur in ϕ (or in any of its subsequent rewritten versions). The prenex disjunctive normal form of the final rewritten conjecture ϕ' can then be represented as

$$(\exists v_1) \ldots (\exists v_p)\psi,$$

where ψ is a separate normal form for ϕ. Since a formula is satisfiable iff its existential closure is satisfiable, and considering that ϕ and ϕ' are logically equivalent, it follows that ϕ is satisfiable iff ϕ' is satisfiable iff $[[\phi']]$ is satisfiable iff ψ is satisfiable. ∎

The class

$$\{\neg\psi \mid \psi \in Fml_{\mathcal{L}_C} \;\&\; \psi \text{ is in separate normal form}\}$$

is a reduction class for the class of quantifier-free formulae of $\mathcal{L}_C$ w.r.t. T_C. It suffices to consider that, given a quantifier-free formula $\phi \in Fml_{\mathcal{L}_C}$, $\phi \in T_C$ iff $\neg\phi$ is unsatisfiable in T_C iff ψ' is unsatisfiable in T_C iff $\neg\psi' \in T_C$, where ψ' is a separate normal form for

$\neg\phi$. The reduction function for the above class w.r.t. T_C associates each of its formulae ϕ with the separate normal form formula for $\neg\phi$. To determine the validity in T_C of $\neg\psi$, or the unsatisfiability in T_C of ψ,

i. Each expression $\psi_{i,j}^{\mathcal{L}_j}$ must be supplied to the decision procedure for T_j. If at least one of them is unsatisfiable (in T_j), ψ_i is unsatisfiable in T_C (since, if $\psi_{i,j}^{\mathcal{L}_j}$ is unsatisfiable in T_j, then $\neg\psi_{i,j}^{\mathcal{L}_j}$ is valid in T_j, and, by extension, valid in T_C as well, for $T_j \subset T_C$).

ii. As a result, $\bigvee_{k=1}^{n} \neg\psi_{i,k}^{\mathcal{L}_k}$, which is equivalent to $\neg\psi_i$, is also valid in T_C, i.e. ψ_i is unsatisfiable in T_C. And if every disjunct of ψ is unsatisfiable in T_C, the same applies to ϕ.

However, in the event that every conjunct $\psi_{i,j}^{\mathcal{L}_j}$ is satisfiable in T_C, the same conclusion does not necessarily apply to ψ_i, since the satisfiability of all the conjuncts does not amount to the satisfiability of the conjunction. Each decision procedure actually establishes whether $\psi_{i,j}^{\mathcal{L}_j}$ is satisfiable in T_j or not; however, since T_C is a conservative extension of each component theory T_j, $\psi_{i,j}^{\mathcal{L}_j}$ is satisfiable in T_j iff $\psi_{i,j}^{\mathcal{L}_j}$ is satisfiable in T_C. The last stage of the procedure then examines each ψ_i which has not been identified as unsatisfiable in T_C by the original decision procedures.

2.4.2 Equality Propagation

Considering that a formula is satisfiable iff every formula it entails is also satisfiable, a possible indirect method for establishing its satisfiability involves examining *all* of its entailed expressions (excluding those in which the formula itself occurs as subexpression). Let $\mathcal{E}_\psi$ be defined as $\bigcup_{i=1}^{m} \mathcal{E}_{\psi_i}$, where $\mathcal{E}_{\psi_i}$ is made up of expressions of the form

$$v = u \qquad (*)$$

or

$$v_1 = u_1 \vee \cdots \vee v_n = u_n \qquad (**)$$

such that

(a) $(*)$ and $(**)$ (but not the disjuncts of $(**)$) are entailed by ψ_i,

(b) each v, u, v_j, u_j is a variable that occurs in ψ_i, and

(c) $\psi \stackrel{s}{=} \psi_1 \vee \cdots \vee \psi_m$.

For combinations of disjoint theories, if ψ is in *separate normal form*, only the entailed formulae of $\mathcal{E}_\psi$ have to be taken into account. It is possible to prove that ψ_i is unsatisfiable iff the conjunction of at least one of the (satisfiable) conjuncts $\psi_{i,j}^{\mathcal{L}_j}$ with all the formulae in $\mathcal{E}_{\psi_i}$ is unsatisfiable. An iterating procedure can then be applied to ψ_i to assess its satisfiability[19].

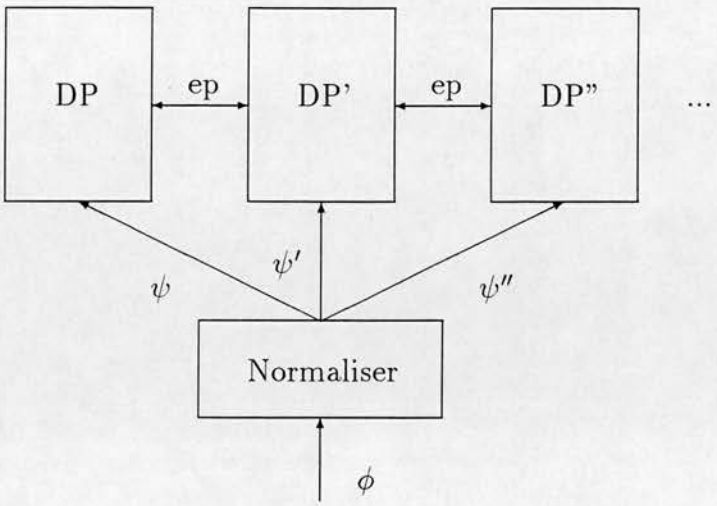
i. If δ is one of the equalities entailed by the conjunct $\psi_{i,j}^{\mathcal{L}_j}$, then δ is conjoined to each of the other conjuncts $\psi_{i,k}^{\mathcal{L}_k}$, and the decision procedure for each original theory is applied to the new conjunctions. If the expanded formula is recognised as unsatisfiable in T_C, it follows that ψ_i is unsatisfiable in T_C as well.

ii. If no atomic entailed equality is left, and no unsatisfiable expression has been detected, entailed disjunctions of equalities have to be considered. Then ψ_i is unsatisfiable iff there exists a formula $\delta^* = \delta_1^* \vee \cdots \vee \delta_n^*$ of the form $(**)$ above such that $\psi \wedge \delta^*$ is unsatisfiable, i.e. ψ is unsatisfiable iff for all $j, 1 \leq j \leq n$, $\psi \wedge \delta_j^*$ is unsatisfiable. The procedure described above has then to be applied to this extended expression.

iii. If unsatisfiability is not detected, and no other expression in $\mathcal{E}_{\psi_i}$ has been left, ψ_i, as well as ψ, is satisfiable.

As $\mathcal{E}_{\psi_i}$ is finite, the procedure always terminates, and is therefore a decision procedure for T_C-satisfiability w.r.t. the class of quantifier-free formulae of $\mathcal{L}_C$. Figure 2.3 exhibits the structure of the mechanism, which contains both cooperation of decision procedures and interfaced connections.

[19] A formula ϕ entails δ iff $\models \phi \supset \delta$; ϕ entails δ in a theory T iff $T \models \phi \supset \delta$. A proof for the statement that a formula is satisfiable iff every formula it entails is also satisfiable can be found in appendix E. Equality propagation is discussed in [Nelson & Oppen 79], p. 253-5.



DP	decision procedure
ep	equality propagation
ϕ	input formula
$\psi, \psi', \ldots$	separate normal form formulae

Figure 2.3: Cooperation of Decision Procedures

2.4.3 Limitations

The mechanism of combination of decidable subclasses has been added to the *Stanford Pascal Verifier*, an interactive system for reasoning about Pascal programs. Although it provides a straightforward procedure for satisfiability, the main limitation is its inability to deal with most of the extensions of combined theories which are commonly used in the representation of data structures. For instance, let the combined theory of *arrays and natural numbers* be defined by the union of an axiom set for Presburger arithmetic[20],

[20] The *Stanford Pascal Verifier* is also described in [Nelson & Oppen 79]. According to Boyer and Moore,

> "Decidable theories are inadequate for the specification of most programs. The situation is improved somewhat by the work of Oppen and Nelson (1979) which shows how one can construct a system of co-operating decision procedures for disjoint theories. But in our experience most theories of use to program verification are not disjoint. For example, the function `LEN` connects the theory of lists to that of the naturals and `DELTA1` connects character strings to naturals." ([Boyer & Moore 88], p. 122)

The set of axioms for Presburger arithmetic is distinct from that presented in appendix B.2, since it contains an additional predicate symbol, *nat*.

$$
\begin{array}{ll}
P_1 & nat(0) \\
P_2 & nat(x) \supset (s(x) \neq 0) \\
P_3 & nat(x) \supset (nat(y) \supset (s(x) = s(y) \supset x = y)) \\
P_4 & nat(y) \supset (0 + y = y) \\
P_5 & nat(x) \supset (nat(y) \supset (s(x) + y = s(x + y))) \\
P_6 & \phi[0] \wedge (nat(y) \supset (\phi[y] \supset \phi[s(y)])) \\
 & \quad \supset \\
 & nat(x) \supset \phi[x],
\end{array}
$$

and an axiom set for TA, the theory of arrays[21],

$$
\begin{array}{ll}
A_1 & array([\ ]) \\
A_2 & atom(u) \supset (array(v) \supset array(u \circ v)) \\
A_3 & atom(u) \supset (array(v) \supset u \circ v \neq [\ ]) \\
A_4 & atom(u) \wedge atom(t) \wedge array(v) \wedge array(w) \\
 & \quad \supset \\
 & (u \circ v = t \circ w) \supset (u = t \wedge v = w) \\
A_5 & atom(u) \supset u \circ [\ ] = u \\
A_6 & \phi[[\ ]] \wedge (atom(u) \supset (array(v) \supset (\phi[v] \supset \phi[u \circ v]))) \\
 & \quad \supset \\
 & array(u) \supset \phi[u], \\
 & (\text{if } u \text{ does not occur free in } \phi[v]),
\end{array}
$$

Since each component theory is decidable in the quantifier-free subclass of their respective languages, which are disjoint, the combined theory is decidable in the class of quantifier-free formulae of the combined language. Nelson & Oppen's mechanism then supplies a decision procedure for it. However, instead of their simple combination, the theory of *arrays of natural numbers*, ANN, is more frequently found in the representation of data structures. ANN is a non-conservative extension of the simple combination that includes the axiom

$$N_1 \quad nat(x) \equiv atom(x)$$

which restricts the elements of arrays to natural numbers. For this theory, the results derived by Nelson and Oppen cannot be applied.

Besides such extensions, new functions and predicates can be also added to those originally available in the combined theory[22]. For instance, the functions min and max, which respectively identify the minimum and maximum elements of an array of

[21] See [Manna & Waldinger 85].

[22] See [Boyer & Moore 88], p. 122.

natural numbers, can be introduced in ANN by means of new axioms, provided that an order relation over natural numbers has been previously added to the theory.

$$
\begin{array}{ll}
M_1 & nat(x) \supset min(x \circ [\ ]) = x \\
M_2 & nat(x) \supset (array(u) \supset (x < min(u) \supset min(x \circ u) = x)) \\
M_3 & nat(x) \supset (array(u) \supset (min(u) \leq x \supset min(x \circ u) = min(u))) \\
M_4 & nat(x) \supset max(x \circ [\ ]) = x \\
M_5 & nat(x) \supset (array(u) \supset (x > max(u) \supset max(x \circ u) = x)) \\
M_6 & nat(x) \supset (array(u) \supset (max(u) \geq x \supset max(x \circ u) = max(u)))
\end{array}
$$

For this extended theory, then, other mechanisms for decidable subclass extension have to be sought.

2.5 Introducing Additional Hypotheses

To address the problem of proving certain verification conditions for JOVIAL programs, Shostak devised a mechanism that explores the decidability of quantifier-free subclasses for certain theories by means of the introduction of additional hypotheses[23]. Let T be a theory in $\mathcal{L}$ which is decidable in the class of quantifier-free formulae, and $\mathcal{L}'$ be an expansion of $\mathcal{L}$. If T' is an extension of T in $\mathcal{L}'$ for which T represents an axiom set, then, according to theorem 2.2.1, the class of quantifier-free formulae of $\mathcal{L}'$ is decidable for T' in $Fml_{\mathcal{L}'}$. Finally, if T'' is a non-conservative extension of T' in $\mathcal{L}'$, whenever a quantifier-free formula ψ in $\mathcal{L}'$ is valid in T', it is also a theorem in T''. However, if $\psi \notin T'$, let $\Phi = \{\phi_1, \ldots, \phi_n, \ldots\}$ is a subset of quantifier-free T''-theorems, some of which involve undefined symbols of T'. Then, according to the *additional hypothesis introduction* lemma (2.2.1), whenever

$$T' \models \phi_{i_1} \supset \cdots \supset \phi_{i_m} \supset \psi$$

where $\phi_{i_j} \in \Phi$, it follows that $\psi \in T''$.

Example 2.5.1 *Since PrA is decidable, the same applies to its subclass of quantifier-free formulae. If* max *is an undefined function symbol in an extension PrA′ of this theory, and PrA″ is an extension of PrA′ where* max *is defined as the standard binary maximum function over natural numbers, in order to prove that*

[23] See [Shostak 79].

$$x = y + 2 \supset \max(x, y) = x \quad (*)$$

is a theorem in PrA'', new hypotheses, such as

$$\{x \geq y \supset \max(x, y) = x, \;\; y \geq x \supset \max(x, y) = y\}$$

should be added as antecedents to the formula above, until it could be recognised as a PrA'-theorem. Thereafter, since

$$PrA'' \;\; \models \;\; x \geq y \supset \max(x, y) = x \;\wedge\; y \geq x \supset \max(x, y) = y$$

it follows that $(*)$ *is also a theorem of PrA''.* □

The hypothesis introduction lemma can be also applied in conjunction with the mechanism for the combination of quantifier-free decidable subclasses. In the case of ANN, discussed in section 2.4, given a conjecture ϕ in the combined language of Presburger arithmetic and the theory of arrays, if

$$N_1[^t/_x] \supset \phi$$

where $N_1[^t/_x]$ is an instance of the axiom $nat(x) \equiv atom(x)$, can be shown to be valid in the simple combination of the original theories, then ϕ is a theorem of ANN. The same mechanism can be used for the new function and predicate symbols of ANN^* as well. A formula ϕ' in $\mathcal{L}_{ANN^*}$ can be expanded to include additional hypotheses $\delta_1, \ldots, \delta_n$ about the new symbols, and

$$\text{if} \quad ANN \models \delta_1 \supset \cdots \supset \delta_n \supset \phi \quad \text{then} \quad ANN^* \models \phi$$

for $\delta_1, \ldots, \delta_n$ are theorems of ANN^*. As a result, if

$$N_1[^t/_x] \supset \delta_1 \supset \cdots \supset \delta_n \supset \phi$$

is valid in the simple combination of PrA and TA, then $ANN^* \models \phi$.

The lack of criteria for the selection of additional hypotheses is one of the deficiencies of the approach. For instance, if PrA^* is the extension of PrA to $\mathcal{L}_{PA} = \{0, 1, s, +, \times\}$ in which $\times$ is an undefined function symbol, given the conjecture

$$x_4 = x_5 \ \wedge \ x_2 \times s(x_1) + x_4 = x_5 + s(x_1) \times x_3 \ \supset \ x_2 = x_3 \quad (\dagger)$$

in order to prove that it is a theorem of Peano arithmetic (PA), additional hypotheses that involve multiplication are required[24]. Finite sequences of such theorems should be tested, up to a pre-established maximum number of tests, until a suitable list, e.g.

$$\{\ s(x_1) \times x_2 = s(x_1) \times x_3 \supset x_2 = x_3, \ \ s(x_1) \times x_2 = x_2 \times s(x_1)\ \}$$

is eventually found. Given a set Φ of such theorems, there is a search problem that involves not only testing finite combinations of theorems, but also identifying the adequate instances, since the mechanism applies only to quantifier-free formulae. In the example above, the first additional hypothesis is the only instance of the arithmetical theorem

$$(x)(y)(z)(s(x) \times y = s(x) \times z \supset y = z))$$

that is suitable for conjecture ($\dagger$). Without control, the introduction of additional hypotheses may be impractical from the viewpoint of effective mechanical theorem proving.

2.6 Extending Combined Decidable Subclasses

Tatzelwurm is a first-order theorem prover based on the analytic tableau method, which consists of the application of the rules of table 2.2 for the generation of semantic trees. When a formula is unsatisfiable, its tree is finite, and each branch of the tree is *closed*, i.e. it ends in a node that contains a complementary pair of literals. In rules of type α, the premise is unsatisfiable if either of the derived formulae, α_1 and α_2, is unsatisfiable, hence both derived formulae are kept in the same branch. For rules of type β, the premise is unsatisfiable if both derived formulae are unsatisfiable, and so

[24] Peano arithmetic is described in appendix B.

α	α_1	α_2
$\neg\neg\phi$	ϕ	–
$\phi \wedge \psi$	ϕ	ψ
$\neg(\phi \vee \psi)$	$\neg\phi$	$\neg\psi$
$\neg(\phi \supset \psi)$	ϕ	$\neg\psi$
β	β_1	β_2
$\phi \vee \psi$	ϕ	ψ
$\neg(\phi \wedge \psi)$	$\neg\phi$	$\neg\psi$
$\phi \supset \psi$	$\neg\phi$	ψ
γ	γ_1	γ_2
$(\forall v)\phi[v]$	$\phi[^a/_v]$	$(\forall v)\phi[v]$
$\neg(\exists v)\phi[v]$	$\neg\phi[^a/_v]$	$\neg(\exists v)\phi[v]$
δ	δ_1	δ_2
$(\exists v)\phi[v]$	$\phi[^a/_v]$	–
$\neg(\forall v)\phi[v]$	$\neg\phi[^a/_v]$	–

Table 2.2: Rules for the Analytic Tableau

an additional derived branch is created. For any first-order language without equality, the analytic tableau method provides a semi-decision procedure for the set of logically valid formulae[25].

Example 2.6.1 *A refutation for the formula* $(x)p(x,z) \wedge (\exists y)\neg p(y,z)$ *can be obtained by means of the analytic tableau method as follows.*

$$\frac{(x)p(x,z) \wedge (\exists y)\neg p(y,z)}{\downarrow \alpha}$$
$$\frac{(x)p(x,z),\ (\exists y)\neg p(y,z)}{\downarrow \gamma}$$
$$p(a,z),\ (x)p(x,z),\ \underline{(\exists y)\neg p(y,z)}$$
$$\downarrow \delta$$
$$p(a,z),\ (x)p(x,z),\ \neg p(a,z)$$

The above tree has a single branch with four nodes. The fourth node contains a complementary pair of literals, therefore the branch is closed (i.e. there is no structure for $\mathcal{L} = \{a, p\}$ *in which the formulae of the fourth node could be jointly satisfied). Given the nature of the rules employed in the derivation of a tableau, the root formula is unsatisfiable as well.* □

[25] *Tatzelwurm* is described in [Käufl 89]. Concerning the analytic (or semantic) tableau method, a more detailed description can be found, for instance, in [Ben-Ari 93], p. 33-44, 106-117.

For a non-conservative extension T of $T_0^{\mathcal{L}}$, formulated in the same language $\mathcal{L}$, the set of T-unsatisfiable formulae cannot be entirely identified by the tableau, since $T_0^{\mathcal{L}} \subset T$, and the mechanism loses completeness. However, when T is obtained from $T_0^{\mathcal{L}}$ by the introduction of axioms for equality, as well as the axiom sets for disjoint theories $T_1, \ldots, T_n$, respectively formulated in the disjoint languages (with equality) $\mathcal{L}_1, \ldots, \mathcal{L}_n$, the inclusion of new rules and special procedures in the tableau method enlarges the class of T-unsatisfiable formulae it can identify. Moreover, if each T_i is decidable for the class of quantifier-free formulae of $\mathcal{L}_i$, and if the union of these languages coincides with $\mathcal{L}$, the class of quantifier-free formulae of $\mathcal{L}$ is decidable for T, as described in section 2.4, and their decision procedures are also integrated into the extended tableau.

In particular, the *theory of pure equality*, formulated in any language $\mathcal{L}^f$ that contains only function symbols, can be included in virtually any combined theory. Since it has no axioms, it is identical to $T_0^{\mathcal{L}^f}$ and every non-logical symbol of $\mathcal{L}^f$ is undefined in the theory. The quantifier-free class of formulae of $\mathcal{L}^f$ is decidable for $T_0^{\mathcal{L}^f}$, and a decision procedure for this class can be built based on the congruence closure of sets of equations[26]. This procedure allows a more efficient handling of equality inside the combined theory: alternative solutions, based for instance on the introduction of the rule

$$\frac{\phi[\![s]\!], s = t, \Sigma'}{\phi[\![s]\!], \phi[\![^t/_s]\!], s = t, \Sigma'}$$

which causes an explosive increase in the number of derived nodes, can be therefore avoided, without any loss of deductive completeness. When the theory of pure equality is a component of T, the extended tableau method includes

(a) rules for generating literals in *separate normal form*,

(b) decision procedures for the component theories $T_1, \ldots, T_n$ (w.r.t. quantifier-free subclasses), and

(c) an equality propagation procedure, responsible for generating relevant entailed equalities from satisfiable sets of homogeneous literals.

The generation of expressions in separate normal form is accomplished by two new rules, η and ζ, listed in table 2.3. Sets of homogeneous literals are then supplied to

[26] See [Nelson & Oppen 80].

η	η_1	η_2
$P(t_1, \ldots, t_n)$	$\left[\left[\bigwedge_{i=1}^{n} v_i = t_i \wedge P(v_1, \ldots, v_n) \right]\right]$	$P(t_1, \ldots, t_n)$
ζ	ζ_1	ζ_2
$\neg P(t_1, \ldots, t_n)$	$\neg \left[\bigwedge_{i=1}^{n} v_i = t_i \supset P(v_1, \ldots, v_n) \right]$	$P(t_1, \ldots, t_n)$

Conditions
$v_1, \ldots, v_n$ do not occur in $P(t_1, \ldots, t_n)$
$P(t_1, \ldots, t_n)$ is non-homogeneous
$[\phi]$ universal closure of ϕ
$[[\phi]]$ existential closure of ϕ

Table 2.3: New Rules for the Analytic Tableau

the decision procedure for each theory T_i with respect to the quantifier-free class of formulae. When a set of literals is satisfiable in T_i, entailed equalities are generated and added to the new node of the branch. The definition of a closed branch has to be expanded to conform to the new method.

Definition 2.6.1 (Closed branch in the extended tableau)

A branch of a tableau is closed *iff it has a node* Σ *that contains*

i. a pair of complementary formulae, or

ii. a formula of the form $t \neq t$, *or*

iii. two formulae, ϕ *and* $\psi[\![t_1, \ldots, t_n]\!]$, *such that* ϕ *and* $\sigma\psi$ *are complementary, where* σ *is the replacement* $\{^{s_1}/_{t_1}, \ldots, ^{s_n}/_{t_n}\}$, *such that* $\Delta \models (s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$ *and* $\Delta \subseteq \Sigma$ *is a set of equations, or*

iv. a subset of formulae Γ *which belongs to a decidable subclass and is unsatisfiable.*

The application of the extended tableau method is informally described in table 2.4. It seems that the extended tableau represents a semi-decision procedure for T w.r.t. the fully quantified underlying language. The integration of tableau rules with decision

procedures ends up extending the boundaries of the decidable subclasses as well. For instance, due to the application of γ and δ tableau rules, it is possible to derive a quantifier-free formula ϕ respectively from $[\phi]$ and $[[\phi]]$. If ϕ is unsatisfiable in T, the same then applies to its universal and existential closures, which can then be added to an extended decidable class.

The extension of the quantifier-free subclass of $\mathcal{L}$ is obtained through *weak quantifier elimination*. Given a sentence τ of $\mathcal{L}$, if $\neg\tau$ is the root note of a tableau entirely generated by the application of logical rules (α, β, γ and/or δ), whose terminal nodes, $\Gamma_1, \ldots, \Gamma_n$, where $\Gamma_i = \{\gamma_{i,1}, \ldots, \gamma_{i,m_i}\}$, are all made up of quantifier-free literals, let ψ be the quantifier-free formula defined as

$$\neg\psi \stackrel{s}{=} ((\gamma_{1,1} \wedge \cdots \wedge \gamma_{1,m_1}) \vee \cdots \vee (\gamma_{n,1} \wedge \cdots \wedge \gamma_{n,m_n}))$$

Then $T \models \psi$ implies $T \models \tau$, which represents a weak transformation, as described in section 2.2.3. The extended decidable class includes any quantified formula τ whose negation can be transformed into a quantifier-free formula that is identified as unsatisfiable by the decision procedure for the combined theory[27].

Since the analytic tableau does not represent a decision procedure, there is no guarantee that every quantified formula reducible to a quantifier-free formula can be effectively recognised as such. Effectiveness however is achieved once limits are imposed on the size of the tableau. Formulae that cannot be proved unsatisfiable by *a tableau of maximum size m*, but have been identified as such by the extended tableau, under the same size constraints, are elements of the extended decidable class. The extended class can be defined as

$$\Sigma = \{\phi \in Fml_{\mathcal{L}} \mid \neg\phi \xrightarrow{d} \neg\psi \;\&\; \psi \in T \;\&\; \psi \text{ is quantifier-free}\}$$

[27] The identification of the above formula ψ results from the following process. Let $\{\neg\tau\}$ be the root node of a semantic tree, and let $\Gamma_1, \ldots, \Gamma_n$ be all its terminal nodes. Clearly, given the nature of the tableau rules, if all such sets Γ_i are T-unsatisfiable, then $\neg\tau$ is T-unsatisfiable as well. The unsatisfiability of a node $\Gamma_i = \{\gamma_{i,1}, \ldots, \gamma_{i,m_i}\}$ (i.e. the joint unsatisfiability of its elements) amounts to the unsatisfiability of the conjunction $\gamma_{i,1} \wedge \cdots \wedge \gamma_{i,m_i}$. On the other hand, considering that

$$sat_T(\phi_1) \text{ or } sat_T(\phi_2) \quad \text{iff} \quad sat_T(\phi_1 \vee \phi_2)$$

the unsatisfiability of all branches (i.e. their individual unsatisfiability) then amounts to the unsatisfiability of $((\gamma_{1,1} \wedge \cdots \wedge \gamma_{1,m_1}) \vee \cdots \vee (\gamma_{n,1} \wedge \cdots \wedge \gamma_{n,m_n}))$.

Let $\Gamma_0 = \{\neg\phi\}$ be the root node of a semantic tableau.

i. If each branch of the tableau is closed (according to definition 2.6.1), then $T \models \phi$

ii. If the terminal node Γ_n of each open branch of the tableau has only T-satisfiable homogeneous literals, then $T \not\models \phi$

iii. Otherwise let Γ_n be the terminal node of an open branch.

   (a) If Γ_n contains composite formulae, logical rules are applied and new nodes $\Gamma_{n+1}, \ldots, \Gamma_{n+m}$ are generated.

   (b) If Γ_n contains non-homogeneous literals, rules η and ζ are applied to them, and the transformed expressions are included in a new node Γ_{n+1}

   (c) If Γ_n has a subset of homogeneous literals which fall into the domain of a decidable class, the equality propagation procedure generates the relevant entailed equations and add them to the new node Γ_{n+1}

Table 2.4: Extended Analytic Tableau Procedure

where $\phi_1 \xrightarrow{d} \phi_2$ indicates that a formula equivalent to ϕ_2 can be associated with a tableau of maximum depth d generated from ϕ_1. When a maximum depth is given, there are only finite many cases to be examined for each formula of $\mathcal{L}$. Then there is an effective procedure to determine whether a formula belongs to Σ or not. The corresponding weak reduction function associates ϕ with ψ: for all $\phi \in \Sigma$, if its reduced form ψ is a T-theorem, then $\phi \in T$.

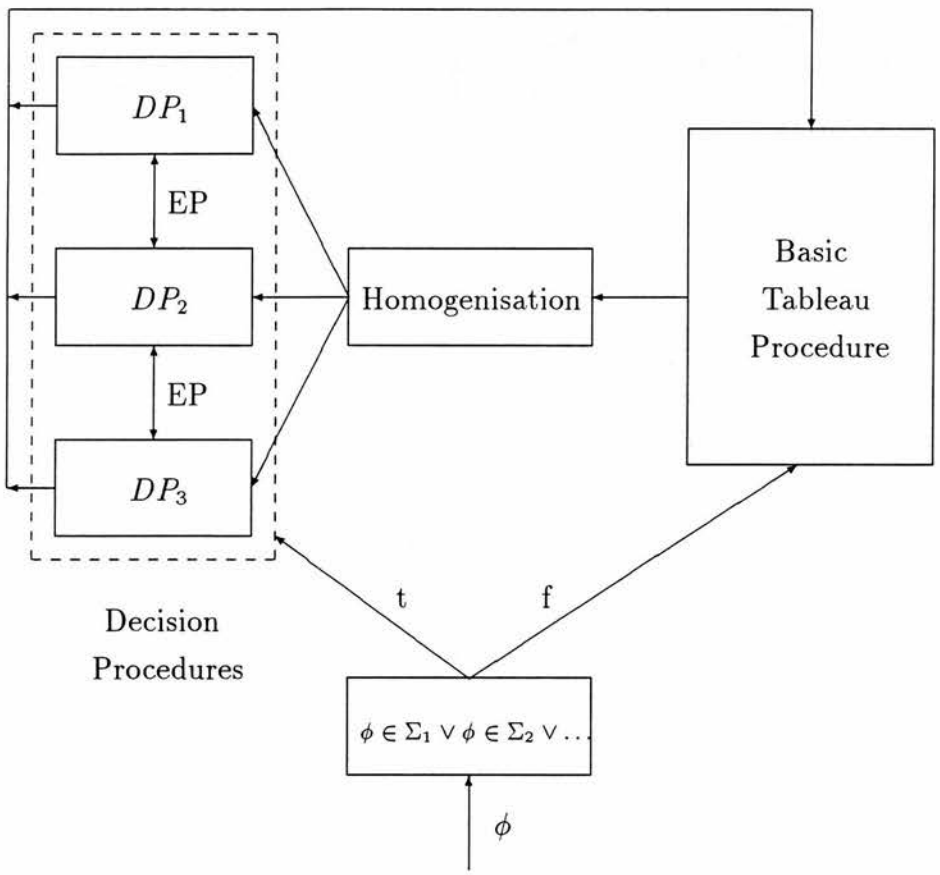
2.7 Propositionally Valid Formulae

Decision procedures are also used in the *Prototype Verification System* (*PVS*), which combines a language for writing specifications and an interactive proof checker for verifying their correctness. The proof checker has access to decision procedures for ground equalities and linear inequalities, which are called to simplify IF-expressions, datatype expressions, function definitions and conditions of conditional rewrite rules[28].

The mechanism for the integration of decision procedures in *PVS* enlarges the class of propositionally valid formulae[29]. Other decidable subclasses for a theory T in $\mathcal{L}$ can be

[28] See [Owre *et al* 92].

[29] As discussed in appendix C.3, this class is decidable for any theory in the corresponding language $\mathcal{L}$, and has type I.



Σ_1	quantifier-free decidable subclass
DP_i	decision procedure for T w.r.t. Σ_i
EP	equality propagation

Figure 2.4: Extended Semantic Tableau Procedure

explored to extend the propositionally valid set. The extension procedure consists of the replacement of subformulae of a conjecture with propositional constants, $\top$ or $\bot$, whenever such subformulae or their negations are identified as T-theorems by a decision procedure for T w.r.t. a particular subclass. The extended class can be characterised as

$$\Sigma = \{\phi \in Fml_{\mathcal{L}} \mid \phi' \text{ is propositionally valid}\}$$

where ϕ' is obtained from ϕ by the replacement of T-valid or T-unsatisfiable subformulae respectively with the propositional constants $\top$ or $\bot$. The process is analogous to subformula rewriting, with the proviso that all the rules have the form $\phi \Rightarrow \top$ or $\phi \Rightarrow \bot$. The procedure that performs this replacement therefore computes the reduction function for Σ w.r.t. T. Also, for all $\phi \in \Sigma$, $\phi \in T$ iff $\phi' \in T$, since

i. If $\phi \in \Sigma$ and $\phi \in T$, then according to the definition of Σ, ϕ' is propositionally valid. According to lemma C.3.1, $\phi' \in T$.

ii. If $\phi \in \Sigma$ and $\phi' \in T$, since ϕ' is obtained from ϕ by the replacement of T-valid/unsatisfiable subformulae of ϕ with $\top/\bot$, and considering that, for any formula ψ of $\mathcal{L}$, if $T \models \psi[\![\psi']\!]$ and $T \models \psi' \equiv \psi''$, then $T \models \psi[\![^{\psi''}/_{\psi'}]\!]$, it follows that $\phi \in T$.

The same approach can be used for *formula simplification*, where subformulae are replaced by propositional constants, as indicated above, but the transformed formula does not have to be propositionally valid: instead of the original conjecture, its simplified version can be thereafter supplied to a theorem prover. The example that follows illustrates the role of the integrated system both in extending the decidable subclass and simplifying expressions.

Example 2.7.1

i. Given the arithmetical formula ϕ,

$$(\exists z)(w)(z^2 + w^3 = 5zw) \supset (x)(\exists y)(x + y = 3x)$$

as the consequent, $(x)(\exists y)(x + y = 3x)$*, belongs to a decidable class for PA, it is possible to effectively establish that it is a PA-theorem; as a result, this formula is equivalent in PA to*

$$(\exists z)(w)(z^2 + w^3 = 5zw) \supset \top$$

whose propositional form, $p \supset \top$*, is valid. Hence,* ϕ *is also a PA-theorem, and belongs to the extension of the class of propositionally valid formulae in PA.*

ii. Let ϕ' *be the formula*

$$(x)(\exists y)(x + y = 3x) \supset (\exists z)(w)(z^2 + w^3 = 5zw)$$

Since its antecedent is a valid formula of a decidable subclass of PA, ϕ' *can be replaced by an equivalent propositional constant,*

$$\top \supset (\exists z)(w)(z^2 + w^3 = 5zw)$$

Although its propositional form is not valid, it can nonetheless be replaced by a propositionally equivalent and syntactically simpler formula,

$$(\exists z)(w)(z^2 + w^3 = 5zw)$$

which may be supplied to a theorem prover. □

The second example illustrates the possible use of decision procedures in conditional rewriting: once an instance $\sigma\psi$ of the condition of a rule

$$\psi \rightarrow \epsilon_1 \Rightarrow \epsilon_2$$

has been proven, $\sigma\epsilon_1 \Rightarrow \sigma\epsilon_2$ can then be applied to the expression under consideration.

2.8 Conclusions

Several mechanisms for decidable subclass extension have been incorporated in theorem provers with the purpose of strengthening the role of decision procedures in the

System	Integration Mode*	Decision Procedures	Extension Validation	Additional Applications for DP
GETFOL	Interfacing	UE-class quantifier-free class propositional calculus	Rewrite rules	—
PVS	Cooperation	ground equalities linear inequalities	Valid subformula replacement	Simplification Conditional rewriting
Tatzelwurm	Interfacing & Cooperation	quantifier-free classes (pure equality & others)	Inference rules (weak quantifier elimination)	Simplification
Stanford Pascal Verifier	Interfacing & Cooperation	quantifier-free classes • $\langle \mathbb{R}, +, \leq \rangle$ • arrays, lists, • pure equality	Separate normal forming Equality propagation	—
JOVIAL program Verifier	Interfacing	quantifier-free classes • *PrA* • extensions of *PrA*	Additional hypothesis introduction	—

* Integration modes are described in chapter 1

Table 2.5: Summary - Hybrid Theorem Provers

mechanical recognition of theorems. The main properties of some of these systems are summarised in table 2.5. Most of them suffer from the lack of suitable control structures for a more efficient decidable subclass extension. Two in particular, one based on the application of rewrite rules, and the other, on the introduction of additional hypotheses, face efficiency problems in the presence of large sets of rules or theorems. Under these circumstances, strategies for the identification of rules or additional hypotheses can have a positive effect on the overall performance of the mechanisms. Some guidelines concerning the selection of hypotheses are discussed in the next chapter.

Chapter 3

Selecting Additional Hypotheses

The procedures for decidable subclass extension examined in the previous chapter rely basically upon either the application of rewrite and inference rules, or the introduction of additional hypotheses. In both cases, there is a potential search problem whenever multiple rules or hypotheses are available. Search control can therefore reduce the computational complexity of the process[1].

Heuristic strategies for the selection of hypotheses are employed in *Nqthm*, the Boyer and Moore theorem prover. Section 3.1 describes the main features of this system. Section 3.2 then presents a mechanism that has been introduced in *Nqthm* for the choice of potentially relevant definitions, axioms and theorems which, after instantiation, are added to the hypothesis set of a conjecture.

3.1 The Boyer and Moore Theorem Prover

Boyer and Moore's *Nqthm* is a heuristic theorem prover that employs strategic information in the derivation of inductive proofs. As it has been used for the verification of program correctness, decision procedures for arithmetic and related theories have been added to the system to improve its efficiency. Such procedures are interfaced to two modules of the prover, the *simplifier* and the *rewriter*.

[1] Search space and control in the context of theorem proving are discussed in appendix D.

3.1.1 Basic System

Nqthm operates in the quantifier-free fragment of a first-order language. Many-sorted axiomatic theories are indirectly represented by means of the *shell principle*, which requires, whenever a new sort has to be implicitly introduced in the underlying language, the inclusion of symbols for (i) a constructor function, (ii) a destructor function, (iii) individual bottom constants, (iv) a well-founded order relation and (v) a sort-recogniser function (or predicate)[2]. Natural numbers are introduced through the shell principle as described in table 3.1. The corresponding axioms are

A_1 $\neg nat(\top) \wedge \neg nat(\bot)$
A_2 $nat(0) \wedge (nat(v) \supset nat(\mathrm{s}(v)))$
A_3 $\mathrm{s}(v) \neq 0$
A_4 $nat(v) \wedge v \neq 0 \supset \mathrm{s}(\mathrm{pr}(v)) = v$
A_5 $nat(v) \supset \mathrm{pr}(\mathrm{s}(v)) = v$
A_6 $\mathrm{pr}(0) = 0 \wedge \neg nat(v) \supset \mathrm{pr}(v) = 0$
A_7 $(nat(u) \wedge nat(v) \wedge v \neq 0) \supset (u = \mathrm{pr}(v)) \equiv \mathrm{pr}(v, u)$
A_8 $[(\neg nat(v) \supset \phi[v]) \wedge \phi[0] \wedge ((nat(v) \wedge v \neq 0 \wedge \phi[^{\mathrm{pr}(v)}/_v]) \supset \phi[v])] \supset \phi[v]$

Three implicit sorts make up the basic theoretical background of *Nqthm*: natural numbers, literal atoms and ordered pairs. The basic language of *Nqthm*, $\mathcal{L}_{Nq}$, contains the symbols required by the shell principle for each of these sorts. The basic theory of the system, T_{Nq}, is defined by the union of the axiom sets for each of the three sorts. Since every sort in T_{Nq} admits a well-founded relation, proofs by Noetherian induction are a priority for the system. Given a quantifier-free conjecture ϕ in conjunctive normal form, before trying to prove it by induction, it is supplied to a *simplifier* and a *rewriter*, which try to reduce ϕ to a propositional constant, $\top$ or $\bot$. When the reduction cannot be accomplished, an inductive proof for the resulting simplified formula or set of formulae is attempted.

The *simplifier* accepts only formulae in conjunctive normal form. Since a formula $\phi_1 \wedge \cdots \wedge \phi_n$ is T_{Nq}-valid iff each conjunct ϕ_i is also T_{Nq}-valid, each ϕ_i is dealt with independently from the other conjuncts. With the purpose of establishing the validity

[2] See [Boyer & Moore 79], chapters II and III. The original formulation of the theory of *Nqthm* employs a functional first-order language, i.e. a language in which every formula is an equation. The original notation has been replaced throughout this text by a language that contains both function and predicate symbols. Many-sorted theories, on the other hand, are defined in appendix D of this dissertation.

Shell element	Symbol	Properties/Intended Meaning
constructor	s	successor function
destructor	pr	$\mathrm{pr}(\mathrm{s}(v)) = v$
bottom constant	0	natural number 0
well-founded relation	pr	$\mathrm{pr}(u, v) \equiv v = \mathrm{s}(u)$
sort-recogniser	*nat*	$nat(v)$ iff v is a natural number

Table 3.1: Natural Numbers in *Nqthm*

of ϕ_i, each of its literals is simplified in turn, starting with $\phi_{i,1}$. The simplification of $\phi_{i,j}, 1 \leq j \leq m_i$, involves putting ϕ_i in conditional form,

$$\bigwedge_{k=1}^{\substack{k \neq j \\ m_i}} \neg\phi_{i,k} \quad \supset \quad \phi_{i,j}$$

and replacing it with the sequent $\Phi - \{\neg\phi_{i,j}\} \rightarrow \phi_{i,j}$, where $\Phi = \{\neg\phi_{i,k} \mid 1 \leq k \leq m_i\}$. $\phi_{i,j}$ is thereafter rewritten under the set of *assumptions* $\Phi - \{\neg\phi_{i,j}\}$. If $\phi'_{i,j}$ is the final rewritten literal, considering that rewrite rules are derived from T_{Nq}-equivalences, then

$$\Phi - \{\neg\phi_{i,j}\} \quad \rightarrow \quad (\phi_{i,j} \equiv \phi'_{i,j}) \quad (*)$$

is valid in T_{Nq}. If the original notation is restored, considering the distributivity of conditionals over biconditionals[3], it follows from $(*)$ that

$$(\bigwedge_{k=1}^{\substack{k \neq j \\ m_i}} \neg\phi_{i,k} \supset \phi_{i,j}) \equiv (\bigwedge_{k=1}^{\substack{k \neq j \\ m_i}} \neg\phi_{i,k} \supset \phi'_{i,j})$$

is valid in T_{Nq}, i.e. $T_{Nq} \models \phi_i \equiv \phi_i[\![^{\phi'_{i,j}}/_{\phi_{i,j}}]\!]$. As a result, the original clause is replaced by the simplified equivalent clause in which $\phi'_{i,j}$ substitutes for $\phi_{i,j}$. If $\phi'_{i,j}$ is the propositional constant $\top$, ϕ_i is T_{Nq}-valid (since it is a disjunction), and is then replaced by $\top$ as well. If $\phi'_{i,j}$ is the constant $\bot$, it is dropped from ϕ_i, otherwise the conjunct $\phi_i[\![^{\phi'_{i,j}}/_{\phi_{i,j}}]\!]$ is preserved. In the last two cases, simplification restarts from literal $\phi_{i,j+1}$.

[3] $(\phi \supset (\psi \equiv \rho)) \equiv ((\phi \supset \psi) \equiv (\phi \supset \rho))$ is a tautology. Concerning sequents, they are defined in appendix D.

The *rewriter* in *Nqthm* works at two levels. It first checks the sort of objects present in a composite term: whenever the corresponding sorts do not satisfy the restrictions for the expression, it is rewritten to $\bot$. For instance, given the equality

$$t_1 = t_2$$

if t_1 and t_2 belong to distinct sorts, the equation is evaluated to $\bot$. *Nqthm* also makes use of conditional rewrite rules of the form[4]

$$\psi_1 \wedge \cdots \wedge \psi_n \quad \rightarrow \quad \delta \Rightarrow \delta' \quad (*)$$

where $\psi_1, \ldots, \psi_n$ are the *conditions* (or hypotheses) of the rule, and $\delta \Rightarrow \delta'$ is its *body*. They are generated from T_{Nq}-valid sequents of the form $\{\psi_1, \ldots, \psi_n\} \rightarrow \delta \equiv \delta'$ (formula rewrite rule) or $\{\psi_1, \ldots, \psi_n\} \rightarrow \delta = \delta'$ (term rewrite rule). In its cooperation with the simplifier, the rewriter works in the presence of a set of assumptions. If ϕ is a formula to be proven from assumptions $\Phi = \{\phi_1, \ldots, \phi_n\}$, and R is a conditional rewrite rule of form $(*)$, the system first tries to establish whether there is an instance of δ that is syntactically identical to a subexpression $\widehat{\phi}$ of the conjecture. If that is the case, the corresponding substitution, σ, is applied to the conditions of the rule, which have then to be established from the set of assumptions. If there is an additional substitution τ (for the variables still free in the chosen instance of the conditions) such that

$$\Phi \quad \rightarrow \quad \tau(\sigma\psi_1) \wedge \cdots \wedge \tau(\sigma\psi_n)$$

is valid in the theory, by an application of *modus ponens*[5], $\Phi \rightarrow \sigma\delta \Rightarrow \sigma\delta'$ is also valid in T_{Nq}. If ϕ' be the formula generated from ϕ after $\widehat{\phi}$ has been replaced with $\sigma\delta'$, then

---

[4] This rule is represented in *Nqthm* as

```
(IMPLIES   (AND h1 ... hn)
           (EQUAL t1 t2))
```

where each h_i is a hypothesis of the rule.

[5] Or rather of the derived rule

$$\frac{\phi \supset (\psi \supset \rho) \qquad \phi \supset \psi}{\phi \supset \rho}$$

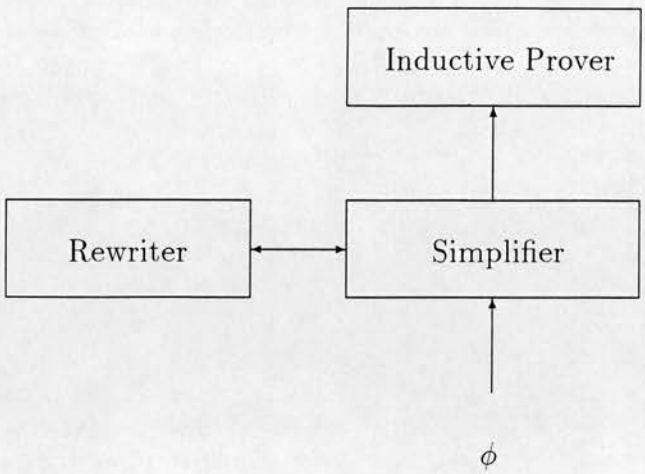


Figure 3.1: *Nqthm*: original configuration

$\Phi \rightarrow \phi \equiv \phi'$ is T_{Nq}-valid. Whenever ϕ' is the constant $\top$, it is possible to conclude that

$$T_{Nq} \models \Phi \rightarrow \phi$$

To prove that $\tau(\sigma\psi_1) \wedge \cdots \wedge \tau(\sigma\psi_n)$ follows from the set of assumptions Φ, the rewriter is recursively called for each conjunct $\tau(\sigma\psi_i)$, until each of them is reduced to $\top$ under Φ, whenever possible.

The links between the three main modules of *Nqthm* are indicated in figure 3.1.

3.1.2 Linear Arithmetic Procedure

An initial difficulty with respect to the introduction of decision procedures in *Nqthm* was the choice of a suitable decidable theory for the examination of arithmetical verification conditions. Although Peano arithmetic has decidable subtheories (e.g. Presburger arithmetic and linear arithmetic), due to the complexity of their decision procedures, the resulting hybrid system that contains them is inefficient, according to Boyer and Moore. The absence of decision procedures, on the other hand, would mean that heuristic modules would have to handle inequalities and face the risk of exponential explosions related to the transitivity axiom for $<$ (or $\leq$)[6].

[6] Linear arithmetic (LA) is formulated in the language $\mathcal{L}_{LA} = \{<, +, -_2, 0, 1, 2, \ldots, n, \ldots\}$. For more comments about the inefficiency of decision procedures for PrA and the difficulties in handling $\leq$,

An alternative solution is the selection of a decidable subclass for linear arithmetic that admits a less complex decision procedure. One of such subclasses results from model theoretical properties of DAG, the theory of dense ordered Abelian groups without endpoints, formulated in $\mathcal{L}_{DAG} = \{0, 1, +, <\}$. Although this theory does not admit models having natural or integer numbers as universe, according to lemma C.5.1, decision procedures for DAG can be used to delimit a decidable subclass for LA, made up of universal formulae, and supply the required information for the construction of a *linear procedure* (i.e. a decision procedure for LA w.r.t. a quantifier-free subclass). This application illustrates the beneficial role of decidable subclasses even in the context of decidable theories.

Instead of DAG, its conservative extension DAG^*, which includes $-$ and $\leq$ respectively as new function and predicate symbols, is selected as the reference theory. Possible definitions for the new symbols in terms of the original language are

$$\begin{aligned} v_1 \leq v_2 &\equiv (v_1 < v_2) \vee (v_1 = v_2) \\ v_1 - v_2 = v_3 &\equiv v_1 = v_2 + v_3 \end{aligned}$$

The linear procedure based on DAG^* has an intricate structure. It has two slightly distinct versions, one linked to the simplifier and the other to the rewriter. In the first case, let ϕ be a (quantifier-free) conjecture in conjunctive normal form. If its conjunct $\phi_i(v_1, \ldots, v_n)$ belongs to $\mathcal{L}_{LA}$, for each free variable v_j that occurs in ϕ, a corresponding *linearisation hypothesis* of the form $nat(v_j)$ is added to an initially empty set of assumptions Φ_1, where $nat(v_j)$ denotes that v_j is a natural number. As a result, considering that, for any formula ψ of $\mathcal{L}_{LA}$,

$$T_{Nq} \models \Phi_1 \rightarrow \psi \quad \text{iff} \quad LA \models \psi \quad (*)$$

where Φ_1 is the set of linearisation hypotheses for ψ, attention can be shifted from T_{Nq} to linear arithmetic. A second group of *linearisation hypotheses*, Φ_2, is built as follows: if the subterm $t_j - u_j$, $1 \leq j \leq p$, occurs in ϕ_i, then $(u_j \leq t_j) \in \Phi_2$, where p denotes the total number of subexpressions in ϕ_i dominated by $-$. Hence

$$\text{if} \quad T_{\exists^*} \models \phi_i' \quad \text{then} \quad LA \models \Phi_2 \rightarrow \phi_i \quad (**)$$

see [Boyer & Moore 88], p. 88 and 85.

where $\phi_i' \stackrel{s}{=} \bigwedge_{j=1}^{n}(0 \leq v_j) \supset \phi_i$, $\mathfrak{Z}^* = \langle \mathbb{Z}, 0, 1, -_2, +, <, \leq \rangle$, and $\mathbb{Z}$ is the set of integers[7]. Conditions of the form $0 \leq v$ are necessary to guarantee that variables range only over natural numbers (which can be linked to the set of non-negative integers). The formula that is supplied to the decision procedure for DAG^* is derived from the negation of ϕ_i' by a process that puts it into *linear form*. It involves

(i) the removal of conditionals and the stratification of negation under other connectives (based on the application of the de Morgan law for negation over e.g. conjunctions, as described in [Bundy 91]),

(ii) the elimination of occurrences of $\neg$, $=$ and $<$, which requires an ordered set of rewrite rules[8],

$$\begin{array}{rcl} \neg(t_1 = t_2) & \Rightarrow & t_1 < t_2 \vee t_2 < t_1 \\ \neg(t_1 < t_2) & \Rightarrow & t_2 \leq t_1 \\ \neg(t_1 \leq t_2) & \Rightarrow & t_2 < t_1 \\ t_1 = t_2 & \Rightarrow & t_1 \leq t_2 \wedge t_2 \leq t_1 \\ t_1 < t_2 & \Rightarrow & t_1 + 1 \leq t_2 \end{array}$$

(iii) the reordering of terms inside inequalities, which requires

$$t_1 \leq t_2 \quad \Rightarrow \quad t_1 - t_2 \leq 0$$

(iv) the exhaustive application of rules

$$\begin{array}{rcl} k_1 \times (k_2 \times t) & \Rightarrow & (k_1 \times k_2) \times t \\ k \times (t_1 + t_2) & \Rightarrow & k \times t_1 + k \times t_2 \\ k \times (t_1 - t_2) & \Rightarrow & k \times t_1 - k \times t_2 \\ t_1 - (t_2 + t_3) & \Rightarrow & (t_1 - t_2) - t_3 \end{array}$$

where k, k_1, k_2 denote natural numbers[9], and

---

[7] Concerning Φ_1, it has to be recalled that *Nqthm* works with distinct sorts, and that the linear procedure derives results only about (natural) numbers. For this reason, sort restrictions have to be imposed upon all the variables of any formula supplied to this procedure.

[8] As the set of rules is ordered, each of them has to be exhaustively applied before its successor is considered. Rule $t_1 < t_2 \Rightarrow t_1 + 1 \leq t_2$ in particular is invalid in DAG^*, given that

$$DAG^* \quad \not\models \quad (t_1 < t_2) \equiv (t_1 + 1 \leq t_2)$$

[9] With respect to $\mathcal{L}_{DAG^*}$, k and $k \times t$ respectively represent abbreviations for

$$\underbrace{1 + 1 + \cdots + 1}_{k \text{ occurrences}} \quad \underbrace{t + t + \cdots + t}_{k \text{ occurrences}}$$

Phase	Action	Notation
1	Introduction of linearisation hypotheses (first set)	$nat(v_i)$
2	Introduction of linearisation hypotheses (second set)	$u_j \leq t_j$ (if $t_j - u_j$ occurs in ϕ)
3	Restriction of range of variables (non-negative integers)	$0 \leq v_i$
4	Reduction of atoms to linear form	$\pm k_1 \times v_1 \pm \cdots \pm k_n \times v_n \leq 0$

Table 3.2: Linear Arithmetic Procedure in *Nqthm*

(v) the collection of summands and subtrahends containing the same variable and the reduction of this formula to disjunctive normal form, which results in a new formula, ϕ_i'', such that

$$T_{\mathfrak{Z}^*} \models \phi_i' \equiv \neg\phi_i'' \quad (***)$$

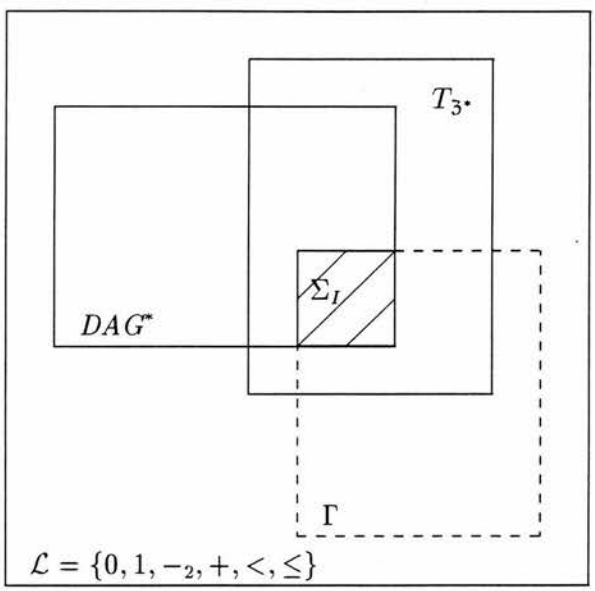
ϕ_i'' is thereafter supplied to the decision procedure for DAG^*. If it is unsatisfiable in DAG^*, as a consequence of theorem C.5.1, it is also unsatisfiable in $T_{\mathfrak{Z}^*}$. This stems from the fact that $\mathfrak{Z}^*$ is a substructure of one of the models of DAG^*, and, as a result, every quantifier-free (and hence universal) formula that is valid in DAG^* is also valid in $\mathfrak{Z}^*$. From $(***)$, ϕ_i' is a theorem in $T_{\mathfrak{Z}^*}$, and from $(**)$, $LA \models \Phi_2 \rightarrow \phi_i$. Finally, from $(*)$, $T_{Nq} \models \Phi_1 \cup \Phi_2 \rightarrow \phi_i$. To complete the proof, it is necessary to establish that ϕ_i follows from the negation of each of the hypotheses in $\Phi_1 \cup \Phi_2$. When that is the case[10],

$$T_{Nq} \models \phi_i$$

The main features of the linear procedure are summarised in table 3.2. The relationship between DAG^* and $T_{\mathfrak{Z}^*}$ is illustrated in figure 3.2. An example that illustrates all the phases of the procedure follows.

Example 3.1.1 *Let ϕ be the conjecture $x \leq y \wedge y \leq x \supset x = y$, whose conjunctive normal form is*

[10] See appendix E.



DAG^*	Theory of dense ordered Abelian groups without endpoints
$T_{\mathfrak{Z}^*}$	Theory of $\mathfrak{Z}^* = \langle \mathbb{Z}, 0, 1, -_2, +, <, \leq \rangle$
Γ	Class of universal formulae
Σ_I	Decidable subclass of type I for $T_{\mathfrak{Z}^*}$

Figure 3.2: DAG^* & $T_{\mathfrak{Z}^*}$

$$\neg(x \leq y) \ \vee \ \neg(y \leq x) \ \vee \ x = y$$

The application of the linear arithmetic procedure involves the following steps.

i. Since ϕ has only two variables, the first set of linearisation hypotheses is

$$\Phi_1 = \{nat(x),\ nat(y)\}$$

Φ_2, on the other hand, is empty, since $-$ does not occur in ϕ.

ii. ϕ' is generated from ϕ when restrictions are imposed on the range of its variables,

$$((0 \leq x) \wedge (0 \leq y)) \ \supset \ (\neg(x \leq y) \vee \neg(y \leq x) \vee x = y)$$

iii. $\neg\phi'$ is linearised to ϕ'', as follows

$$-(1 \times x) \leq 0 \ \wedge \ -(1 \times y) \leq 0 \ \wedge \ (1 \times x) - (1 \times y) \leq 0 \ \wedge \ (1 \times y) - (1 \times x) \leq 0$$
$$\wedge$$
$$1 + (1 \times x) - (1 \times y) \leq 0 \ \wedge \ 1 + (1 \times y) - (1 \times x) \leq 0$$

iv. ϕ'' is supplied to the decision procedure for DAG^*. Since it is identified as unsatisfiable in DAG^*, ϕ' is a theorem in $T_{3\bullet}$ and ϕ is a theorem in LA, hence

$$T_{Nq} \models \Phi \rightarrow \phi$$

v. It is then necessary to show that ϕ can be also derived from the negation of each of the linearisation hypotheses, as for instance

$$T_{Nq} \models \neg nat(x) \rightarrow \phi$$

However, this is not possible, since $\top \leq \bot \wedge \bot \leq \top \supset \top = \bot$ is not *a T_{Nq}-theorem*[11]*. Therefore $T_{Nq} \not\models \phi$.* □

[11] See [Boyer & Moore 88], p. 89-90.

The integration of the linear procedure with the rewriter follows similar guidelines. If

(i) $\psi_1 \wedge \cdots \wedge \psi_n \rightarrow \delta \Rightarrow \delta'$ is a conditional rewrite rule,

(ii) $\Phi = \{\phi_1, \ldots, \phi_n\}$ is the current set of assumptions, and

(iii) ψ_i is a conjunct in the condition of the above rule which belongs to $\mathcal{L}_{LA}$,

then the sequent $\Phi' \rightarrow \psi_i$, where Φ' is the strictly arithmetical subset of Φ (i.e. $\Phi' \subset Fml_{\mathcal{L}_{LA}}$), is supplied to the simplifier and dealt with as any conjunct of a conjecture: linearisation hypotheses are identified and the non-negative status of all variables is explicitly added to the formula, which is then negated, linearised and supplied to the decision procedure[12]. If the resulting linearised formula is recognised as T_{Nq}-unsatisfiable, then

$$T_{Nq} \models \Phi \rightarrow \psi_i$$

otherwise no conclusion about the derivability of ψ_i from Φ in LA can be taken.

The decision procedure for DAG^* is derived from Hodes' algorithm, which is based on *quantifier elimination*: given a formula of the form $(\exists v)\phi$, where ϕ is a quantifier-free conjunction of atoms,

i. if ϕ has the form $(v = t[\not v]) \wedge \phi'$, the existential quantifier is removed once the DAG-equivalence

$$(\exists v)(v = t[\not v] \wedge \phi'[v]) \quad \equiv \quad \phi'[^t/_v]$$

is applied.

ii. if ϕ has the form $\bigwedge_{i=1}^n t_i[\not v] < v$ or $\bigwedge_{i=1}^n v < t_i[\not v]$, $(\exists v)\phi$ is valid in DAG, and can be replaced by any DAG-valid atomic sentence, e.g. $0 = 0$.

iii. if ϕ has the form $\bigwedge_{i=1}^n t_i[\not v] < v \wedge \bigwedge_{j=1}^m v < u_j[\not v]$, the existential quantifier can be dropped after the application of the DAG-equivalence

[12] A more precise description of this stage of the procedure is presented in section 3.2.

$$(\exists v)\left(\bigwedge_{i=1}^{n} t_i[\not\!\!\!\not] < v \wedge \bigwedge_{j=1}^{m} v < u_j[\not\!\!\!\not]\right) \equiv \bigwedge_{1\le i\le n}^{1\le j\le m} t_i < u_j$$

A specialised procedure for DAG^* has to take into account that *Nqthm* operates only in the quantifier-free fragment of $\mathcal{L}_{DAG^*}$ (and therefore has to eliminate variables only), that formulae are supplied in disjunctive normal form, and that atoms are represented in linear form. As a result, there is only one case to be considered,

$$\bigwedge_{i=1}^{n}(t_i[\not\!\!\!\not] + k_i v \le 0) \wedge \bigwedge_{j=1}^{m}(u_j[\not\!\!\!\not] - l_j v \le 0) \quad (*)$$

where $k_i, l_j \in \mathbb{N}$. Since this formula is quantifier-free, it is satisfiable iff its existential closure is also satisfiable. The elimination of quantifiers from the equi-satisfiable formula follows basically the same properties adopted in the case of Hodes' algorithm, and $(\exists v)(*)$ can be replaced with its DAG^*-equivalent version,

$$\bigwedge_{1\le i\le n}^{1\le j\le m} l_j t_i + k_i u_j \le 0$$

which is unsatisfiable iff $(*)$ is unsatisfiable.

The linear procedure can be extended to deal with an expansion of $\mathcal{L}_{LA}$ that includes undefined constant, function and predicate symbols. For the only component that requires modification, the decision procedure for DAG^*, two alternative versions are available. The first one, based on theorem 2.2.1, eliminates undefined symbols of quantifier-free formulae and generates an equi-valid formula that is supplied to the algorithm. The second solution, which explores certain properties of the canonical form to which literals are reduced, amounts to the *elimination of composite terms*. Let ϕ be a conjunction of literals,

$$\bigwedge_{i=1}^{n}(u_i[\![\not\!\!\!\not]\!] + k_i t \le 0) \wedge \bigwedge_{j=1}^{m}(w_j[\![\not\!\!\!\not]\!] - l_j t \le 0)$$

where t is a term whose dominant symbol is not sum or subtraction. This formula is equi-satisfiable to

$$(\exists v)\left(\bigwedge_{i=1}^{n}(u_i[\not\ell] + k_i v \leq 0) \wedge \bigwedge_{j=1}^{m}(w_j[\not\ell] - l_j v \leq 0)\right)$$

where v does not occur in ϕ, to which the algorithm can be applied unchanged. When there is a single occurrence of an undefined function or predicate symbol in a conjunction of literals, both solutions above coincide.

In the presence of additional non-logical symbols, the elimination of variables and/or composite terms takes place according to an order imposed on the set of terms. The *heaviest terms* are eliminated in the first place, through the cancellation of terms with distinct signs (positive or negative) in distinct inequalities.

Definition 3.1.1 *Let t_1 and t_2 be terms of $\mathcal{L}_{Nq}$.*

i. t_1 *is* heavier *than* t_2 *iff*

   (a) the number of occurrences of variables in t_1 is greater than in t_2, or,

   (b) in case of the number of occurrences of variables is equal, the number of function symbols in t_1 is greater than in t_2, or,

   (c) t_2 lexicographically precedes t_1.

ii. t_1 *and* t_2 *have the* same weight *iff t_1 is not heavier than t_2 and t_2 is not heavier than t_1.*

An immediate consequence of this definition is that two terms have the same weight iff they are syntactically identical. The elimination of heaviest terms preserves satisfiability: Φ is satisfiable iff the set of assumptions generated from Φ after the heaviest terms have been removed is also satisfiable.

3.2 Hypothesis Introduction in *Nqthm*

The linear procedure described in the previous section has limitations at two levels: it does not represent a decision procedure for LA (even though LA is decidable), and,

as it stands, it cannot deal with extensions of LA, except those containing undefined non-logical symbols. Both limitations can be partially overcome by the introduction of additional formulae in the set of assumptions of a conjecture.

3.2.1 Additional Lemmas

The incompleteness of the linear procedure vis-a-vis decision procedures for LA is due to the existence of $\mathfrak{Z}^*$-valid quantifier-free formulae, e.g.

$$3 * x \neq 2$$

that cannot be identified as such by the decision procedure for DAG^*. Incompleteness at this level can be reduced by means of *explicitation of arithmetical hypotheses*, a process that expands quantifier-free formulae which are valid in LA but invalid in DAG^* until their validity in LA becomes recognisable. If ϕ is such a formula, the mechanism of explicitation of arithmetical hypotheses selects a set of quantifier-free theorems Φ in LA, such that $DAG^* \models \Phi \rightarrow \phi$ and, as a result, $LA \models \Phi \rightarrow \phi$. Since $\Phi \subset LA$, then

$$LA \models \phi$$

This mechanism can been successfully applied to some verification conditions.

Example 3.2.1 *Let P be a program that searches for objects in a* table, *i.e. an array of (even) length d in which odd positions contain* keys, *while each even position stores the value associated with the key of the previous odd position*[13]. *A verification condition (v.c.) for this program establishes that, whenever an index i corresponds to a key, then $i+1 \leq d$. If l and k are variables over natural numbers, this v.c. can be represented as*

$$(0 < k \wedge 0 < l \wedge 2k + 1 \leq 2l) \supset (2k + 2 \leq 2l) \quad (*)$$

where $2l = d$ and $2k + 1 = i$ (if i identifies a key). $(*)$ is valid in LA, but invalid in DAG^*. Therefore the linear procedure cannot recognise it as a theorem. However, the inclusion of the formula

[13] See [Boyer & Moore 88], p. 90.

$$(k < l) \supset (k + 1 \leq l)$$

which represents the property that the difference between two distinct natural numbers is not smaller than 1, *as a new assumption for* $(*)$ *transforms it into a* DAG^**-valid formula and a recognisable LA-theorem.* □

Arithmetical hypotheses must be invalid in DAG^* to contribute towards the identification of additional arithmetical theorems. For otherwise if $\Phi \subset DAG^*$, then, as $DAG^* \not\models \phi$, it must follow that $DAG^* \not\models \Phi \rightarrow \phi$, and once again no conclusion about the validity of ϕ in LA can be obtained. This mechanism faces serious search problems, since virtually all arithmetical theorems that are invalid in DAG are suitable candidates for arithmetical hypotheses. Such a solution therefore is not explicitly employed in *Nqthm*, and formulae of $\mathcal{L}_{LA}$ that are invalid in DAG^* are automatically forwarded to the inductive module[14]. The only linearisation stage where arithmetical hypotheses are implicitly introduced takes place when rule

$$t_1 < t_2 \;\Rightarrow\; t_1 + 1 \leq t_2,$$

which is invalid in DAG, is applied.

The second deficiency of the linear arithmetic procedure — its inability to deal with extensions of LA — also has a negative impact on applications to program verification. For instance, in the verification of the Euclid algorithm for the greatest common divider, it is necessary to prove that, whenever $x < y$ and $z|x$ (i.e. z *divides* x), then $z|y$ iff $z|(y - x)$, i.e.

$$(x < y \wedge z|x) \supset (z|y \equiv z|(y - x))$$

Also, in the linearisation of bidimensional arrays, one of the conditions to be verified states that elements occupying distinct positions in a bidimensional array, represented as $\langle i_1, j_1 \rangle$ and $\langle i_2, j_2 \rangle$, must be mapped into distinct positions of the corresponding

[14] This is actually the case of conjecture $(*)$ above; see [Boyer & Moore 88], p. 89-91.

linear array, $i_1 + d \times j_1$ and $i_2 + d \times j_2$, where d is the rank of the bidimensional array, i.e.

$$(i_1 < d \wedge i_2 < d) \supset (i_1 + d \times j_1 = i_2 + d \times j_2 \equiv (i_1 = i_2 \wedge j_1 = j_2))$$

Both formulae lie outside the domain of the procedure for LA, due to the occurrence of symbols ($|$ and $\times$) that do not belong to $\mathcal{L}_{LA}$. Since many verification conditions involve function and predicate symbols over other sorts, the impossibility of recognising theorems of this nature reduces substantially the role of the new module inside the prover[15].

Given that *Nqthm* operates in the quantifier-free fragment of $\mathcal{L}_{Nq}$, it is possible to consider the application of the additional hypothesis lemma to overcome the above limitation. A new module of the system is responsible for the introduction of hypotheses, after several consultations of a database of definitions and theorems about recursive operations and relations. This process makes the interfacing of the simplifier to the decision procedure more complex than the mere application of Hodes' algorithm to arithmetical conjectures[16].

The validity of the process of introducing additional hypotheses has been discussed in section 2.5. It is requested by the system whenever a conjunction of inequalities that contains undefined symbols is identified as satisfiable by the linear procedure. The criterion for the selection of theorems, in the form of (conditional) inequalities (stored by the system), follows the same guideline for the elimination of multiplicands: a new inequality is added to the formula only if it allows the elimination of the heaviest multiplicand in that formula, and does not introduce any heavier term. After selection, they are instantiated and conjoined to the formula, until unsatisfiability is detected, whenever possible. This guideline is more clearly illustrated in the example that follows.

Example 3.2.2 *Let ϕ be the conjecture*

$$l \leq min(x) \wedge 0 < k \supset l < max(x) + k$$

[15] Comments in this respect are presented in [Boyer & Moore 88], p. 91, 94.

[16] See [Boyer & Moore 88], p. 90.

Once $\neg\phi$ has been linearised to

$$l - min(x) \leq 0 \ \wedge \ 1 - k \leq 0 \ \wedge \ max(x) + k - l \leq 0 \quad (*)$$

the resulting formula is supplied to the decision procedure for DAG^*. Since it is satisfiable, the procedure restarts from the linearised negated conjecture, where the heaviest multiplicand to be found in any of the atoms is $min(x)$. After consulting a list of stored lemmas, the theorem[17]

$$min(v) \leq max(v)$$

which can be instantiated and linearised to

$$min(x) - max(x) \leq 0$$

is selected and conjoined to $(*)$. Since it is suitable for the removal of the heaviest term, the expanded formula

$$min(x) - max(x) \leq 0 \ \wedge \ l - min(x) \leq 0 \ \wedge \ 1 - k \leq 0 \ \wedge \ max(x) + k - l \leq 0$$

is then supplied to the decision procedure: first $min(x)$ is eliminated,

$$l - max(x) \leq 0 \ \wedge \ 1 - k \leq 0 \ \wedge \ max(x) + k - l \leq 0$$

followed by $max(x)$,

$$l + k - l \leq 0 \ \wedge \ 1 - k \leq 0$$

and then the variable k, resulting in $1 \leq 0$, which is unsatisfiable in DAG^*. Hence ϕ is a DAG^*-theorem. □

There are cases where the additional hypothesis is a conditional formula. As a result, besides checking the conjunction of the new hypothesis with the negated conjecture for unsatisfiability, it is necessary to prove its associated condition from the current set of hypotheses as well.

[17] To simplify the presentation, additional conditions limiting the range of variables to non-negative integers have been omitted.

Example 3.2.3 *Given the following function and predicate symbols, defined over lists,*

$memb(x, s)$	*x occurs in a list s*
$len(s) = l$	*l is the length of a list s*
$del(x, s) = s'$	*s' is obtained from a list s after all occurrences of x have been deleted*

let ϕ be the conjecture

$$memb(z, a) \wedge w + len(a) \leq k \wedge 0 \leq v \supset w + len(del(z, a)) < k + v$$

After the linearisation of its negation,

$$memb(z, a) \wedge w + len(a) - k \leq 0 \wedge -v \leq 0 \wedge k + v - w - len(del(z, a)) \leq 0$$

since unsatisfiability cannot be detected, additional hypotheses for the heaviest multiplicand, $len(del(z, a))$, are sought. A conditional theorem,

$$memb(v, u) \supset len(del(v, u)) < len(u)$$

is a suitable additional hypothesis, provided that its antecedent can be proved. As in the case of conditional rewriting, this task is given to the rewriter. Assuming that $memb(z, a)$ is amongst the current assumptions, the rewriter can derive as valid the formula

$$len(del(z, a)) < len(a)$$

which, after linearisation, is conjoined to the linearised negated conjecture, thus allowing unsatisfiability to be found[18].

$$1-len(a)+len(del(z,a)) \leq 0 \wedge memb(z,a) \wedge w+len(a)-k \leq 0 \wedge -v \leq 0 \wedge k+v-w-len(del(z,a)) \leq 0$$
$$1 - len(a) + k + v - w \leq 0 \wedge memb(z, a) \wedge w + len(a) - k \leq 0 \wedge -v \leq 0$$
$$1 + k + v - w + w - k \leq 0 \wedge memb(z, a) \wedge -v \leq 0$$
$$1 + v \leq 0 \wedge memb(z, a) \wedge -v \leq 0$$
$$1 \leq 0 \wedge memb(z, a)$$
$$\perp$$

□

[18] As in the previous example, restrictions concerning the range of variables have been omitted when not necessary for the derivation of $\perp$.

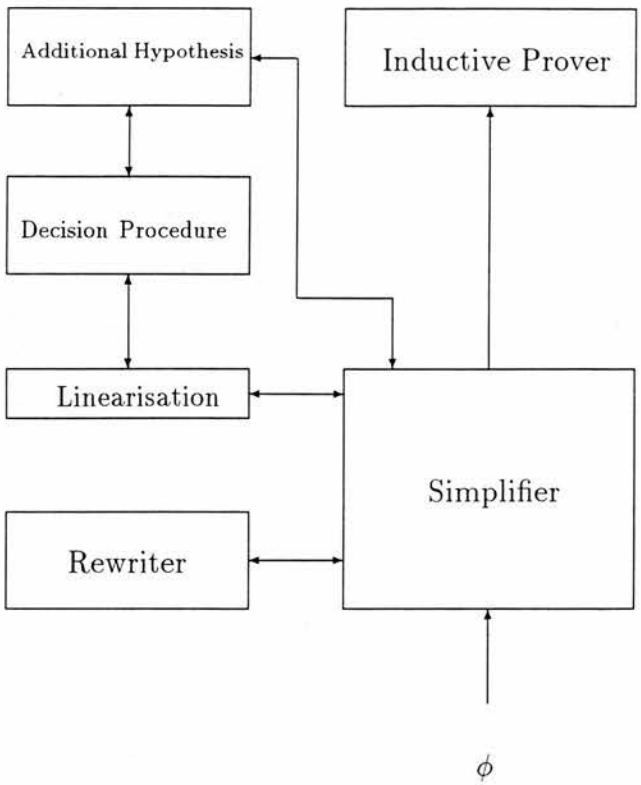


Figure 3.3: *Nqthm* & the Linear procedure

The complete integration mechanism is described in table 3.3. A block diagram can be found in figure 3.3.

3.2.2 Optimisations

Once a disjunction of literals ϕ is supplied to the simplifier of *Nqthm*, while one of them is simplified, the remaining literals are negated and added to an assumption set. When the simplifier operates in conjunction with the linear procedure, the heaviest multiplicands are eliminated from the chosen literal after the introduction of heuristically selected additional hypotheses, whenever necessary. This operation has to be repeated, in principle, for each literal in ϕ. However, exploring the fact that

$$\models \Gamma \rightarrow \phi \quad \text{iff} \quad \models \Gamma \cup \{\neg\phi\} \rightarrow \phi$$

a single assumption set can be defined, containing the complement of every literal in ϕ. After the removal of the heaviest multiplicands from this set, the simplification of

Simplifier. Given a disjunction of literals $\phi_1 \vee \cdots \vee \phi_n$,

i. If $(\phi_1 \vee \cdots \vee \phi_n) \notin Fml_{\mathcal{L}_{LA}}$, the simplifier successively supplies the sequents $\Phi \rightarrow \phi_i, 1 \leq i \leq n$, to the rewriter.

ii. If $\phi \in Fml_{\mathcal{L}_{LA}}$, the simplifier supplies it to the linearisation module.

If all the disjuncts of the simplified formula $\phi'_1 \vee \cdots \vee \phi'_n$ are identical to $\bot$, or at least one of them is identical to $\top$, the process halts, otherwise $\phi'_1 \vee \cdots \vee \phi'_n$ is supplied to the inductive prover.

Rewriter. Let $\Phi \rightarrow \phi[\![\epsilon]\!]$ be a sequent. If $\psi_1, \ldots, \psi_m \rightarrow \delta_1 \Rightarrow \delta_2$ is a conditional rewrite rule, and if there is a substitution σ such that $\sigma\delta_1 \stackrel{s}{=} \epsilon$, then the sequents $\Phi \rightarrow \sigma\psi_1, \ldots, \Phi \rightarrow \sigma\psi_m$ are supplied to the simplifier. In the event they are all reduced to $\top$, $\Phi \rightarrow \phi$ is replaced by $\Phi \rightarrow \phi[\![^{\sigma\delta_2}/_{\epsilon}]\!]$.

Linearisation module. If $\phi \in Fml_{\mathcal{L}_{LA}}$, then

(a) a set of linearisation assumptions for ϕ is built,
(b) conditions stating the non-negative status of variables are introduced in ϕ,
(c) the extended formula is negated, and its atoms are linearised, and
(d) the resulting formula is put into disjunctive normal form, and then supplied to the decision procedure for DAG^*.

If the decision procedure returns $\bot$, the linearisation module transforms ϕ into $\top$ and the process is complete. Otherwise ϕ is returned unchanged.

Decision procedure. If $\psi_1 \wedge \cdots \wedge \psi_m$ is a linearised formula of $\mathcal{L}_{LA}$ and it is reduced to $\bot$ by the decision procedure for DAG^*, the process halts. Otherwise, if it reduced to $\top$ and at least one of the literals contains deviant symbols, then $\psi_1 \wedge \cdots \wedge \psi_m$ is supplied to the additional hypothesis mechanism.

Additional hypothesis mechanism. If $\psi_1 \wedge \cdots \wedge \psi_m$ is a linearised formula whose heaviest term is t, an additional hypothesis $\widehat{\psi}$ involving t is sought. If no suitable hypothesis is found, the original formula is returned unchanged. Otherwise

i. If $\widehat{\psi}$ is a literal or a conjunction of literals, it is then conjoined to $\psi_1 \wedge \cdots \wedge \psi_m$, and the expanded formula is supplied to the decision procedure.

ii. If $\widehat{\psi}$ is a conditional formula, its consequent is conjoined to $\psi_1 \wedge \cdots \wedge \psi_m$, and the expanded formula is supplied to the decision procedure. If unsatisfiability is detected, the conditions of $\widehat{\psi}$ are individually supplied to the simplifier. If they are all reduced to $\top$, $\psi_1 \wedge \cdots \wedge \psi_m$ is replaced by $\bot$.

Table 3.3: Modules of *Nqthm*

each literal can then be performed[19].

Restrictions have to be observed, however, in the application of conditional rules. Although the above property allows the inclusion of the complement of a literal ϕ_i in its own assumption set, the conditions of a rule cannot be established based on the complement of ϕ_i, or on any formula that depends upon the complement of this literal. The example that follows justifies such restriction.

Example 3.2.4 *Let ϕ be the clause*

$$(a + c = b) \vee \neg(a + 1 < b)$$

and let γ be its second literal, $(a+1 < b)$. If γ is selected for simplification, the set of assumptions is built with the complement of all literals, i.e. $\Phi = \{(a + c \neq b), (a + 1 < b)\}$. At the stage of rewriting γ, the following conditional lemma

$$v_1 < v_2 \rightarrow (v_1 + 1 < v_2 \equiv v_1 + 1 \neq v_2)$$

is selected, since the lhs expression of its body can be matched against a subexpression of γ, by means of substitution $\sigma = \{^{a}/_{v_1}, ^{b}/_{v_2}\}$. The corresponding instance of the condition of this lemma has then to be established from the current set of assumptions Φ, i.e.

$$\{(a + c \neq b), (a + 1 < b)\} \rightarrow a < b,$$

has to be proven, a task the decision procedure for DAG^* accomplishes, once the negation of $a < b$ is conjoined to Φ. Hence

$$DAG^* \models \Phi \rightarrow a + 1 < b \equiv a + 1 \neq b$$

The rule $a + 1 < b \Rightarrow a + 1 \neq b$ is then applied to γ, and the original clause is rewritten to a non-equivalent formula,

$$(a + c = b) \vee (a + 1 = b)$$

□

[19] See [Boyer & Moore 88], p. 97, for a discussion about the selection of multiplicands. The above property involving the sequents $\Gamma \rightarrow \phi$ and $\Gamma \cup \{\neg\phi\} \rightarrow \phi$ is proved in appendix E.

An incorrect step in the above rewriting sequence is the final representation of the transformed clause: since the complement of γ has been employed in the deduction, it has to be included in the final clause,

$$(a + c = b) \vee \neg(a + 1 < b) \quad \equiv \quad \{a + 1 < b\} \rightarrow ((a + c = b) \vee (a + 1 = b))$$

or, in standard first-order notation,

$$(a + c = b) \vee \neg(a + 1 < b) \quad \equiv \quad (a + c = b) \vee \neg(a + 1 < b) \vee (a + 1 = b)$$

Under this new formulation, equivalence is preserved. However, although sound, the process could not be regarded as the *simplification* of the original clause, since it just adds new literals to it.

To prevent this problem, Boyer and Moore adopted a mechanism that links a formula to every assumption from which it has been derived. It is possible then to trace back whether the complement of a literal is involved in a proof or not. For the example above, it would suffice to consider that

i. The condition of the instantiated rewrite rule is derived from the complement of the current literal, γ, after the decision procedure for DAG^* has been called. The rule condition should then be represented by the pair

$$\langle\ a < b,\ [(a + 1 < b)]\ \rangle$$

ii. The body of the rewrite rule has been obtained after its condition was proven from the complement of the current literal. Hence, it could be represented as

$$\langle\ a + 1 < b \equiv a + 1 \neq b,\ [a < b, (a + 1 < b)]\ \rangle$$

iii. The above rule is applied to γ, so its rewritten version has to be indicated as

$$\langle\ a + 1 = b,\ [\underline{\neg(a = 1 < b)},\ a + 1 < b \equiv a + 1 \neq b,\ a < b,\ \underline{(a + 1 < b)}]\ \rangle$$

At least in this example, by simple inspection of the list of assumptions involved in the transformation, it is clear that the complement of the current literal γ has been used

in the transformation of γ. Therefore the resulting formula, $(a + 1 = b)$, is rejected by the simplifier.

Another aspect to be considered is the requirement that clauses have to be proved under the complement of each individual linearisation hypothesis, once it has been established in their presence. To reduce the number of cases, it suffices to determine, as it has been done in example 3.2.4, which of the linearisation hypotheses are actually required in the proof of the clause, restricting attention to them thereafter[20].

3.2.3 Proving Verification Conditions

After the optimisations incorporated in the final system, the linear procedure, in conjunction with the hypothesis introduction module, has been able to prove a class of verification conditions more efficiently[21].

Example 3.2.5 *Let ϕ be the v.c.*

$$0 < ms(a)^4 + 2 \times ms(a)^2 \times ms(b) - ms(a)^2,$$

related to the termination of an algorithm where a unary function ms establishes a certain measure over a set of ordered pairs. After ϕ is negated and linearised, resulting in

$$ms(a)^4 + 2 \times ms(a)^2 \times ms(b) - ms(a)^2 \leq 0$$

it is supplied to the linear arithmetic procedure, which does not identify it as unsatisfiable in LA. Since multiplication is present, the formula does not belong to $\mathcal{L}_{LA}$, and additional hypotheses can be sought. The heaviest multiplicand in the inequality is $ms(a)^4$; the chosen additional hypothesis,

---

[20] Boyer and Moore discuss additional ways of reducing the cases to be proven in [Boyer & Moore 88], p. 100.

[21] This example has been taken from [Boyer & Moore 88], p. 101-3. v^n is an abbreviation for

$$\underbrace{v \times \cdots \times v}_{n \text{ occurrences of } v}$$

$$0 < v \;\rightarrow\; u \leq v \times u \quad (\dagger)$$

is first instantiated, resulting in

$$0 < ms(a) \;\rightarrow\; ms(a)^3 \leq ms(a)^4$$

After its condition $0 < ms(a)$ *is reduced to* $\top$*, the inequality is conjoined to the negated conjecture and the heaviest term is eliminated. Since the new formula*

$$ms(a)^3 + 2 \times ms(a)^2 \times ms(b) - ms(a)^2 \leq 0$$

is not DAG-unsatisfiable either, the next multiplicand,* $ms(a)^2 \times ms(b)$ *(which is heavier than* $ms(a)^3$*, according to the lexicographic order adopted in* Nqthm*) is chosen, and formula* $(\dagger)$*, or rather its instance*

$$0 < ms(a) \;\rightarrow\; ms(a) \times ms(b) \leq ms(a) \times (ms(a) \times ms(b))$$

is chosen as additional hypothesis. Its condition is reduced to $\top$*, and the linearised version of the inequality is conjoined to the transformed conjecture. After the elimination of the heaviest multiplicand, the resulting formula,*

$$ms(a)^3 + 2 \times ms(a) \times ms(b) - ms(a)^2 \leq 0$$

is not DAG-unsatisfiable either. Lemma* $(\dagger)$ *is used two more times to eliminate both* $ms(a)^3$ *and* $ms(a) \times ms(b)$*, thus generating*

$$2 \times ms(b) \leq 0$$

Unsatisfiability in DAG is found only after the last multiplicand,* $ms(b)$*, is eliminated by the application of lemma* $0 < ms(v)$*, which has been successively used in the previous steps to prove the condition of lemma* $(\dagger)$*. After its instantiation and linearisation, the final conjunction,*

$$2 \times ms(b) \leq 0 \wedge 1 - ms(b) \leq 0$$

is reduced to

$$2 \leq 0$$

which is DAG**-unsatisfiable.* □

The example shows that the introduction of additional hypotheses and the operations that take place thereafter bear a clear resemblance with conditional rewriting. Firstly, once a substitution for the variables of the additional hypothesis is found, it is then necessary to apply this substitution to the conditions of the hypothesis and to prove the resulting instance. Secondly, when t is the heaviest multiplicand of a conjunction of literals, assuming that the literal in which it occurs can be put in the form $t \leq u$, a suitable additional hypothesis has the form $\psi \supset u' \leq t$. The introduction of the consequent of this hypothesis amongst the other literals would then allow the use of *implication rewrite rules*[22],

$$\begin{array}{lcl} v_1 \leq v_2 \wedge v_2 \leq v_3 & \Rightarrow & v_1 \leq v_2 \wedge v_1 \leq v_3 \\ v_1 \leq v_2 \wedge v_2 \leq v_3 & \Rightarrow & v_1 \leq v_3 \end{array}$$

derived from the transitivity of $\leq$, where the second rule is chosen when the term to be removed occurs only in two literals[23]. For the multiplicand t, the suitable instance of the second rule is $u' \leq t \wedge t \leq u \Rightarrow u' \leq u$. This two-step operation, consisting of the introduction of a hypothesis and the application of a rewrite rule, can be collapsed into a single rewriting step, based on another implication rule,

$$\psi \quad \rightarrow \quad t \leq v \quad \Rightarrow \quad u' \leq v$$

where $\psi \supset u' \leq t$ is valid in the underlying theory. For instance, in example 3.2.5, the first required implication rule would be

$$0 < ms(a) \quad \rightarrow \quad ms(a)^4 \leq v \quad \Rightarrow \quad ms(a)^3 \leq v$$

---

[22] See appendix D.

[23] When the heaviest multiplicand is an existentially quantified variable, the above implication rules are replaced with an arithmetically valid equivalence,

$$(\exists v_2)(v_1 \leq v_2 \wedge v_2 \leq v_3 \wedge \cdots \wedge v_2 \leq v_n) \quad \equiv \quad v_1 \leq v_3 \wedge \cdots \wedge v_1 \leq v_n$$

The example also reveals that the immediate purpose of the introduction of additional hypotheses is the *removal* of chosen terms (the heaviest multiplicands) from a conjunction of literals. The removal of assigned subexpressions is a syntactic operation that occurs in other mechanisms related to the extension of decidable subclasses, particularly in the presence of rewrite rules, as described in chapter 4. Moreover, the previous remarks open the possibility of representation of the whole process present in the linear procedure of *Nqthm* as an instance of a more general rewriting process.

3.3 Conclusion

The mechanism of integration of decision procedures in *Nqthm* involves multiple modules, one of which controls the introduction of additional hypotheses. The strategy embedded in the system privileges the elimination of syntactically heavier terms, through the selection of adequate theorems. As the maximum syntactic weight diminishes when an elimination is successful, the procedure succeeds or fails finitely. The selection of additional hypotheses, apart from weight restrictions for the removed and the introduced terms (which may have been adopted mainly to guarantee termination), does not observe any other guideline, and has in principle to rely on exhaustive search in a database of theorems.

The complexity of the task of interfacing Hodes' algorithm to the simplifier suggests that the overall efficiency of the system may increase more significantly when the interface, rather than the algorithm, is improved. Given the possibility of simulation of this mechanism by means of (implication) rewrite rules, control structures for rewrite systems may contribute towards the strengthening of the selection strategies present in *Nqthm*. Search problems involving rewrite rules start to be addressed in the next chapter.

Chapter 4

Tactics and Proof Plans

Since the linear procedure created by Boyer and Moore is restricted to a subclass of the underlying language, additional mechanisms are needed whenever the decidable domain has to be enlarged. One of these mechanisms, based on a controlled version of the hypothesis introduction principle, turned out to be the most complex component of the extended linear procedure.

Alternative or complementary strategies for extending decidable subclasses include the use of rewrite rules, which also incur search problems. *Proof plans*, developed for the representation of proof structures, provide a suitable framework for the creation of rewriting control structures. Tactics and methods, upon which proof plans are defined, are discussed in section 4.1. The links between reduction classes and normalisation, and the representation of basic syntactic patterns present in normalisation are examined in section 4.2. Section 4.3 describes plans for normal forming.

4.1 Plans for Theorem Proving

Proof plans have been originally conceived by Bundy to guide inductive proofs. They are composed of *methods*, which in turn provide a partial specification for *tactics*. All three notions have as common purpose the description of the structure of proof classes[1].

[1] See [Bundy 88].

4.1.1 Tactics

LCF (*Logic for Computable Functions*) is an interactive theorem prover provided with a set of commands for the representation of complex inference steps[2]. It operates at two linguistic levels: axioms of formal theories are expressed at the object language, whereas the corresponding metatheory, including the inference rules, is represented in *ML*. Universal instantiation, for example, corresponds to the metatheoretical procedure

$$instantiate(\phi, v, t) = \phi'$$

where ϕ, v and t, respectively associated with the metalinguistic sorts *form*, *var* and *term*, assume as values names for formulae, variables and terms of the object language. When ϕ is instantiated to a formula of the form $(v)\psi[v]$, ϕ' is $\psi[^t/_v]$. The definition of the *rank* of a metalinguistic function or predicate symbol has to take into account not only the number of arguments, but also their sorts; the rank of *instantiate*, for instance, is (*form var term form*). An additional subsort, *thm*, contained in *form*, allows specific metavariables to range strictly over sets of theorems.

The use of metavariables and metaconstants eliminates any direct contact with the object language. It also makes redundant the complete description of object level tasks. For instance, once the required rewrite rules are introduced in the system, a LCF-proof for $(\forall X)(X = X \cup X)$ in Boolean algebra is obtained by exhaustive rewriting, without any additional instruction[3]. The representation of proof skeletons and *strategies* in LCF captures, to a certain extent, the structure of informal reasoning.

> "When one mathematician A asks another B 'what is your proof of X?' he often means 'how do you prove X?'; that is, a formal proof, step by step, will not satisfy him nearly so well as a recipe for proof. (...) The point is that such recipes — or strategies — appear to be built from combinations of smaller recipes (which we shall call tactics rather than strategies)." ([Gordon *et al* 77], p. 5)

[2] See [Milner 79], p. 3. There are already at least two different implementations of this system, the *Edinburgh LCF* and the *Cambridge LCF*, which later evolved into *HOL*, developed by Mike Gordon. In this section, no attention has been paid to possible dissimilarities between them.

[3] See [Milner 79], p. 7.

Tactics can represent at least two operations,

(i) the application of inference rules and

(ii) the application of rewrite rules, a special case of deduction where the rules are usually derived from equivalences.

An (*inference*) *tactic* is a (partial) function that associates a sequent of a language $\mathcal{L}$ with a pair $\langle \Lambda, Val \rangle$, where Λ is a finite list of sequents of the same language and *Val* is a *validation function* of arity (*thmlist thm*), such that, if $Val([\Gamma_1 \rightarrow \phi_1, \ldots, \Gamma_n \rightarrow \phi_n]) = (\Gamma_{n+1} \rightarrow \phi_{n+1})$, then

$$\frac{\Gamma_1 \rightarrow \phi_1, \ldots, \Gamma_n \rightarrow \phi_n}{\Gamma_{n+1} \rightarrow \phi_{n+1}}$$

Tactics generate proofs backwards, i.e. when applied to a conjecture or *goal*, a list of new sequents, Λ (called *subgoals*), is obtained. *Val* determines an object level derivation of the goal from Λ such that, when the subgoals are proved, a formal proof for the goal follows[4].

Since formulae are represented as sequents, the inference system in LCF is provided by the *sequent calculus*. Each *left* and *right* rule for connectives and quantifiers has an associated primitive inference tactic[5]. For instance, given the right rule for conjunction,

$$\frac{\Gamma \rightarrow \phi \qquad \Gamma \rightarrow \psi}{\Gamma \rightarrow \phi \wedge \psi}$$

the tactic *conj* consists of the subgoal list $[\Gamma \rightarrow \phi, \Gamma \rightarrow \psi]$ and the validation function $Val_\wedge$,

$$Val_\wedge\,(\Gamma \rightarrow \phi, \Gamma \rightarrow \psi) \;=\; (\Gamma \rightarrow \phi \wedge \psi)$$

When the goal is not a conjunction, the tactic fails. Apart from tactics for inference rules, there is a special primitive tactic, *idtac*, that generates as single subgoal the goal itself. Validation in this case is supplied by the identity function for sequents.

Rewriting tactics, on the other hand, are functions that take as input a goal and a list of theorems, returning as output either an empty list of sequents, in the event the goal

[4] See [Paulson 87], p. 209-10.

[5] The rules for Gentzen sequent systems are described, for instance, in [Gallier 87], p. 187-92.

is rewritten to $\top$, or a list containing the exhaustively rewritten goal under the rewrite set derived from the list of theorems, and a validation for the process. Validation is provided by the sequence of instances of rewrite rules successively applied to the goal, plus the list of subexpressions of the goal against which rules have been matched[6]. Rewriting tactics in LCF differ amongst themselves only in terms of their sets of theorems. Rewrite rules are classified into four groups, according to the features of the associated theorem, $(\Gamma \rightarrow \phi)$: if ϕ has the form

(a) $(t_1 = t_2)$, then $\Gamma \rightarrow t_1 \Rightarrow t_2$ and $\Gamma \rightarrow t_2 \Rightarrow t_1$ are term rewrite rules.

(b) $(\psi_1 \equiv \psi_2)$, then $\Gamma \rightarrow \psi_1 \Rightarrow \psi_2$ and $\Gamma \rightarrow \psi_2 \Rightarrow \psi_1$ are formula rewrite rules.

(c) $p(t_1, \ldots, t_n)$, where p does not denote equality, then $\Gamma \rightarrow p(t_1, \ldots, t_n) \Rightarrow \top$ is a formula rewrite rule.

(d) $\neg p(t_1, \ldots, t_n)$, then $\Gamma \rightarrow p(t_1, \ldots, t_n) \Rightarrow \bot$ is a formula rewrite rule.

When limited to the domain of primitive inference rules, tactic deductions are essentially identical to their object level images, since every primitive step is explicitly reproduced in the metalanguage[7]. The use of *tacticals*, functions of arity $(tactic^n \ \ tactic)$, $n \in \mathbb{N}$, for the generation of composite tactics overcomes this limitation. The five main examples are *then, thenl, try, orelse* and *repeat.*

i. tac_1 *then* tac_2 corresponds to the composition of its component tactics. When applied to a goal $\Gamma \rightarrow \phi$, $tac_1(\Gamma \rightarrow \phi)$ is first computed, followed by the application of tac_2 to every subgoal generated by tac_1. The subgoal list for tac_1 *then* tac_2 is the concatenation of the lists of subgoals generated from the subgoals of $tac_1(\Gamma \rightarrow \phi)$, and its validation is the composition of the validation functions for tac_1 and tac_2.

ii. *tac thenl* $[tac_1, \ldots, tac_n]$, is similar to *then*, except that a specific tactic tac_i is applied to each subgoal $\Gamma_i \rightarrow \psi_i$ generated by *tac*. It fails if the number of such subgoals is different from n.

iii. *try tac* extends *tac* to a total function in the domain of sequents of the underlying language. For those goals where *tac* succeeds, *try tac* $(\Gamma \rightarrow \phi) = tac\ (\Gamma \rightarrow \phi)$.

[6] See [Paulson 87], p. 248-9.

[7] Primitive rewrite tactics, on the other hand, may apply several rules in a single step.

When *tac* fails, it generates the input sequent as sole subgoal.

iv. tac_1 *orelse* tac_2 partially extends tac_1 w.r.t. tac_2. It either fails, or successfully applies one and only one of the tactics tac_1 or tac_2 to the goal, where tac_1 has precedence over tac_2. The output contains the subgoal list and the validation function of the successfully applied tactic.

v. *repeat tac* exhaustively applies *tac* to the current goal and subsequent subgoals. For each subgoal, the process is iterated unless it is already the empty list of subgoals, or *tac* is not applicable. The subgoals for which it fails are part of the output. It can be defined as

$$(\mathit{repeat}\ \mathit{tac})(\Gamma \rightarrow \phi) \quad = \quad ((\mathit{tac}\ \mathit{then}\ \mathit{repeat}\ \mathit{tac})\ \mathit{orelse}\ \mathit{idtac})\ (\Gamma \rightarrow \phi)$$

Composite tactics, i.e. those defined by the application of tacticals to other tactics, can be seen as families of derived inference rules. For instance, *repeat conj* represents the infinite family

$$\frac{\Gamma \rightarrow \phi_1 \quad \Gamma \rightarrow \phi_2}{\Gamma \rightarrow \phi_1 \wedge \phi_2} \quad \frac{\Gamma \rightarrow \phi_1 \quad \Gamma \rightarrow \phi_2 \quad \Gamma \rightarrow \phi_3}{\Gamma \rightarrow \phi_1 \wedge \phi_2 \wedge \phi_3} \quad \cdots \quad \frac{\Gamma \rightarrow \phi_1 \quad \ldots \quad \Gamma \rightarrow \phi_n}{\Gamma \rightarrow \phi_1 \wedge \cdots \wedge \phi_n} \quad \cdots$$

where $n \in \mathbb{N}$. The introduction of tacticals allows the construction of *complete tactics* to represent proof strategies that generate the empty list of sequents for a particular goal.

4.1.2 Methods

The construction of complete proof strategies requires the organisation of several tactics into a tree-structured object, where each derived subgoal is transformed until the empty list of sequents is achieved. The scope of application of each strategy, however, is usually limited to a subclass of sequents: a complete tactic for putting expressions into disjunctive normal form in Boolean algebras, for instance, is unsuitable for other normalisation tasks. To form a more general automated theorem prover, a collection of strategies is required .

There are two solutions for the representation of this collection: either the strategies are explicitly given, or else a mechanism for their generation in the presence of a particular goal has to be built. The construction of strategies involves the selection of suitable tactics, and can therefore be simplified once information about the behaviour of each tactic is taken into account. Properties of the output for a given input could be obviously obtained from the direct application of a tactic to a conjecture, but the amount of information generated in most cases exceeds what is essential for devising a strategy.

An alternative source of information about a tactic is its *specification*, which describes its behaviour in terms of properties of the domain and image sets, ignoring the elementary processes that take place between input and output. Specifications consist of four elements. The *header* identifies the name of the function and describes syntactic properties of its arguments and the output, such as their sorts and arity. Names of parameters given in the header are shared with all the three remaining components. Semantic properties of the domain are described under the form of *preconditions*, which become superfluous whenever the domain is equal to the underlying universe (i.e. the specification for a tactic that is applicable to any formula has an empty list of preconditions). The *output* names the parameters of the header that are modified by the function. *Postconditions* (or *effects*), on the other hand, describe semantic properties that apply to the parameters listed in the *output*, whenever the input satisfies the preconditions. Assuming that ϕ' is the expression computed by a tactic for an input formula ϕ, the effects can be represented as a formula $\boldsymbol{F}[\phi']$ of a specification language. In those cases where $\boldsymbol{F}$ establishes a functional dependency between ϕ and ϕ', it provides an alternative procedure for computing the output of a tactic. If the dependency is not functional, $\boldsymbol{F}$ determines a class,

$$\{\psi \in Fml_{\mathcal{L}} \mid \boldsymbol{F}[\psi]\}$$

to which ϕ' belongs[8].

Methods provide a partial specification for tactics. Their representation resorts to the predicate *method/6*,

[8] See [Liskov & Guttag 86], p. 7-8, 42-5 and [Liskov & Berzins 86], p. 3-4.

method (*name* (*...Args...*),
Input formula,
Preconditions,
Effects,
Output formulae,
tactic(*...Args...*)
).

The components of a method are essentially those present in a specification. A method and its related tactic usually share the same *name*. The *input formula* represents the goal, and the *preconditions* determine the patterns a goal has to satisfy to allow the application of the tactic. The *effects* and the *output formulae* characterise the structure of the results generated from the input. The final element of a method is an implementation for the specified tactic, under the form *tactic(...Args...)*. Methods also inherit the same classification established for tactics, thus belonging either to the *inference* or the *rewriting* group[9].

As the specification is partial, the description of properties may be incomplete (or possibly incorrect) either for the *input* or the *output formulae*. If a tactic is applicable to a goal, the derived subgoals match the output pattern and satisfy the *effects* of the corresponding method. The converse, however, is not necessarily true: when the tactic is a partial function, it may fail for a goal, no matter if the corresponding method succeeds. This disadvantage is a consequence of the choice of working with partial specifications[10]. To overcome it, it would be necessary to employ the tactic itself, or its reproduction in the metatheory.

> "One could argue that the preconditions of methods should be strengthened so that they implied the success of the tactic. However, note that, in practice, this would amount to running the tactic 'unofficially' in the precondition, to see if it succeeded, before running it 'officially'." ([Bundy 88], p. 16)

[9] Methods are defined in [Bundy *et al* 91]. The distinction between *inference* and *rewriting* methods is actually not present in *Clam*, a proof planning system described in [Van Harmelen & *et al* 93].

[10] See [Bundy 88], p. 7.

The limitations associated with partial representations, nonetheless, are balanced in most cases by the fact that search is pruned.

4.1.3 Proof Plans

Planners are systems designed for the mechanical creation of proof strategies from the composition of methods. A mechanically generated strategy corresponds to a *proof plan*. Methods may be implemented in *Clam*, a proof planning system interfaceable to interactive theorem provers, including *Oyster*, which is based on Martin Löf's intuitionistic type theory[11].

Planners have to solve a search problem: given a goal (a conjecture to be proven, a formula or term to be rewritten), the system has to build a tree of methods, corresponding to a tree of tactics, that, when applicable, reduces the goal to the empty list of subgoals. At each stage, there may be choices to be made, since more than one tactic may be applicable to intermediate subgoals. Since the unguided construction of the planning tree, until a suitable plan is found, is undesirably inefficient in the average case, planning techniques are necessary. Four search approaches are available in *Clam*, each of which features a distinct planner: *depth-first*, *breadth-first*, *iterative deepening* and the heuristically guided *best-first*.

> " Each planner takes the theorem to be proved as the initial state and finds a tree of methods which will transform it into a list of *trues*. At each cycle it finds a method that is applicable to the current state by matching that state to the input pattern of the method and checking the preconditions. The list of output formulae is then calculated from the output and the effects of the method. The cycle is repeated for each of these output formulae." ([Bundy *et al* 91], p. 7)

In the LCF terminology, instead of a list of *trues*, a planner has as target the generation of an *empty* list of subgoals.

[11] See [Van Harmelen & *et al* 93].

4.2 Rewrite Systems

The role of proof plans in theorem proving is not limited to inferences, since they can control rewriting processes in general, particularly those related to expression normalisation. The identification of special methods for normal forming is based on the analysis of the syntactic features of classes of transformations that make up normalisation.

Once adequate methods have been identified, two main types of plans can be implemented: *special-purpose plans*, which are devised to represent specific normalisation tasks, and *general-purpose plans*, which correspond to families of special-purpose plans that share certain properties.

4.2.1 Expression Normalisation

From an algebraic point of view, the concept of *normal form* involves an equivalence relation defined on a universe set and a subset of the universe that exhibits a pre-established (in general syntactic) property[12].

Definition 4.2.1 (Normal Forms)

Let $\mathcal{A}$ and $\mathcal{B}$ be subsets of a universe set $\mathcal{U}$, and let E be an equivalence relation in $\mathcal{U}$.

i. Given $a \in \mathcal{A}$, an element $b \in \mathcal{B}$ is a normal form *of a (with respect to E and $\mathcal{B}$) iff $E(a, b)$.*

ii. A normaliser *for $\mathcal{A}$ (with respect to E and $\mathcal{B}$) is a function $\mathcal{N}$: $\mathcal{A} \rightarrow \mathcal{B}$ such that, for all $a \in \mathcal{A}$, $E(a, \mathcal{N}(a))$. $\mathcal{N}(\mathcal{A})$ is the set of normalised expressions of $\mathcal{A}$ w.r.t. $\mathcal{N}$. $\mathcal{N}$ is an* effective normaliser *iff $\mathcal{N}$ is effectively computable.*

Hence, the normal form of an element of $\mathcal{A}$ with respect to E and $\mathcal{B}$ is not necessarily unique. When there is more than one normal form for at least one element of $\mathcal{A}$,

[12] See [Bundy 91], p. 2.

more than one normaliser can be exhibited. This definition seems to capture all the distinct uses of the term in mathematical logic, at least for first-order theories. In the case of prenex normal forming in a language $\mathcal{L}$, for example, the universe, as well as the subset $\mathcal{A}$ of formulae to be normalised, corresponds to $Fml_{\mathcal{L}}$. The equivalence relation is logical equivalence ($\equiv$), while the set $\mathcal{B}$ of normalised expressions is the class of prenex normal form formulae. Also, the identification of reduction classes for a decision problem, as described in definition 2.1.1, is a special case of normalisation.

Lemma 4.2.1 *Let $\mathcal{A}$ and $\mathcal{B}$ be subsets of $\mathcal{U}$, and let $\mathcal{B}'$ be a reduction class for $\mathcal{B}$ w.r.t. $\mathcal{A}$ in $\mathcal{U}$. Let $\mathcal{N}\colon \mathcal{B} \to \mathcal{B}'$ be a reduction function for $\mathcal{B}$ w.r.t. $\mathcal{A}$. Then $\mathcal{N}$ is a normaliser for $\mathcal{B}$ w.r.t. $\mathcal{B}'$.*

PROOF. Let $E(b, b')$ be defined as

$$b \in \mathcal{A} \quad \text{iff} \quad b' \in \mathcal{A}$$

Then E is an equivalence relation in $\mathcal{U}$, since it is reflexive, symmetric and transitive. According to the definition of reduction classes, for every $b \in \mathcal{B}$, $b \in \mathcal{A}$ iff $\mathcal{N}(b) \in \mathcal{A}$, i.e. $E(b, \mathcal{N}(b))$. From this result and definition 4.2.1, it follows that $\mathcal{N}\colon \mathcal{B} \to \mathcal{B}'$ is a normaliser for $\mathcal{B}$ with respect to E and $\mathcal{B}'$. ∎

Definition 4.2.1 can be specialised to the context of reduction problems once the universe set is replaced with the set of formulae of a language.

Definition 4.2.2 (Γ-normaliser)

Let Γ be a subset of $Fml_{\mathcal{L}}$, and let $\ulcorner\Sigma_1\urcorner$ and $\ulcorner\Sigma_2\urcorner$ be recursive subclasses of $Fml_{\mathcal{L}}$.

i. A Γ-normaliser *for Σ_1 w.r.t. Σ_2 is a function $\mathcal{N}_\Gamma\colon \Sigma_1 \to \Sigma_2$ such that*

$$\Gamma \models \mathcal{N}_\Gamma(\phi) \quad \textit{iff} \quad \Gamma \models \phi$$

ii. $\mathcal{N}_\Gamma(\Sigma_1)$ is the Γ-normalised subclass *generated by f_Γ from Σ_1.*

According to the definition of reduction classes, when $\Sigma_2 \subseteq \Sigma_1$, a Γ-normaliser for Σ_1 w.r.t. Σ_2 is also a reduction function for Σ_1 with respect to the theory that has Γ as axiom set.

As discussed in section 2.2.1, certain normalisers may be implemented as noetherian and confluent rewrite sets. Even when a rewrite set $\mathcal{R}$ does not possess such properties, it is still possible to generate effective mechanisms through the introduction of control structures. Given two sets of formulae, Σ_1 and Σ_2, $\mathcal{R}$ determines a relation r in $\Sigma_1 \times \Sigma_2$, defined as

$$r(\phi, \phi') \quad \text{iff} \quad \phi \in \Sigma_1 \;\&\; \phi' \in \Sigma_2 \;\&\; \phi \overset{\mathcal{R}}{\Rightarrow} \phi'$$

whose computation requires the construction of every rewriting sequence for each element of Σ_1. Any subset of r extensionally represents a subrelation for r. *Rewrite systems* (or controlled rewrite sets) provide an alternative representation for such subrelations. Control in this context amounts to a mechanism for the selection of rewrite rules and subexpressions of a conjecture ϕ, such that the set of rewritten versions of ϕ becomes ordered.

An example of a rewrite system is based on (deterministic) Markov algorithms: a computable function is generated from a rewrite set once its rules are linearly ordered, and rewriting proceeds e.g. from left to right of the input expression. The resulting function is effective whenever the rewriting sequence so generated is finite[13]. Alternative control structures are nonetheless required to explore the full potential of rewrite-based computability.

Definition 4.2.3 (Rewrite systems)

Let $\mathcal{R}$ be a rewrite set and $\mathcal{E}$ be a set of expressions such that $\epsilon \in \mathcal{E}$.

i. A rewriting tree *for ϵ and $\mathcal{R}$ is a tree of expressions such that each path in the tree that starts with ϵ is a rewrite sequence for ϵ and $\mathcal{R}$. A rewriting tree is* complete *iff each path of the tree is either infinite or ends in a normal form expression (under $\mathcal{R}$).*

[13] See [Dershowitz & Jouannaud 90], p. 245, and [Sommerhalder & Van Westrhenen 88], p. 265-7.

ii. The triple $\langle \mathcal{E}, \mathcal{R}, CF \rangle$ *is a* rewrite system *iff CF is a function defined in* $\mathcal{E}$ *with the set of all finite lists of* $\mathcal{R}$*-rewriting sequences as range, such that, for all* $\epsilon \in \mathcal{E}$,

$$CF(\epsilon) = \langle\langle \epsilon, \epsilon_{1,1}, \ldots, \epsilon_{1,m_1}, \ldots \rangle, \ldots, \langle \epsilon, \epsilon_{n,1}, \ldots, \epsilon_{n,m_n}, \ldots \rangle\rangle$$

CF corresponds to the control function *of the rewrite system.*

CF determines an order for the generation of the search space (or rather of a fragment of such space), which is implicit in the order of the rewriting sequences associated with every element of $\mathcal{E}$. A subrelation $r' \subseteq r$, where r is the relation defined by a rewrite set $\mathcal{R}$ and the classes Σ_1 and Σ_2, can be characterised by reference to a rewrite system $\langle \mathcal{E}, \mathcal{R}, CF \rangle$ as follows

$$r'(\phi, \phi') \quad \text{iff} \quad \phi \in \Sigma_1 \;\&\; \phi' \in \Sigma_2 \;\&\; (\phi \overset{\mathcal{R}}{\Rightarrow} \phi') \in CF(\phi)$$

When *CF* associates each element of Σ_1 with at most one rewriting sequence, and if this sequence is finite, the rewrite system defines a *normaliser* $\mathcal{N}$: $dom(CF) \rightarrow \Sigma_2$, where $dom(CF) \subseteq \Sigma_1$ is the domain of *CF*, and $\mathcal{N}(\phi) = \phi'$ iff $r'(\phi, \phi')$. *CF* then is the key element in the construction of a normaliser $\mathcal{N}$ based on a set of rewrite rules $\mathcal{R}$, and the effectiveness of $\mathcal{N}$ depends on the recursiveness of *CF*.

Besides an effective rewriter, the computational effectiveness of normalisation also requires a recursive domain. Certain recursive sets of expressions can be represented as *syntactically defined* classes, for which the membership relation is defined strictly in syntactic terms, i.e. an object belongs to a syntactically defined class iff it satisfy certain syntactic conditions. The definition below tries to capture this informal concept.

Definition 4.2.4 (Syntactically Defined Classes)

Let $\widehat{\mathcal{L}} = \langle \ell, \mathcal{V}, \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$ *be an effectivised first-order language, and* Σ *be a subclass of* $Exp_{\widehat{\mathcal{L}}}$.

i. Σ *is* syntactically defined *iff one of the following conditions hold.*

(a) $\ulcorner \Sigma \urcorner$ *is a recursive subset of* $\ulcorner \mathcal{V} \urcorner$.

(b) $\ulcorner\Sigma\urcorner$ *is a recursive subset of* $\ulcorner\mathcal{C}\urcorner$.

(c) $\Delta \subset \Sigma$ (base set), *and, for every* $\epsilon_i \in \Delta_i$ *and* $\boldsymbol{\epsilon} \subset \Sigma$, $S_k(\epsilon_1, \ldots, \epsilon_{m_k}, \boldsymbol{\epsilon}) \in \Sigma$, $1 \leq k \leq n$, *where* (1) S_k *is a non-logical symbol of* $\widehat{\mathcal{L}}$, (2) $\Delta, \Delta_1, \ldots, \Delta_{m_k}$ *are syntactically defined classes, and* (3) $S_k(\epsilon_1, \ldots, \epsilon_{m_k}, \boldsymbol{\epsilon})$ *is a well-formed expression of* $\widehat{\mathcal{L}}$.

(d) Σ *has the form* $\Sigma_1 \cup \Sigma_2$, $\Sigma_1 \cap \Sigma_2$ *or* $\Sigma_1 - \Sigma_2$, *where* Σ_1 *and* Σ_2 *are syntactically defined.*

ii. Σ *is* syntactically definable *iff there is a syntactically defined set* Σ' *such that* $\Sigma = \Sigma'$.

The sets of terms, formulae and expressions of a first-order language are only a few examples of syntactically definable classes.

Example 4.2.1 *Let* $\mathcal{L}$ *be a first-order language.*

i. The set of universal formulae of $\mathcal{L}$ *can be syntactically defined by the productions*

$$\begin{aligned} unv &:= qff \mid (var)unv \\ qff &:= atm \mid \neg qff \mid qff \wedge qff \mid qff \vee qff \mid qff \supset qff \mid qff \equiv qff \end{aligned}$$

ii. The set of formulae of $\mathcal{L}$ *whose prenex form are universal can be syntactically defined as*

$$\begin{aligned} unv' &:= atm \mid (var)unv' \mid \neg exs' \mid unv' \wedge unv' \mid unv' \vee unv' \mid exs' \supset unv' \mid qff \equiv qff \\ exs' &:= atm \mid (\exists var)exs' \mid \neg unv' \mid exs' \wedge exs' \mid exs' \vee exs' \mid unv' \supset exs' \mid qff \equiv qff \end{aligned}$$

where qff denotes a quantifier-free formula of $\mathcal{L}$.

iii. The class of terms $\mathcal{E}$ *recursively defined as*

(a) *Any atomic term of* $\mathcal{L}$ *is an element of* $\mathcal{E}$ *(base set)*

(b) *If* $t \in \mathcal{E}$ *and* $S \in Sym_{\mathcal{L}}$, *then* $S(t, \ldots, t) \in \mathcal{E}$, *provided that* $S(t, \ldots, t)$ *is a well-formed term of* $\mathcal{L}$

(c) *Only the terms defined above are elements of* $\mathcal{E}$

is syntactically defined, according to item i(c) of definition 4.2.4. □

All syntactically defined classes are recursive. Some of them can be generated by context-free grammars, as shown by in the above examples, whereas others cannot.

Lemma 4.2.2 *There is a syntactically defined class of expressions of a language $\mathcal{L}$ that cannot be generated by a context-free grammar.*

PROOF. Context-free generated classes are not closed under intersection or complementation. As a result, there are sets of expressions Γ_1 and Γ_2 generated by context-free grammars such that $\Gamma_1 \cap \Gamma_2$ is not context-free. However, assuming that every context-free generated class is syntactically definable, then according to definition 4.2.4, $\Gamma_1 \cap \Gamma_2$ is syntactically defined[14]. ∎

Example 4.2.1 iii involves a class that apparently cannot be generated by a context-free grammar, as argued in appendix E. To describe this and all other syntactically defined classes, a special many-sorted syntactic metalanguage, containing predicates such as *occ_sym*, *pos_occ_sym* and *lnk_occ_sym*,

$occ\_sym(S, \epsilon)$	iff	S occur in expression ϵ
$pos\_occ\_sym(S, \epsilon, p)$	iff	S occur in expression ϵ at position p
$lnk\_occ\_sym(S_1, S_2, \epsilon)$	iff	S_2 has an occurrence in the immediate scope of S_1 in expression ϵ

has been devised. In this language, a *description* for a procedure that links an initial class Σ_1 to a final class Σ_2 can be formulated in terms of syntactic features of both classes[15].

Three aspects of the current study on decidable class extension are affected by the above results. First, the reduction of classes is entirely addressable from the point of view of normalisation processes. Also, effective procedures for the reduction of recursive classes into proper subclasses can be expressed by means of rewrite systems

[14] See [Lewis & Papadimitriou 81], p. 126

[15] A *description*, therefore, is not the same as a *specification* for a program, since the former lacks the syntactic components of the latter, such as the names of input and output variables, or the arity of function and predicate symbols. The above distinction does not match that established, for instance, in [Lukey 80], where a description presents what a program does, and a specification determines what it should do. The restriction of descriptions to the semantic aspects of a specification has been proposed by R. C. Sanchez (private conversation).

once suitable control functions are identified. Finally, syntactically definable sets, some of which are generated by context-free grammars, form suitable normalisation domains, since they are recursive.

4.2.2 Normalisation Patterns

A class of effective normalisers can be built by means of *normalisation tactics*, parameterised functions of arity $\langle form\ formlst\ form \rangle$ that result from the inclusion of special control functions in a rewrite set[16]. Primitive normalisation tactics capture the structure of normal forming operations present in theories such as the predicate calculus, Abelian groups and Presburger arithmetic. One group of primitive operations concerns the *elimination* of occurrences of symbols, by their complete *removal* or by a *reduction* in their number. Another group involves the *reorganisation* of occurrences of symbols, and includes the *stratification* of occurrences of a symbol over others and the *reordering* of occurrences of a single symbol.

Given a syntactically defined set of formulae and a subclass of normalised expressions, it is in some cases possible to provide a syntactic description of the subclass w.r.t. the initial set in terms of *absent symbols*, *forbidden linked occurrences of symbols*, *forbidden relative positions for occurrences of symbols* and similar properties. Properties of both initial and normalised classes provide a *description* for the tactic that links them[17]. Each of these operations is a normalisation process in its own right. They can be schematically described by means of

(i) a pair of sets, one of which is a universe set of expressions, while the other is the subset of normalised expressions, whose syntactic form varies according to the specific transformation performed, and

(ii) a set of rewrite rules that are adequate to perform the transformation.

The construction of the universe set requires a single production,

$$exp \quad := \quad bas|S_1(\boldsymbol{exp})|\ldots|S_n(\boldsymbol{exp})$$

[16] As discussed in appendix D.7, the arity of many-sorted function and predicate symbols consists of an ordered list of sorts. *Form* and *formlst* are two metalinguistic sorts, respectively made up of object level formulae and finite lists of such formulae.

[17] Primitive operations present in normalisation are described in [Bundy 91], p. 2-8.

where $\boldsymbol{exp}$ is a finite sequence of the form $exp, \ldots, exp$. The *removal* of occurrences of a symbol S_k demands a normaliser that associates an expression of the universe with an element of its subclass

$$exp' \quad := \quad bas|S_1(\boldsymbol{exp'})|\ldots|S_{k-1}(\boldsymbol{exp'})|S_{k+1}(\boldsymbol{exp'})|\ldots|S_n(\boldsymbol{exp'})$$

by applications of rewrite rules such as

$$S_k(v_1, \ldots, v_{m_k}) \quad \Rightarrow \quad \delta\{v_1, \ldots, v_{m_k}\}$$

provided that $\delta\{v_1, \ldots, v_{m_k}\}$ is an expression that does not contain S_k.

Example 4.2.2 *If the initial set of expressions is generated by*

$$prp := var|\neg prp|prp \vee prp|prp \wedge prp|prp \supset prp|prp \equiv prp,$$

the exhaustive application of

$$\phi_1 \supset \phi_2 \quad \Rightarrow \quad \neg\phi_1 \vee \phi_2$$

removes $\supset$ *and creates a new class,*

$$prp' := var|\neg prp'|prp' \vee prp'|prp' \wedge prp'|prp' \equiv prp',$$

which is a strict subclass of the universe. With respect to the normalised class, $\supset$ *is a* deviant symbol. □

There may also be rules for the *reduction* of the number of occurrences of S_k, whose repeated application does not lead to the complete elimination of S_k. *Reduce* rules have the form

$$\epsilon[\![S_k(\boldsymbol{t_1}), \ldots, S_k(\boldsymbol{t_n})]\!] \quad \Rightarrow \quad \epsilon'[\![S_k(\boldsymbol{t'_1}), \ldots, S_k(\boldsymbol{t'_{n-m}})]\!]$$

where $\boldsymbol{t_i}$ and $\boldsymbol{t'_i}$ are finite lists of terms free from occurrences of S_k, and $m \neq 0$. The rules

$$(\phi_1 \wedge \phi_2) \vee (\phi_1 \wedge \phi_3) \Rightarrow \phi_1 \wedge (\phi_2 \vee \phi_3)$$
$$(v_1 \times v_2) + (v_1 \times v_3) \Rightarrow v_1 \times (v_2 + v_3)$$

respectively cause the reduction of occurrences of conjunction and multiplication, and the description of the final classes, in each case, would have to acknowledge the residual presence of either $\wedge$ or $\times$.

For the *stratification* of symbols, if the occurrences of a function symbol S_k have to be moved below occurrences of other symbols of the universe set, the final subclass

$$exp' := bas' | \ldots | S_i(\boldsymbol{exp'}) | \ldots, i \neq k, 1 \leq i \leq n$$
$$bas' := bas | S_k(\boldsymbol{bas'}),$$

would be generated by a set of rewrite rules of the form

$$S_k(v_1, \ldots, v_{i-1}, S_j(\boldsymbol{u}_j), v_i, \ldots, v_{m_k}) \Rightarrow \delta\{v_1, \ldots, v_{m_k}, \boldsymbol{u}_j\}$$

for each $j, 1 \leq j \leq n, j \neq k$ and $i, 1 \leq i \leq m_k$, such that $\delta\{v_1, \ldots, v_{m_k}, \boldsymbol{u}_j\}$ belongs to the normalised subclass.

Example 4.2.3 *If*

$$prp := lit | prp \vee prp | prp \wedge prp$$

is the initial set of expressions, the exhaustive application of rules

$$\phi_1 \vee (\phi_2 \wedge \phi_3) \Rightarrow (\phi_1 \vee \phi_2) \wedge (\phi_1 \vee \phi_3)$$
$$(\phi_1 \wedge \phi_2) \vee \phi_3 \Rightarrow (\phi_1 \vee \phi_3) \wedge (\phi_2 \vee \phi_3)$$

reduces it to

$$prp' := cls | prp' \wedge prp'$$
$$cls := lit | cls \vee cls$$

With respect to the normalised class, occurrences of conjunctions in the immediate scope of disjunctions represent deviantly linked occurrences. □

Concerning the *reordering* of symbols, there are classes of expressions where one or more arguments of a symbol S_k range over specific subclasses of expressions that do not contain a symbol S_j. An example of an *ordered subclass* is

$$exp' \quad := \quad bas|S_1(exp'_1,\ldots,exp'_{m_1})|\ldots|S_k(exp'_1,\ldots,exp'_{m_k-1},bas)|\ldots|S_n(exp'_1,\ldots,exp'_{m_n})$$

whose generation may require rules of the form

$$S_k(u,S_k(v,w)) \quad \Rightarrow \quad S_k(S_k(u,v),w)$$

Example 4.2.4 *If the associativity of* $+$,

$$v_1+(v_2+v_3) \quad \Rightarrow \quad (v_1+v_2)+v_3$$

is adopted as a rewrite rule, it transforms the class

$$trm := var|(trm+trm)$$

into

$$trm' := var|(trm'+var)$$

With respect to the final class, summands with composite right-hand side arguments, such as $x+(y+z)$, *are* deviantly ordered expressions. □

The rules required to perform each of the basic operations may be classified according to their form. There are at least three different types of rules for the removal of symbols, including *total* rules, applicable to any context of occurrence of the related symbol, and also *partial* rules, which cover particular contexts only.

Definition 4.2.5 (Remove rules)

Let T *be a theory in* $\mathcal{L}$, S *be a symbol of* $\mathcal{L}$, *and* Σ *be a subset of* $Exp_{\mathcal{L}}$, *such that* S *does not occur in* Σ. *Let* $\rightleftharpoons$ *represent either* $=$ *or* $\equiv$.

i. If δ *is an element of* Σ, *then*

(a) *A* (conditional) total *remove* rule *for* S *in* T *w.r.t.* Σ *has the form*

$$\Gamma \rightarrow S(v_1, \ldots, v_n) \Rightarrow \delta$$

where $\Gamma \rightarrow S(v_1, \ldots, v_n) \rightleftharpoons \delta$ *is* T*-valid.*

(b) *A* (conditional) partial dominant *remove* rule *for* S *in* T *w.r.t.* Σ *has the form*

$$\Gamma \rightarrow S(\epsilon_1, \ldots, \epsilon_n) \Rightarrow \delta$$

where $\Gamma \rightarrow S(\epsilon_1, \ldots, \epsilon_n) \rightleftharpoons \delta$ *is* T*-valid, and either there is a symbol* S' *that occurs in* ϵ_j*, for some* $j, 1 \leq j \leq n$ *(i.e. symbols other than variables occur in the scope of* S*), or if all the terms in the scope of* S *are variables, then* $\epsilon_i \stackrel{s}{=} \epsilon_j$*, for some* $i \neq j$*.*

(c) *A* (conditional) partial non-dominant *remove* rule *for* S *in* T *w.r.t.* Σ *has the form*

$$\Gamma \rightarrow \epsilon[\![S]\!] \Rightarrow \delta$$

where ϵ *is not dominated by* S *and* $\Gamma \rightarrow \epsilon[\![S]\!] \rightleftharpoons \delta$ *is* T*-valid.*

ii. *A set of* remove *rules for* S *w.r.t.* Σ *is* complete for $\mathcal{L}$ *iff it either contains a total* remove *rule for* S*, or has partial (dominant or non-dominant) rules,*

$$\begin{array}{lcl} \delta_{1,1} & \Rightarrow & \delta_{1,2} \\ & \vdots & \\ \delta_{n,1} & \Rightarrow & \delta_{n,2} \end{array}$$

such that, for every formula $\phi \in Fml_{\mathcal{L}}$ *in which* S *occurs, and for each occurrence of* S *in* ϕ*, there is a subexpression of* ϕ *containing this occurrence which is an instance of* $\delta_{i,1}$*, for some* $i, 1 \leq i \leq n$*.*

Partial non-dominant rules for S can be classified as *first, ..., n-th degree rules*, according to the position of occurrence of S in the left-hand side expression of the rule. Also, a single rule may act as partial *remove* rule for more than one symbol.

Example 4.2.5

i. Rule

$$\phi \equiv \psi \quad \Rightarrow \quad (\phi \supset \psi) \wedge (\psi \supset \phi)$$

is total *for biconditionals, since* $\equiv$ *is the dominant symbol of the lhs (left hand side) expression, and the only expressions that occur in its scope are distinct variables (over formulae).*

ii. Concerning the elimination of existential quantifiers,

$$(\exists v)(v = t) \quad \Rightarrow \quad \top$$

is a partial dominant *rule, since there are symbols other than variables in the scope of the quantifier. For the removal of negations, rule*

$$\neg(\neg\phi) \quad \Rightarrow \quad \phi$$

is also partial dominant, *due to the occurrence of the very symbol to be removed as dominated symbol as well. Finally, rules*

$$\begin{array}{lll} v + v & \Rightarrow & double(v) \\ v \times v & \Rightarrow & v^2 \end{array}$$

share the same category with the above rules due to the multiple occurrences of a single variable in the scope of the symbol being removed.

iii. The rule

$$v_1 + v_2 = 0 \quad \Rightarrow \quad v_1 = 0 \wedge v_2 = 0$$

provides the partial removal of $+$. *Given that* $+$ *occurs in the immediate scope of the dominant symbol,* $=$, *it is a* first degree rule. *On the other hand, the conditional rule*

$$v_3 \neq 0 \quad \rightarrow \quad (v_1 + v_2) \times v_3 = v_3 \quad \Rightarrow \quad (v_1 = 0 \wedge v_2 = 1) \vee (v_1 = 1 \wedge v_2 = 0)$$

is a second degree rule, *since* $+$ *occurs in the immediate scope of* $\times$, *which in turn is in the immediate scope of the dominant symbol,* $=$. *The second rule is also a partial non-dominant* remove *rule of first degree for multiplication.*

iv. Given the class of variable-free formulae defined as

$$\begin{array}{rcl} vfm & := & conj | vfm \wedge vfm \\ conj & := & atm | conj \vee atm \\ atm & := & 0 = 0 | 0 < 0, \end{array}$$

the rules

$$\begin{array}{rcl} \phi \vee (0 = 0) & \Rightarrow & 0 = 0 \\ \phi \vee (0 < 0) & \Rightarrow & \phi \end{array}$$

form a complete set of partial remove rules *for disjunction (with respect to the initial class), as they cover all possible contexts of occurrence of this symbol.* □

Stratify and *reorder* rules can also be classified into *complete*, *partial dominant* and *partial non-dominant.* For instance,

$$(\exists v)(\phi[\not v] \wedge \psi) \Rightarrow \phi[\not v] \wedge (\exists v)\psi$$

is a partial dominant *stratify* rule for $\exists$ over $\wedge$. Complete sets of *stratify* and *reorder* rules are defined in a similar way. Since *remove* is the only group to be examined in later applications, the formal definitions for the other two groups are not relevant in the present context. Termination measures for *remove* rules are discussed in chapter 9. Primitive normalisation tactics are then definable in terms of the type of rule required to perform the corresponding transformation.

Definition 4.2.6 (Primitive Normalisation Tactic)

A primitive normalisation tactic *is a parameterised function associated with a rewrite system* $\langle \mathcal{E}, \mathcal{R}, CF \rangle$, *where the control function CF is defined by the restriction of* $\mathcal{R}$ *to its subset of* remove, reduce, stratify, reorder *or any other particular type of rules, which are applied according to the order guidelines established in a (deterministic) Markov algorithm.*

Remove$(\mathcal{R}, S)$, for instance, is defined as

$$remove(\mathcal{R}, S)(\phi) = \phi' \quad \text{iff} \quad \mathcal{R}' \text{ is the subset of } \mathcal{R} \text{ formed by } remove \text{ rules for } S, \text{ and } \phi \overset{\mathcal{R}'}{\Rightarrow} \phi'$$

The corresponding control function depends on the *argument* (the input formula) and the *parameters* of the tactic, which, in the case of *remove*, are the rewrite set and (the name of) a deviant symbol. *CF* is responsible for the selection of the elements of $\mathcal{R}$ that are *remove* rules for S, and for their ordered application thereafter. Similar definitions apply to each of the other tactics. The rewriting sequence computed by a normalisation tactic provides its *validation* component, already defined for inference tactics, and reveals intensional features of the tactic, since it describes the process of construction of the value of the function for a particular input. When the subset of rules selected by a normalisation tactic is noetherian, the corresponding function is effective.

The restrictions embedded in a normalisation tactic can be simulated in a LCF rewriting tactic once the set of rules is delimited according to the syntactic task to be performed: for instance, when only *remove* rules are available, a rewriting tactic assumes the behaviour of *remove*. In the proof planning approach, however, there is a single rule base for all normalisation tactics, and each of them selects those suitable for the task at stake, instead of applying every rule exhaustively.

4.2.3 Conditional Tacticals

Primitive operations are insufficient to represent complex normalisation tasks, which may require the combination of several tactics by means of tacticals such as *then*, *orelse*, *try* and *repeat*. Complete normalisation tactics are defined similarly to the inference case: given two syntactically defined subclasses, Σ_1 and Σ_2, of a language $\mathcal{L}$ and a theory T in $\mathcal{L}$, a normalisation tactic is *complete w.r.t.* Σ_1 *and* Σ_2 iff it represents a T-normaliser from a subset of Σ_1 into Σ_2. Conjunctive normal forming is an example of a normalisation task for which a complete tactic can be built.

Example 4.2.6 *Let* Fml *be the quantifier-free class of formulae defined by the rule*

$$fml \;:=\; atm|\neg fml|fml \equiv fml|fml \supset fml|fml \wedge fml|fml \vee fml$$

where atm denotes an atomic formula of the underlying language. Let Nrm *be the set of quantifier-free formulae in conjunctive normal form of the same language, defined by*

$$
\begin{array}{rcl}
\phi \equiv \psi & \Rightarrow & (\phi \supset \psi) \wedge (\psi \supset \phi) \\
\phi \supset \psi & \Rightarrow & \neg\phi \vee \psi \\
\neg(\phi \wedge \psi) & \Rightarrow & \neg\phi \vee \neg\psi \\
\neg(\phi \vee \psi) & \Rightarrow & \neg\phi \wedge \neg\psi \\
\neg\neg\phi & \Rightarrow & \phi \\
\phi \vee (\psi \wedge \rho) & \Rightarrow & (\phi \vee \psi) \wedge (\phi \vee \rho) \\
(\phi \wedge \psi) \vee \rho & \Rightarrow & (\phi \vee \rho) \wedge (\psi \vee \rho) \\
\phi \wedge (\psi \wedge \rho) & \Rightarrow & (\phi \wedge \psi) \wedge \rho \\
\phi \vee (\psi \vee \rho) & \Rightarrow & (\phi \vee \psi) \vee \rho
\end{array}
$$

Table 4.1: Rewrite Rules for Conjunctive Normal Forming

$$
\begin{array}{rcl}
nrm & := & cjt | nrm \wedge cjt \\
cjt & := & lit | cjt \vee lit \\
lit & := & atm | \neg atm
\end{array}
$$

A normaliser from Fml *into* Nrm *is supplied by the rewrite set listed in table 4.1. Since it is noetherian and locally confluent, the exhaustive application of its rules to quantifier-free formulae effectively reduces them to conjunctive normal form, and any path in the rewriting tree leads to the same normal form. However, if a comparative analysis of both initial and final classes is carried out in advance, their syntactic features can be explored in the construction of more efficient normalisers.*

- *Given that the final class,* Nrm, *does not contain either* $\equiv$ *or* $\supset$, remove *rules for these symbols should be applied in the first place. Considering that the* remove *rule for biconditionals introduces conditionals, which in turn also have to be removed, biconditionals are exhaustively eliminated before conditionals. Hence, the subclass defined by*

$$fml' \quad := \quad atm | \neg fml' | fml' \supset fml' | fml' \wedge fml' | fml' \vee fml'$$

is obtained before the class

$$fml'' \quad := \quad atm | \neg fml'' | fml'' \wedge fml'' | fml'' \vee fml''$$

- *In* Nrm, *negations occur only in the scope of conjunctions and disjunctions, whereas occurrences of disjunctions are limited to the scope of conjunctions. As a*

result, occurrences of negation have to be moved beneath the remaining symbols, followed by the stratification of disjunctions over conjunctions. The process is repeated until the desired strata,

$$\begin{array}{rcl} fml''' & := & cjt|fml''' \wedge fml''' \\ cjt & := & djt|cjt \vee cjt \\ djt & := & atm|\neg djt \end{array}$$

are generated.

* *The removal of negations poses a special problem, since its* remove *rule is partial dominant*[18]*. It can be applied to the conjecture at the initial stage, along with other* remove *rules, at the stratification stage, and at any stage when another rule is successfully applied, since any change of the context may make a partial rule applicable. A negation-free class,*

$$\begin{array}{rcl} fml'''' & := & cjt|fml'''' \wedge fml'''' \\ cjt & := & djt|cjt \vee cjt \\ djt & := & atm|\neg atm \end{array}$$

is then eventually obtained.

* *The final stage consists of reordering conjuncts and disjuncts in their respective strata.*

The complete normalisation tactic that represents the above normaliser,

```
try remove(bicond)
then try remove(cond)
     then try remove(neg)
          then try stratify(neg,[conj,disj]) then try remove(neg)
               then try stratify(disj,[conj]) then try remove(neg)
                    then try reorder(conj)
                         then try reorder(disj)
```

[18] The stratification rules for negation over conjunction and disjunction,

$$\begin{array}{rcl} \neg(\phi \wedge \psi) & \Rightarrow & \neg\phi \vee \neg\psi \\ \neg(\phi \vee \psi) & \Rightarrow & \neg\phi \wedge \neg\psi \end{array}$$

could both in principle be classified as partial non-dominant *remove* rules for conjunctions and disjunctions as well. However, since each of them introduces the other symbol in the rewritten formula, there is a potential risk of mutual cancellation of intended effects, as in the rewriting sequence

$$\neg(\neg(p \wedge q)) \Rightarrow \neg(\neg p \vee \neg q) \Rightarrow (\neg\neg p \wedge \neg\neg q)$$

where the first rule replaces $\wedge$ with $\vee$, and the second one reintroduces $\wedge$ at the expense of $\vee$. For this reason, they are not included in the set of *remove* rules.

requires the use of three primitive tactics. □

Even though the set of tacticals inherited from LCF is suitable for the construction of a large class of complete normalisers, there are cases where such constructs are insufficient to provide effective search control, as in the case of normalisation processes with multiple target classes, described in section 5.3.2. The implementation of richer search strategies then requires a set of *conditional tacticals.* They include

$$\textit{if } \psi \textit{ then } tac_1 \textit{ else } tac_2,$$

of rank ⟨ *form tactic tactic tactic* ⟩, which applies tac_1 in the case ψ, a formula that describes properties of the conjecture, is satisfiable, and tac_2, otherwise,

$$\textit{if } \psi \textit{ then } tac,$$

of rank ⟨ *form tactic tactic* ⟩, which applies *tac* whenever ψ is satisfiable, and

$$\textit{while } \psi \textit{ do } tac,$$

also of rank ⟨ *form tactic tactic* ⟩, which applies *tac* whilst ψ remains satisfiable. The last two tacticals are actually derived: *if* ψ *then tac* can be represented as *if* ψ *then tac else failtac*, whereas *while* ψ *do tac* is equivalent to *repeat* (*if* ψ *then tac*). Conditional tacticals allow the construction of context-sensitive structures for rule ordering and selection that are more flexible than the plain application of ordered rewrite sets. In their presence, the execution of a tactic involves, besides the application of rewrite rules, the evaluation of conditions.

4.3 Normalisation Plans

Given an equivalence relation, any pair of classes of expressions $\langle \Sigma_1, \Sigma_2 \rangle$ defines a normalisation problem and provides the description for a normaliser, under the form of a domain, a range and a property that each normalised expression must satisfy (i.e. input and output expressions must be equivalent). If both Σ_1 and Σ_2 are syntactically

defined, a rewrite-based normaliser $\mathcal{N}$: $\Sigma_1 \rightarrow \Sigma_2$ may be built from the composition of elementary normalisation tactics, as described in example 4.2.6. After their selection, it is possible to determine the rewrite set $\mathcal{R}$ required to complete the implementation of $\mathcal{N}$. The task at stake can then be formally defined as

$$(\exists r)(\exists \mathcal{N})(\mathcal{N} \subseteq r \;\&\; dom(\mathcal{N}) \subseteq \Sigma_1 \;\&\; rng(\mathcal{N}) = \Sigma_2)$$

where r is the relation defined by $\mathcal{R}$.

There are, however, alternative ways of specifying $\mathcal{N}$. If a rewrite set $\mathcal{R}$ is given in place of Σ_1, a normaliser derived from the description $\langle \mathcal{R}, \Sigma_2 \rangle$ has the form $\mathcal{N}$: $\mathcal{E} \rightsquigarrow \Sigma_2$, where $\mathcal{E}$ is the domain of $\mathcal{R}$. The construction of $\mathcal{N}$ then involves the selection of tactics which gradually reduce some elements of $\mathcal{E}$ into Σ_2 and employ, for this purpose, only those rules available in $\mathcal{R}$. The process therefore requires finding a constructive proof for

$$(\exists \mathcal{N})(\mathcal{N} \subseteq r \;\&\; dom(\mathcal{N}) \subseteq \mathcal{E} \;\&\; rng(\mathcal{N}) = \Sigma_2)$$

where r is the relation defined by $\mathcal{R}$.

Two final descriptions replace the input class with a particular formula. If $\phi \notin \Sigma_2$, it has then to be established whether there is a rewrite-based normaliser $\mathcal{N}$ such that $\mathcal{N}(\phi) \in \Sigma_2$. Alternatively, if a rewrite set $\mathcal{R}$ is also given, it may be relevant to determine whether there is a normaliser $\mathcal{N}$ derived from $\mathcal{R}$ such that $\mathcal{N}(\phi) \in \Sigma_2$. Each case is respectively representable as

$$(\exists r)(\exists \mathcal{N})(\mathcal{N} \subseteq r \;\&\; \mathcal{N}(\phi) \in \Sigma_2)$$

and

$$(\exists \mathcal{N})(\mathcal{N} \subseteq r \;\&\; \mathcal{N}(\phi) \in \Sigma_2)$$

where r is the relation defined by $\mathcal{R}$. The description in each case is provided by $\langle \phi, \Sigma_2 \rangle$ and $\langle \phi, \mathcal{R}, \Sigma_2 \rangle$.

As in the case of proof strategies, the construction of a normaliser for any of the above descriptions from a set of elementary normalisation tactics faces potential search

problems. The task of solving these problems is simplified once a (partial) specification for primitive tactics is given.

4.3.1 Primitive Methods

Partial specifications for normalisation tactics can be provided with increasing degree of accuracy depending on the amount of knowledge extracted from the corresponding rule set. As a result, more than one method for a single tactic may be exhibited. A *primitive normalisation method* is represented by the predicate *norm_method*/6, defined as

norm_mthd (*name* (*...Args...*),
Input class,
Preconditions,
Effects,
Output class,
tactic(*...Args...*)
).

The *name*, *preconditions*, *effects* and *tactic* have the same role as in the definition of a *method*. The *input class* defines syntactic properties of the domain of the corresponding tactic, whereas the *output class* refers to the range of the tactic. Let $m(\boldsymbol{v})$ be a primitive normalisation method, where $m(\boldsymbol{v})$ is an abbreviation for $\lambda\Gamma\lambda\Gamma'\ m(\Gamma, \boldsymbol{v}, \Gamma')$ and $\boldsymbol{v}$ is the list of parameters of m. If $\{\psi_1, \ldots, \psi_n\}$ is the set of preconditions and $\{\psi'_1, \ldots, \psi'_n\}$ is the set of effects (or postconditions), then

$$m(\boldsymbol{v})(\Sigma, \Sigma')$$
$$\equiv$$
$$(\exists \boldsymbol{u})(\psi_1\{\Sigma, \boldsymbol{v}, \boldsymbol{u}\} \wedge \cdots \wedge \psi_n\{\Sigma, \boldsymbol{v}, \boldsymbol{u}\} \wedge \psi'_1\{\Sigma, \Sigma', \boldsymbol{v}, \boldsymbol{u}\} \wedge \cdots \wedge \psi'_m\{\Sigma, \Sigma', \boldsymbol{v}, \boldsymbol{u}\})$$

is a logical consequence of the underlying specification theory, i.e. the set of preconditions and postconditions are both simultaneously satisfiable for a particular assignment for $\boldsymbol{v}, \Gamma$ and Γ'. $\boldsymbol{u}$ is a list of additional variables.

The main difference with respect to inference methods is the presence of classes of expressions, rather than a single goal or a list of subgoals, in the input and output slots.

This feature does not exclude applications to specific formulae, though, since classes may be unitary. Such a representation is particularly relevant for the construction of normalisers defined in terms of pairs of classes of formulae.

The simplest specification for a normalisation tactic does not take into account any information about the set of rewrite rules, and assumes that a complete rewrite set is available for every symbol or lists of symbols of the object language. A conjecture is abstractly represented by means of a structured list containing its symbols (other than variables) and their relative positions, which describes a class of formulae that share the same structure. The use of abstraction functions leads, as in the above case, to the removal of irrelevant information in the representation of expressions, and is an important element of program specification[19]. The method *remove*1, defined as

```
norm_mthd(remove1(Sym),
          CjtSymLst,
          [member(Sym,CjtSymLst),
           rem_rul(Sym,_)],
          [delete(Sym,CjtSymLst,NewCjtSymLst)],
          NewCjtSymLst,
          remove(Sym)
         ).
```

where

rem_rul(Sym,Rul) iff *Rul* is a *remove* rule for *Sym*

provides minimal information about the tactic *remove*. *CjtSymLst* represents the class to which the conjecture belongs. *Remove*1*(Sym)* assumes the application of the corresponding tactic succeeds whenever *Sym* occurs in *CjtSymLst* and there is at least one *remove* rule for this symbol: it does not check, however, whether any of such rules is actually applicable to the conjecture. The abstract representation of the output class is obtained from *CjtSymLst* through the deletion of occurrences of the removed symbol. Since the rhs (right hand side) expressions of the corresponding rules are not examined, it is not possible to determine whether new symbols are introduced in the conjecture as a result of rewriting. The method *stratify*1, on the other hand, is defined as

[19] See for instance [Liskov & Guttag 86], p. 56-7.

```
norm_mthd(stratify1(Sym,SymLst),
          CjtSymLst,
          [lnk_occ_sym(Sym,SymLst,CjtSymLst),
           str_rul(Sym,_,_)],
          [exchange(Sym,SymLst,CjtSymLst,NewCjtSymLst)],
          NewCjtSymLst,
          stratify(Sym,SymLst)
         ).
```

where

lnk_occ_sym(Sym,SymLst,CjtSymLst)	iff	there is a symbol of *SymLst* which occurs in the immediate scope of *Sym* in the list *CjtSymLst*
str_rul(Sym1,Sym2,Rul)	iff	*Rul* is a *stratify* rule for *Sym*1 w.r.t. *Sym*2
exchange(Sym,SymLst,CjtSymLst,NewCjtSymLst)	iff	*NewCjtSymLst* is generated from *CjtSymLst* by interchanging occurrences of symbols of *SymLst* in the scope of *Sym* with *Sym*

*Stratify*1*(Sym,SymLst)* succeeds whenever there is an occurrence of one of the symbols of *SymLst* in the immediate scope of *Sym*, and there is at least one stratification rule for *Sym*: as in the case of *remove*1, the applicability of the rule to the conjecture is not checked. The resulting list of symbols, *NewCjtLst*, is generated by simple exchange of *Sym* with the symbols of *SymLst*.

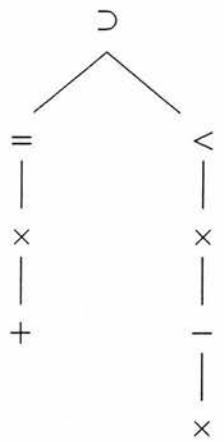
Example 4.3.1 *The formula*

$$x \times (y + z) = y \supset (x \times x - z) \times z < y$$

can be abstractly represented as a list of symbols,

$$[\ \supset,\ \ [=,[\times,[+]],\ <,[\times,[-,[\times]]]\ ]\ ]$$

or a symbolic tree,



where the order of branches is immaterial. If it is supplied to remove1($\supset$), *all the occurrences of* $\supset$ *are deleted, thus generating*

$$[[=,[\times,[+]],\ <,[\times,[-,[\times]]]]]$$

If the resulting representation is supplied to the method stratify1($\times,[+,-]$), *the output list,*

$$[[=,[+,[\times]],\ <,[-,[\times,[\times]]]]],$$

does not contain occurrences of either sum or subtraction in the immediate scope of multiplication. □

The accuracy of the specification provided by each of these methods depends upon the type of rules available to the tactic. Let Σ_1 and Σ_1' be the domain and range of a normalisation tactic in a language $\mathcal{L}$, and let Σ_2 and Σ_2' be the largest classes that respectively satisfy the preconditions and postconditions of the corresponding method. In the case of *remove*1, Σ_2 is syntactically defined as the subclass of all formulae ϕ of $\mathcal{L}$ such that

$$occ\_sym(S,\phi)$$

(i.e. those that contain a symbol S), whereas Σ_2' is contained in the complement of Σ_2 w.r.t. $Fml_{\mathcal{L}}$, which satisfies

$$\neg occ\_sym(S, \phi)$$

(i.e. the class of formulae of $\mathcal{L}$ in which S is absent). *Remove*1(S) therefore defines a relation in the domain $\wp(\Sigma_2) \times \wp(\Sigma_2')$, since any subset of Σ_2 that contains S satisfies the preconditions of *remove*1, whereas the corresponding output class must be a subset of Σ_2', since it does not contain S. For instance, given $\mathcal{L}_{PA} = \{0, 1, \mathrm{s}, +, \times\}$, if $+$ is the chosen symbol, Σ_2 can be abstractly represented as

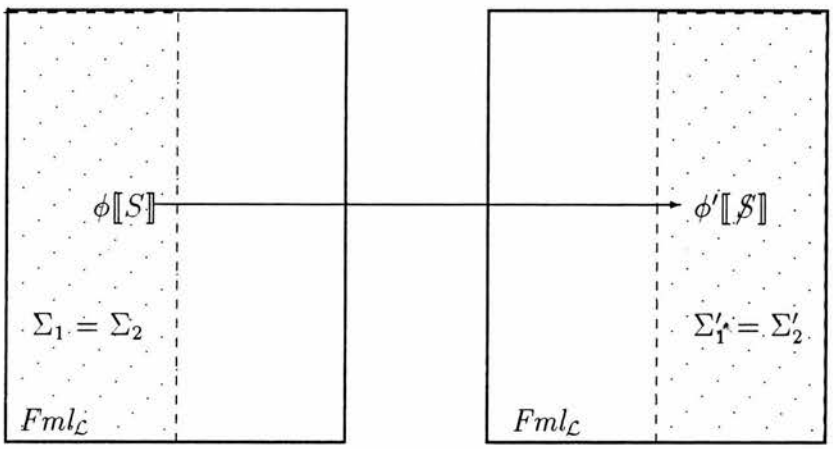
$$\Sigma_2 \;=\; [+, =] \cup [+, =, \neg] \cup \cdots \cup [+, =, \neg, \wedge] \cup \cdots \cup [+, \neg, \wedge, \vee, \supset, \equiv, \exists, \forall, \top, \bot, =, 0, 1, \mathrm{s}, \times]$$

which is the union of the (disjoint) subclasses of formulae that respectively contain, as sole non-variable symbols, $+$ and $=$, $+$, $=$ and $\neg$, etc. The removal of $+$ from Σ_2 generates

$$\Sigma_2' \;=\; [=] \cup [=, \neg] \cup \cdots \cup [=, \neg, \wedge] \cup \cdots \cup [\neg, \wedge, \vee, \supset, \equiv, \exists, \forall, \top, \bot, =, 0, 1, \mathrm{s}, \times]$$

as output: in particular, $\langle [+, =, \neg, \wedge], [=, \neg, \wedge] \rangle \in$ *remove*1(+). Since the method *remove*1(S) assumes that there is always a total *remove* rule for S, when such a rule is actually available to the corresponding tactic, the (largest) *input class* for *remove*1(S) and the domain of the *remove*(S) tactic coincide, as shown in figure 4.1. However, if the set of *remove* rules for S is incomplete, Σ_2 is always larger than the domain of the tactic, Σ_1, since there are formulae in Σ_2 from which no occurrence of S can be removed. For this reason, *remove*1(S) provides a partial specification for *remove*(S), considering that *remove*(S) is just a partial function in Σ_2. Moreover, the range of the tactic may not be entirely contained in Σ_2', since there are formulae from which only some occurrences of S can be removed, as it is shown in figure 4.2.

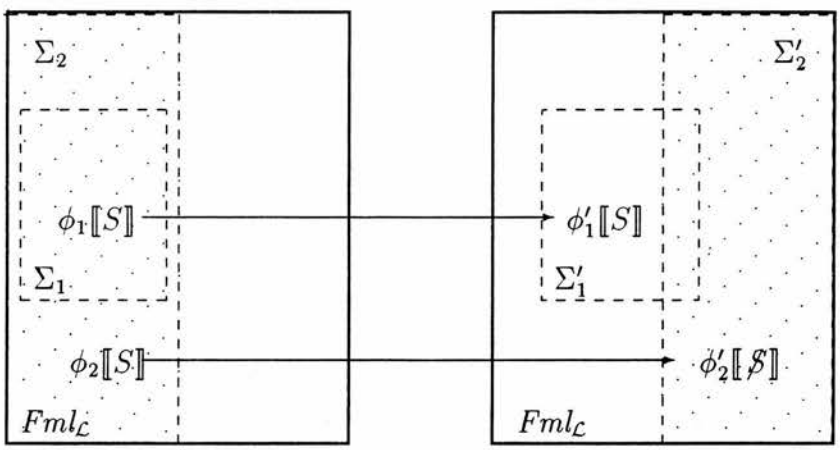
More accurate specifications for a normalisation tactic are obtained once information about the rule set is taken into account. For instance, the preconditions of *remove*1 can be extended to include a test for the applicability of a rule against each particular conjecture ϕ. The input class, on the other hand, may contain ϕ as sole element. For the new method,



tactic	$remove(S)$	$\in$	$\Sigma_1'^{\Sigma_1}$
method	$remove1(S)$	$\subset$	$\wp(\Sigma_2) \times \wp(\Sigma_2')$

When a total *remove* rule for S is available, the (largest) *input class* for $remove1(S)$, Σ_2, and the domain of the $remove(S)$ tactic, Σ_1, coincide.

Figure 4.1: Totally removable symbols



tactic	$remove(S)$	$\in$	${\Sigma'_1}^{\Sigma_1}$
method	$remove1(S)$	$\subset$	$\wp(\Sigma_2) \times \wp(\Sigma'_2)$

In the absence of total *remove* rules (or complete sets of *remove* rules) for S, the domain of the tactic $remove(S)$, Σ_1, is just a proper subset of the (largest) *input class* for $remove1(S)$, Σ_2, given that the latter always presupposes the existence of total rules.

Figure 4.2: Partially removable symbols

```
norm_mthd(remove2(Sym),
          [Cjt],
          [sym(Cjt,CjtSymLst),
           occ_sym(Sym,CjtSymLst),
           rem_rul(Sym,Rul),
           applicable(Rul,Cjt)],
          [delete(Sym,CjtSymLst,NewCjtSymLst)],
          NewCjtSymLst,
          remove(Sym)
         ).
```

the largest class Σ_2 that satisfies the preconditions coincides with the domain of the tactic *remove*, Σ_1. Σ_1' and Σ_2', however, still do not coincide, since *remove*2 presupposes that every occurrence of *Sym* is removed from every conjecture. A similar version can be built for *stratify*2. In the limit, the specification provided by the method entirely coincides with the tactic, for any set of rewrite rules, both for *remove*,

```
norm_mthd(remove(Sym),
          [Cjt],
          [occ_sym(Sym,Cjt),
           rem_rul(Sym,Rul),
           applicable(Rul,Cjt)],
          [exh_rem(Sym,Cjt,NewCjt)],
          [NewCjt],
          remove(Sym)
         ).
```

where

applicable(Rul,Cjt)	iff	*Cjt* has a subexpression against which *Rul* can be matched
exh_rem(Sym,Cjt,NewCjt)	iff	*NewCjt* is generated from *Cjt* by the exhaustive application of *remove* rules for *Sym*

and *stratify*,

```
norm_mthd(stratify(Sym,SymLst),
          [Cjt],
          [lnk_occ_sym(Sym,SymLst,Cjt),
           str_rul(Sym,OthSym,Rul),
           member(OthSym,SymLst),
           applicable(Rul,Cjt)],
          [exh_str(Sym,SymLst,Cjt,NewCjt)],
          [NewCjt],
          stratify(Sym,SymLst)
         ).
```

where

exh_str(Sym,SymLst,Cjt,NewCjt) iff *NewCjt* is generated from *Cjt* by the exhaustive application of *stratify* rules for *Sym* w.r.t. *SymLst*

or for any other tactic. Given that both input and output classes are unitary, they are normally indicated by their single element instead, hence *remove* is usually denoted as $\lambda\phi\lambda\phi'$ *remove*(ϕ, Sym, ϕ').

Even when a method exactly describes the domain and image of a tactic, the construction of strategies with methods does not amount to the same process at the tactic level. While normalisation tactics apply rules to an expression, methods assess properties of the input expression in the preconditions before rewriting starts. In the case of *remove*, the corresponding tactic is applicable only if the chosen symbol occurs in the current conjecture; if it does not, the method fails, without any consultation of the rewrite set. At the tactic level, however, each *remove* rule for the chosen symbol would have to be tentatively matched against the conjecture before the inapplicability of the tactic could be asserted. Preconditions therefore establish an essential distinction between methods and tactics.

Methods of the last type are relevant to the development of normalisers for descriptions of the forms $\langle\phi, \Sigma_2\rangle$ or $\langle\phi, \mathcal{R}, \Sigma_2\rangle$, i.e. those given in terms of a formula (and a rule set) and a (decidable) class to which this formula does not belong. The development of normalisers for such descriptions is further examined in chapter 6.

4.3.2 Methodicals

The combination of primitive methods requires the use of *methodicals* such as *then, orelse, try, repeat, if_then, if_then_else* and *while_do*, which share similar properties with the homonymous tacticals. Conditional methodicals represent a convenient mechanism for extending the set of *preconditions* of a primitive method m: every composite method of the form $(if\ \psi\ then\ m)$ could in principle be replaced by a new atomic method generated from m by the inclusion of ψ amongst its preconditions. The introduction of methodicals adds the set of composite methods to the primitive core.

Definition 4.3.1 (Normalisation Methods)

The set of normalisation methods *is recursively defined as follows.*

i. idmethod *and* failmethod *are normalisation methods.*

ii. Primitive normalisation methods (*e.g.* remove(Sym), stratify($Sym, SymLst$) *and* reorder(Sym)) *are normalisation methods.*

iii. If m_1 *and* m_2 *are normalisation methods, then* (m_1 then m_2), (m_1 orelse m_2), (try m_1) *and* (repeat m_1) *are normalisation methods.*

iv. If m_1 *and* m_2 *are normalisation methods and* ψ *is a formula of the underlying specification language, then* (if ψ then m_1), (if ψ then m_1 else m_2), (while ψ do m_1) *are normalisation methods.*

v. Only the objects that satisfy one of the conditions above are normalisation methods.

A normalisation method is conditional *if it contains a conditional methodical, and* non-conditional *otherwise.*

Composite methods may be also identified by a name, which is linked to two types of parameters. Essential parameters are strictly those inherited by a method from its primitive components, which in turn coincide, by definition, with the parameters of the corresponding tactic. Inessential parameters are those that occur in the condition of conditional methodicals.

Definition 4.3.2 (Essential & Inessential Parameters)

Let $m(\boldsymbol{v}, \boldsymbol{v}')$ *be a normalisation method, where* $m(\boldsymbol{v}, \boldsymbol{v}')$ *is an abbreviation for* $\lambda\Sigma\lambda\Sigma'\ m(\Sigma, \boldsymbol{v}, \boldsymbol{v}', \Sigma')$. *The finite sequences of variables* $\boldsymbol{v}$ *and* $\boldsymbol{v}'$ *are respectively* essential *and* inessential parameters *of* m *iff one of the following conditions holds.*

i. $\boldsymbol{v}' = [\,]$ *and* $m(\boldsymbol{v})$ *is a primitive method.*

ii. $m(\boldsymbol{v}, \boldsymbol{v}')$ *has any of the forms*

$$m_1(\boldsymbol{u_1}, \boldsymbol{u'_1}) \text{ then } m_2(\boldsymbol{u_2}, \boldsymbol{u'_2})$$
$$m_1(\boldsymbol{u_1}, \boldsymbol{u'_1}) \text{ orelse } m_2(\boldsymbol{u_2}, \boldsymbol{u'_2})$$
$$\text{repeat } m_1(\boldsymbol{u_1}, \boldsymbol{u'_1})$$
$$\text{try } m_1(\boldsymbol{u_1}, \boldsymbol{u'_1})$$

where

(a) $\boldsymbol{u_1}$ *and* $\boldsymbol{u_2}$ *are the essential parameters of* m_1 *and* m_2*, and* $\boldsymbol{u_1} \cup \boldsymbol{u_2} = \boldsymbol{v}$

(b) $\boldsymbol{u'_1}$ *and* $\boldsymbol{u'_2}$ *are the inessential parameters of* m_1 *and* m_2*, and* $\boldsymbol{u'_1} \cup \boldsymbol{u'_2} = \boldsymbol{v'}$

iii. $m(\Sigma, \boldsymbol{v}, \boldsymbol{v}', \Sigma')$ *has the any of the forms*

$$\text{if } \psi(\Sigma, \boldsymbol{w}, \boldsymbol{w}') \text{ then } m_1(\Sigma, \boldsymbol{u_1}, \boldsymbol{u'_1}, \Sigma') \text{ else } m_2(\Sigma, \boldsymbol{u_2}, \boldsymbol{u'_2}, \Sigma')$$
$$\text{if } \psi(\Sigma, \boldsymbol{w}, \boldsymbol{w}') \text{ then } m_1(\Sigma, \boldsymbol{u_1}, \boldsymbol{u'_1}, \Sigma')$$
$$\text{while } \psi(\Sigma, \boldsymbol{w}, \boldsymbol{w}') \text{ do } m_1(\Sigma, \boldsymbol{u_1}, \boldsymbol{u'_1}, \Sigma')$$

where

(a) $\boldsymbol{u_1}$ *and* $\boldsymbol{u_2}$ *are the essential parameters of* m_1 *and* m_2 *such that* $\boldsymbol{u_1} \cup \boldsymbol{u_2} = \boldsymbol{v}$

(b) $\boldsymbol{u'_1}$ *and* $\boldsymbol{u'_2}$ *are the inessential parameters of* m_1 *and* m_2

(c) $\boldsymbol{w} \subseteq \boldsymbol{v}$, $\boldsymbol{w}' \cap \boldsymbol{v} = \emptyset$ *and* $\boldsymbol{v}' = \boldsymbol{u'_1} \cup \boldsymbol{u'_2} \cup \boldsymbol{w}'$

Example 4.3.2 *Let* (λ Exp λ NewExp cond_remove) *be a method where* cond_remove *is defined as*

```
if [occ_sym(Exp,Sym,Pos)
    remove(Exp,Sym,NewExp)
    idmethod]
```

Sym and Pos are respectively its essential and inessential parameters. The instance cond_remove($\leq$,[ ]) *specifies a tactic whose domain is the set of atomic formulae dominated by* $\leq$. □

Concerning the meaning of a composite normalisation method, let $m(\boldsymbol{v})$, $m_1(\boldsymbol{v_1})$ and $m_2(\boldsymbol{v_2})$ be methods and $\psi(\Sigma, \boldsymbol{u})$ be a formula.

(a) If $m(\boldsymbol{v})$ has the form ($m_1(\boldsymbol{v_1})$ *then* $m_2(\boldsymbol{v_2})$), then

$$m(\boldsymbol{v})(\Sigma, \Sigma') \equiv (\exists \Sigma'')(m_1(\boldsymbol{v_1})(\Sigma, \Sigma'') \wedge m_2(\boldsymbol{v_2})(\Sigma'', \Sigma'))$$

(b) If $m(\boldsymbol{v})$ has the form $(m_1(\boldsymbol{v_1})\ \mathit{orelse}\ m_2(\boldsymbol{v_2}))$, then

$$m(\boldsymbol{v})(\Sigma, \Sigma') \equiv m_1(\boldsymbol{v_1})(\Sigma, \Sigma') \vee (m_2(\boldsymbol{v_2})(\Sigma, \Sigma') \wedge (\forall \Sigma'') \neg m_1(\boldsymbol{v_1})(\Sigma, \Sigma''))$$

(c) If $m(\boldsymbol{v})$ has the form $(\mathit{repeat}\, m'(\boldsymbol{v}))$, then

$$\begin{aligned} \mathit{repeat}(m'(\boldsymbol{v}))(\Sigma, \Sigma') \equiv\ & (\exists \Sigma'')(m'(\boldsymbol{v})(\Sigma, \Sigma'') \wedge \mathit{repeat}(m'(\boldsymbol{v}))(\Sigma'', \Sigma')) \\ & \vee \\ & \mathit{idmethod}(\Sigma, \Sigma') \wedge \neg(\exists \Sigma''') m'(\boldsymbol{v})(\Sigma, \Sigma''') \end{aligned}$$

(d) If $m(\boldsymbol{v})$ has the form $(\mathit{try}\, m'(\boldsymbol{v}))$, then

$$m(\boldsymbol{v})(\Sigma, \Sigma') \equiv m'(\boldsymbol{v})(\Sigma, \Sigma') \vee (\mathit{idmethod})(\Sigma, \Sigma') \wedge (\forall \Sigma'') \neg m'(\boldsymbol{v})(\Sigma, \Sigma'')$$

(e) If $m(\boldsymbol{v})$ has the form $\lambda\Gamma\lambda\Gamma'$ $(\mathit{if}\ \psi(\Gamma, \boldsymbol{u})\ \mathit{then}\ m_1(\Gamma, \boldsymbol{v_1}, \Gamma')\ \mathit{else}\ m_2(\Gamma, \boldsymbol{v_2}, \Gamma'))$, then

$$m(\boldsymbol{v})(\Sigma, \Sigma') \equiv (\psi(\Sigma, \boldsymbol{u}) \wedge m_1(\boldsymbol{v_1})(\Sigma, \Sigma')) \vee (\neg\psi(\Sigma, \boldsymbol{u}) \wedge m_2(\boldsymbol{v_2})(\Sigma, \Sigma'))$$

A method that specifies a complete normalisation tactic is a *proof plan for normalisation.* A plan is *special-purpose* if it specifies a particular normaliser for a subclass of formulae, as e.g. in prenex normal forming. A *general-purpose proof plan*, on the other hand, represents a family of special-purpose plans that share a common structure[20]. Given a decidable class Σ_2, an extended syntactically defined class Σ_1 in a language $\mathcal{L}$ and a rewrite set $\mathcal{R}$ of rules valid in a theory T, a T-normaliser from Σ_1 into Σ_2 could be implemented from descriptions such as $\langle \Sigma_1, \Sigma_2 \rangle$, $\langle \mathcal{R}, \Sigma_2 \rangle$, $\langle \phi, \Sigma_2 \rangle$ or $\langle \phi, \mathcal{R}, \Sigma_2 \rangle$, where ϕ belongs to the domain of $\mathcal{R}$. In each case, a tactic is mechanically created in at least two ways.

i. The description is supplied to a *planner*, which combines primitive methods by means of methodicals until a suitable special-purpose plan is completed. The original problem is implicitly decomposed into a series of intermediate syntactically defined classes, such that each pair of consecutive classes is linked by a primitive normalisation tactic.

[20] See [Bundy 91], p. 8.

ii. Parameters of a general-purpose plan are instantiated until a tactic that satisfies the chosen description is obtained.

Hence processes for the construction of a special-purpose proof plan vary along two dimensions, the type of the description and the nature of the construction. In the first case, if a description for the extended class is given, then the necessary rewrite rules have to be sought, otherwise when the rewrite set is supplied with the description, the extended class has to be built instead. For the second case, if the description is supplied to a planner, planning techniques must be made available to it, whereas if it is generated from a general-purpose plan, a suitable set of such plans must be given. The dynamic generation of special-purpose plans for a description $\langle \Sigma_1, \Sigma_2 \rangle$ is discussed in the next section.

4.3.3 Automated Planning

Planners perform search in a set of methods. Since there are several methods for each normalisation tactic, the required level of accuracy of the specification has to be determined before search starts. When the description for a normaliser has the form of a pair of syntactically defined classes and no rule set is given, methods such as *remove*1, *stratify*1 and *reorder*1 are a natural choice.

Example 4.3.3 *Let $\mathcal{L}$ be a language, and $\Sigma_1 \subseteq Fml_{\mathcal{L}}$ be the class such that $S_1, \ldots, S_n$ occur in every one of its elements. Let Σ_2 be the class of the same language whose elements have at least an occurrence of one of the above symbols. Finally, let Σ_3 be the class of formulae in which the above symbols are absent. They can be represented as*

$$\Sigma_1 = \{\phi \in Fml_{\mathcal{L}} \mid occ\_sym(\phi, S_1) \;\&\; occ\_sym(\phi, S_2) \;\&\; \ldots \;\&\; occ\_sym(\phi, S_n)\} \quad (*)$$

$$\Sigma_2 = \{\phi \in Fml_{\mathcal{L}} \mid occ\_sym(\phi, S_1) \;\vee\; occ\_sym(\phi, S_2) \;\vee\; \ldots \;\vee\; occ\_sym(\phi, S_n)\}$$

and

$$\Sigma_3 = \{\phi \in Fml_{\mathcal{L}} \mid \neg occ\_sym(\phi, S_1) \;\&\; \neg occ\_sym(\phi, S_2) \;\&\; \ldots \;\&\; \neg occ\_sym(\phi, S_n)\}$$

Let $\langle \Sigma_1, \Sigma_3 \rangle$ and $\langle \Sigma_2, \Sigma_3 \rangle$ be descriptions for two normalisers.

i. Once $\langle \Sigma_1, \Sigma_3 \rangle$ *is supplied to a planner, it looks for a method that is applicable to* Σ_1. *Since there is no reference to the relative positions of symbols in this class, reorganisation methods are excluded, leaving only* remove1 *and* reduce1 *for consideration.*

(a) If it is assumed that a complete set of remove *rules for all the mentioned symbols can be found, then there are n methods, or instances of methods, that can be applied to the mentioned class:* remove1(S_1), ..., remove1(S_n). *The preconditions for these methods require the occurrence of* S_i *in the initial class, which are satisfied by the definition of* Σ_1. *Without loss of generality, let* remove1(S_1) *be the first chosen method. The pre- and postcondition of the method act as a rewrite rule which is applied to the definition of the input class, thus introducing* $\neg occ\_sym(\phi, S_1)$ *in expression* $(*)$,

$$\neg \text{ occ\_sym}(\phi, S_1) \ \& \ \text{occ\_sym}(\phi, S_2) \ \& \ \ldots \ \& \ \text{occ\_sym}(\phi, S_n)$$

thus defining a new class, Σ_1'. *The process is iterated until* Σ_3 *is eventually obtained (on the assumption that the removal of* S_i *does not introduce* S_j, $1 \leq i \leq n, j < i$). *The resulting method is*

$$\textit{remove1}(S_1) \text{ then } \textit{remove1}(S_2) \text{ then } \ldots \text{ then } \textit{remove1}(S_n)$$

(b) In the event not all of the symbols $S_1, \ldots, S_n$ *are expected to have complete sets of* remove *rules, the order for the removal of symbols is relevant. A more complex plan,*

$$\begin{gathered}
\textit{remove1}(S_1) \text{ then } \textit{remove1}(S_2) \text{ then } \ldots \text{ then } \textit{remove1}(S_n) \\
\text{orelse} \\
\textit{remove1}(S_2) \text{ then } \textit{remove1}(S_1) \text{ then } \ldots \text{ then } \textit{remove1}(S_n) \\
\text{orelse} \\
\vdots \\
\text{orelse} \\
\textit{remove1}(S_n) \text{ then } \textit{remove1}(S_{n-1}) \text{ then } \ldots \text{ then } \textit{remove1}(S_1)
\end{gathered}$$

is then required.

ii. For the pair $\langle \Sigma_2, \Sigma_3 \rangle$, *a suitable plan,*

try $remove1(S_1)$ then try $remove1(S_2)$ then ... then try $remove1(S_n)$

could be similarly built. □

A normalisation planner, therefore, links a description for a normaliser with an implementation in the form of a tactic. For the above example, once the tactic is defined, *remove* rules for $S_1, \ldots, S_n$ have to be sought.

For the particular normalisation problem of decidable subclass extension, given a T-decidable class Σ in $\mathcal{L}$, the description of a normaliser for a generic extended class is based on the full set of formulae of $\mathcal{L}$. When T is undecidable, no extended class coincides with the full set of formulae, hence the description $\langle Fml_{\mathcal{L}}, \Sigma \rangle$ is partial, and the described normaliser $\mathcal{N}$ is a partial function,

$$\mathcal{N}\colon Fml_{\mathcal{L}} \rightsquigarrow \Sigma$$

This problem can be represented as the search for a constructive proof for the conjecture

$$(\exists r)(\exists r')(\forall \phi_1)(\forall \phi_2)(r' \subseteq r \;\&\; \boldsymbol{U} \;\&\; ((\phi_1 \in Fml_{\mathcal{L}} \;\&\; \phi_2 \in \Sigma \;\&\; r'(\phi_1, \phi_2)) \supset (T \models \phi_1 \equiv \phi_2))) \quad (\dagger)$$

where

(a) r is the binary relation defined by a rewrite set $\mathcal{R}$,

(b) r' defines a partial function $\mathcal{N}_{r'}\colon Fml_{\mathcal{L}} \rightsquigarrow \Sigma$, and

(c) $\boldsymbol{U}$, defined as

$$(\forall \psi_1)(\forall \psi_2)(\forall \psi_3)[(r'(\psi_1, \psi_2) \;\&\; r'(\psi_1, \psi_3)) \supset (\psi_1 \in Fml_{\mathcal{L}} \;\&\; \psi_2 \in \Sigma \;\&\; \psi_2 \stackrel{s}{=} \psi_3)]$$

asserts the (syntactic) unicity of $\mathcal{N}_{r'}(\phi)$, for any formula ϕ in the domain of $\mathcal{N}_{r'}$.

More precisely, the construction of a partial normaliser for $\langle Fml_{\mathcal{L}}, \Sigma \rangle$ requires a rewrite set $\mathcal{R}$ from which $\mathcal{N}_{r'}$ is extracted by the introduction of a suitable control function. The range of $\mathcal{N}_{r'}$ is Σ, and for any formula $\phi \in dom(\mathcal{N}_{r'})$, ϕ and $\mathcal{N}_{r'}(\phi)$ are equivalent in T. Given that both $Fml_{\mathcal{L}}$ and Σ are syntactically defined, the atoms $\phi_1 \in Fml_{\mathcal{L}}$ and $\phi_2 \in \Sigma$ can be replaced by formulae $\boldsymbol{F_1}(\phi_1)$ and $\boldsymbol{F_2}(\phi_2)$ of the specification language. Since a planner is limited to the construction of special-purpose plans, (†) may be simplified to

$$(\exists p)(\exists v_1)\ldots(\exists v_n)(p(v_1,\ldots,v_n)(\Sigma_1,\Sigma_2))$$

where p is a normalisation plan.

The application of planners to a description is not examined in the present study. Attention is otherwise concentrated on the generation of normalisers from general-purpose plans, which involves proving that

$$(\exists v_1)\ldots(\exists v_n)(p(v_1,\ldots,v_n)(\Sigma_1,\Sigma_2))$$

where $\lambda\Gamma_1\lambda\Gamma_2\, p(\Gamma_1, v_1, \ldots, v_n, \Gamma_2)$ is a plan and $\langle\Sigma_1, \Sigma_2\rangle$ is a description. The instantiation of parameters is handled e.g. by a resolution-based first-order theorem prover[21].

4.4 Conclusions

The extension of decidable classes is a special case of formula normalisation. Syntactically defined sets in general, and context-free grammars in particular, are amongst the recursive classes for which a specification language has been identified. Normalisers may be represented by means of rewrite rules, whose exhaustive application nonetheless usually leads to inefficiency.

Effective procedures for the reduction of a class into a decidable domain are derived from rewrite sets after the incorporation of control functions for the selection of rules. Proof plans for normalisation in particular limit the application of rules through the syntactic analysis of classes of formulae. Given a description for a normaliser, its generation under the proof planning approach may take place either by the application of a planner to a set of methods, or by a simpler process of parameter instantiation in a general-purpose plan. Before such plans are precisely defined, the analysis of a few special-purpose plans in the next chapter clarifies the nature of the transformations that take place inside decision procedures.

[21] Special and general-purpose plans are formally defined in the next chapter.

Chapter 5

Special & General Purpose Proof Plans

General-purpose plans represent parameterised families of normalisers. Once parameters are instantiated, a range of complete normalisation tactics is derived, covering operators such as decision procedures and reduction functions. Special and general-purpose plans are formally defined in section 5.1. The structure of two decision procedures based on quantifier-elimination is examined in section 5.2, followed by a tentative representation for rewrite-based decision procedures in section 5.3.

5.1 Recursive Plans

A proof plan for normalisation is a method that specifies a complete tactic for a normalisation task. The generality of a plan derives from the occurrence of free variables among its parameters. Some of the parameters identify symbols, or lists of symbols, present in primitive tactics. For instance, the plan

```
repeat(try remove(Sym1) then try stratify(Sym2,SymLst))
```

is general-purpose due to the presence of variables in its parameters. Its instance,

```
repeat(try remove(neg) then try stratify(neg,[conj,disj]))
```

is a special-purpose plan for the reduction of formulae containing negations, conjunctions and disjunctions as sole logical symbols to the subclass where negations occur

only in the scope of literals. The family of special-purpose plans associated with a general-purpose plan is made up of its variable-free (or *ground*) instances[1].

Definition 5.1.1 (Proof plan for normalisation)

i. The set of proof plans for normalisation (*ppn*) *is recursively defined as follows.*

(a) Every normalisation method is a ppn.

(b) If p_1 and p_2 are ppn, then

$$p(v_1, \ldots, v_{n-1}, v_n) \equiv \left\{ \begin{array}{rl} \text{if} & v_n = [\,] \\ \text{then} & p_1(v_1, \ldots, v_{n-1}) \\ \text{else} & \text{if } v_n = [\epsilon | v_n'] \\ & \text{then } p_2[\![^{\beta}/_{\alpha}]\!](v_1, \ldots, v_{n-1}, v_n) \end{array} \right.$$

where α is a primitive method and $\beta \stackrel{s}{=} p(v_1, \ldots, v_{n-1}, v_n')$, is a ppn recursively defined from p_1 and p_2. *For $1 \leq i \leq (n-1)$, if v_i is an essential parameter of p_1 and p_2, then v_i is an essential parameter of p, and if v_i is inessential in both p_1 and p_2, it is inessential in p. v_n is essential in p iff it is essential in p_2.*

(c) Only the objects defined above are ppn.

ii. A ppn p is conditional *if a conditional methodical occurs in p, and* non-conditional *otherwise. The set of conditions C of a plan p is recursively defined as follows.*

(a) If p is non-conditional, then $C = \emptyset$.

(b) If p is conditional and has the form p_1 orelse *p_2,* repeat *p_1 or* try *p_1, and C_1 and C_2 are respectively the sets of conditions of p_1 and p_2, then $C = C_1 \cup C_2$.*

(c) If p has the form if *ψ* then *p_1* else *p_2, and C_1 and C_2 are the sets of conditions of p_1 and p_2, then $C = \{\psi\} \cup C_1 \cup C_2$.*

(d) If p is conditional and is recursively defined from p_1 and p_2, then $C = C_1 \cup C_2$.

[1] None of the proof plans for normalisation informally described in this section has been actually implemented.

iii. An instance of a ppn,

$$\sigma\,(\lambda\phi_1\lambda\phi_2\; p(\phi_1, \mathcal{R}, v_1, \ldots, v_n, \phi_2))$$

where $\sigma = \{^{t_1}/_{v_1}, \ldots, ^{t_n}/_{v_n}\}$ *and* $v_1, \ldots, v_n$ *are the (essential and inessential) parameters of* p*, is a* special-purpose *ppn iff* t_i *is either variable-free or has* ϕ_1 *and/or* $\mathcal{R}$ *as sole variables, for all* $i, 1 \leq i \leq n$.

iv. A ppn is general-purpose *iff it is not special-purpose.*

Example 5.1.1 *The plan* multiple_remove*/1, which removes the occurrences of the symbols listed in its argument,* SymLst*, is recursively defined as*

```
if [SymLst = [ ]
    idmethod
    if [SymLst = [Sym|Lst]
        remove(Sym)
        then multiple_remove(Lst)]]
```

With respect to definition 5.1.1, the component plans p_1 *and* p_2 *respectively correspond to* `idmethod` *and* `remove(Sym) then idmethod`*: in this case, the replacement required to generate* multiple_remove*/1 is* `multiple_remove(Lst)` *for* `idmethod`. □

Definition 5.1.1 is correct with respect to the informal notion of special-purpose plan, since every ground instance of a general-purpose plan is linked to a single normaliser.

Lemma 5.1.1 *Every special-purpose plan specifies a unique normaliser.*

PROOF. (By induction on the length of p)

Let $\sigma(\lambda\phi_1\lambda\phi_2 \quad p(\phi_1, \mathcal{R}, v_1, \ldots, v_n, \phi_2))$ be a special-purpose plan, where $\sigma = \{^{t_1}/_{v_1}, \ldots, ^{t_n}/_{v_n}\}$ and each t_i is either variable-free, or contains ϕ_1 and/or $\mathcal{R}$ as sole variables.

Base case. If σp is either $remove(t_1)$, $stratify(t_1, [t_2, \ldots, t_n])$, $reorder(t_1)$ or any other primitive method, then σp defines a unique normaliser. For instance,

$$\lambda\phi_1\lambda\phi_2\ remove(\phi_1, \mathcal{R}, t_1, \phi_2)$$

represents a rewrite system composed of *remove* rules for the symbol associated with t_1. Since t_1 is either variable-free or contains ϕ and/or $\mathcal{R}$ as sole variables, t_1 is uniquely defined for each input formula ϕ and rewrite set $\mathcal{R}$. The homonymous tactic consists of the application of a finite set of rewrite rules that yields a unique rewritten expression, due to the control imposed on their application.

Step case. Let σp_1 and σp_2 be special-purpose plans that respectively represent normalisers $\mathcal{N}_1$ and $\mathcal{N}_2$.

i. If p has the form p_1 *then* p_2, considering that

$$(p_1\ then\ p_2)(\phi_1, \phi_2) \equiv (\exists\phi_3)(p_1(\phi_1, \phi_3) \wedge p_2(\phi_3, \phi_2))$$

its instance $(\sigma p_1\ then\ \sigma p_2)(\phi_1, \phi_2)$ is equivalent to

$$(\exists\phi_3)(\mathcal{N}_1(\phi_1) = \phi_3 \wedge \mathcal{N}_2(\phi_3) = \phi_2)$$

which can be simplified to $(\mathcal{N}_2 \circ \mathcal{N}_1)(\phi_1) = \phi_2$. Therefore σp_1 *then* σp_2 represents $\mathcal{N}_2 \circ \mathcal{N}_1$.

ii. If p has the form p_1 *orelse* p_2, considering that

$$(p_1\ orelse\ p_2)(\phi_1, \phi_2) \equiv p_1(\phi_1, \phi_2) \vee (p_2(\phi_1, \phi_2) \wedge (\forall\phi_3)\neg p_1(\phi_1, \phi_3))$$

then $(p_1\ orelse\ p_2)(\phi_1, \phi_2)$ is equivalent to

$$\mathcal{N}_1(\phi_1) = \phi_2 \vee (\mathcal{N}_2(\phi_1) = \phi_2 \wedge (\forall\phi_3)(\mathcal{N}_1(\phi_1) \neq \phi_3))$$

which amounts to $(\mathcal{N}_1 \cup \mathcal{N}_2^*)(\phi_1) = \phi_2$, where $\mathcal{N}_2^*$ denotes the restriction of $\mathcal{N}_2$ to the subdomain where $\mathcal{N}_1$ is not defined. Hence σp_1 *orelse* σp_2 denotes the normaliser $\mathcal{N}_1 \cup \mathcal{N}_2^*$.

iii. If p has the form *repeat* p_1, considering that

$$\begin{aligned} \textit{repeat } p_1(\phi_1,\phi_2) \quad \equiv \quad & (\exists\phi_3)(p_1(\phi_1,\phi_3) \wedge \textit{repeat } p_1(\phi_3,\phi_2)) \\ & \vee \\ & \textit{idmethod}(\phi_1,\phi_2) \wedge \neg(\exists\phi_4)p_1(\phi_1,\phi_4) \end{aligned}$$

then *repeat* $p_1(\phi_1,\phi_2)$ is equivalent to

$$\begin{aligned} &(\phi_1 \stackrel{s}{=} \phi_2 \wedge (\forall\psi)\neg p_1(\phi_1,\psi)) \vee (p_1(\phi_1,\phi_2) \wedge (\forall\psi)\neg p_1(\phi_2,\psi)) \vee \\ &\vee (\exists\phi_3)(p_1(\phi_1,\phi_3) \wedge p_1(\phi_3,\phi_2) \wedge (\forall\psi)\neg p_1(\phi_2,\psi)) \vee \ldots \end{aligned}$$

The instance *repeat* $\sigma p_1(\phi_1,\phi_2)$ can be transformed into

$$(\forall\psi)(\mathcal{N}_1(\phi_2) \neq \psi) \wedge \left(\phi_1 \stackrel{s}{=} \phi_2 \vee \mathcal{N}_1(\phi_1) = \phi_2 \vee (\exists\phi_3)(\mathcal{N}_1(\phi_1) = \phi_3 \wedge \mathcal{N}_1(\phi_3) = \phi_2) \vee \ldots\right)$$

and simplified to

$$(\forall\psi)(\mathcal{N}_1(\phi_2) \neq \psi) \wedge \bigvee_{i\in\mathbb{N}} \mathcal{N}_1^i(\phi_1) = \phi_2$$

where $\mathcal{N}^0(\psi) = \psi$. If $\widehat{\mathcal{N}}$ is defined as the restriction of $\mathcal{N}$ to a subdomain S such that, for every element e of this subdomain, $\widehat{\mathcal{N}}(e) \notin S$, then the normaliser associated with *repeat* σp_1 is $\cup_{i\in\mathbb{N}}\widehat{\mathcal{N}}_1^i$.

iv. If p has the form *if* ψ *then* p_1 *else* p_2, for any instance of p where variables are replaced by either constants or terms containing ϕ_1 and/or $\mathcal{R}$ as sole variables, ψ can be evaluated to either $\top$ or $\bot$ for each input conjecture ϕ, in which case p is reduced to either p_1 or p_2. Therefore the normaliser associated with p can be defined from $\mathcal{N}_1$ and $\mathcal{N}_2$.

v. Let p be recursively defined from p_1 and p_2,

$$p(v_1,\ldots,v_{n-1},v_n) \equiv \left\{\begin{array}{ll} \textit{if} & v_n = [\,] \\ \textit{then} & p_1(v_1,\ldots,v_{n-1}) \\ \textit{else} & \textit{if } v_n = [\epsilon|v_n'] \\ & \textit{then } p_2[\![^{\beta}/_{\alpha}]\!](v_1,\ldots,v_{n-1},v_n) \end{array}\right.$$

where α is an atomic method and $\beta \stackrel{s}{=} p(v_1,\ldots,v_{n-1},v_n')$. Since σp has only the input formula and $\mathcal{R}$ as free variables, once they are both instantiated, the corresponding definition of σp admits σp_1 and σp_2 as basic components. Hence σp can be defined from $\mathcal{N}_1$ and $\mathcal{N}_2$. ∎

Variable-free instances of general-purpose plans correspond to intensionally distinct normalisers, i.e. any two distinct instances of a general-purpose plan are linked to distinct sequences of atomic operations. There are cases, however, where all such instances are extensionally identical, as for instance in the case of

$$remove(Sym_1)\ then\ \ldots\ then\ remove(Sym_n)$$

If $S_1, \ldots, S_n$ are symbols for which there is a complete set of *remove* rules w.r.t. a class Σ in which $S_1, \ldots, S_n$ are absent, any instance of the plan above that involves a permutation of these symbols amounts to the same normaliser, when defined strictly in extensional terms (i.e. domain and image).

Given a particular rewrite set $\mathcal{R}$ and a formula ϕ, the computation of the rewritten formula ϕ' by the complete tactic related to a special-purpose plan requires the following steps: the *evaluation of parameters*, in case they are given in terms of ϕ and $\mathcal{R}$, the *evaluation of conditions*, which then leads to non-conditional ground tactics, and the *computation of primitive tactics.*

Example 5.1.2 *The plan built in example 4.2.6,*

```
try remove(bicond)
then try remove(cond)
     then try remove(neg)
          then try stratify(neg,[conj,disj]) then try remove(neg)
               then try stratify(disj,[conj]) then try remove(neg)
                    then try reorder(conj)
                         then try reorder(disj)
```

is an instance of the general-purpose plan

```
if [cpl_rem_sym(SymLst1)
    multiple_remove(SymLst1)]
then if [prt_rem_sym(SymLst2)
         multiple_remove(SymLst2)]
     then if [(prt_rem_sym(SymLst3),
               non_rem_sym(SymLst4),
               append(SymLst3,SymLst4,SymLst5))
              multiple_stratify(SymLst5)
              then multiple_reorder(SymLst4)]
```

where multiple_remove(SymLst) *is defined in example 5.1.1,* multiple_stratify(SymLst) *is defined as*

```
if [SymLst == []
    idmethod
    if [(member(Sym,SymLst),
         delete(Sym,SymLst,SubSymLst),
         str_rul(Sym,SubSymLst))
        try stratify(Sym,SubSymLst)
        then if [prt_rem_sym(OthSymLst)
                 multiple_remove(OthSymLst)]
             then multiple_stratify(SubSymLst)]]
```

and multiple_reorder(SymLst) *as*

```
if [SymLst == []
    idmethod
    if [SymLst = [Sym|Lst]
        try reorder(Sym)
        then multiple_reorder(Lst)]]
```

The essential parameters of the top general plan are SymLst1, ..., SymLst5, *plus the rewrite set,* $\mathcal{R}$, *which is not explicitly indicated. The predicates that make up the conditions of plans and subplans are*

cpl_rem_sym(SymLst)	*iff*	SymLst *are completely removable (i.e. there are complete sets of* remove *rules for every symbol in* SymLst *)*
prt_rem_sym(SymLst)	*iff*	SymLst *are partially removable*
non_rem_sym(SymLst)	*iff*	SymLst *are not removable*
str_rul(Sym,SymLst)	*iff*	*there is a* stratify *rule for Sym w.r.t. every symbol in the list* SymLst

The special-purpose plan of example 4.2.6 is obtained by parameter instantiation and condition evaluation. The rule set of table 4.1 is first supplied as argument to the plan, then the substitution

$$\sigma \;=\; \left\{ {}^{[\equiv,\supset]}/_{SymLst1}, {}^{[\neg]}/_{SymLst2}, {}^{[\neg]}/_{SymLst3}, {}^{[\wedge,\vee]}/_{SymLst4}, {}^{[\neg,\wedge,\vee]}/_{SymLst5} \right\}$$

is applied to it, where $[\equiv, \supset]$ *is the list of completely removable symbols,* $[\neg]$, *of partially removable symbols, and* $[\wedge, \vee]$, *of non-removable symbols*[2]. *The first call of the recursive subplan* multiple_remove *is then equivalent to*

[2] The plan lacks control elements requiring the removal of biconditionals before conditionals, since the removal of the former reintroduce the latter. More elaborate conditions that take this case into account are examined in chapter 9.

```
try remove(bicond)
then try remove(cond)
```

whereas the second call of the same plan can be reduced to

```
try remove(neg)
```

The stratification stage is evaluated to

```
try stratify(neg,[conj,disj])
then try remove(neg)
     then stratify(disj,[conj])
          then try remove(neg)
```

and the reordering stage to

```
try order(conj)
then try order(disj)
```

After the composition of these fragments, the expected special-purpose plan is obtained.

□

The original plan captures at least part of the reasoning that underlay the construction of the resulting special-purpose plan.

i. Every symbol that is absent from the final normalised class and for which there is a complete set of *remove* rules is completely eliminated from the conjecture in the first place.

ii. Rules for partially removable symbols are applied after every successful application of a *remove* or *stratify* rule, since any change of context may turn a previously inapplicable rule into an applicable one.

iii. Stratification is performed pending on the availability of rules for a given symbol w.r.t. all the remaining symbols that occur in an expression. The auxiliary plan, *multiple_stratify*, recursively stratify one symbol over all the others that have not been already moved to lower strata.

iv. *Reorder* rules are left to the end of the process.

The example illustrates the relevance of general-purpose plans for the definition of individual normalisers. Their expressiveness may be further enriched through the recognition of additional patterns present in special-purpose plans.

5.2 Special Purpose Plans

A special-purpose plan defines a normaliser for e.g. a pair of syntactically defined classes. Decision procedures fit into this category, since they have the set of sentences of the underlying language as domain and a pair of propositional constants as image. Procedures based on quantifier elimination in particular are a valuable source of general patterns, some of which absent from the current set of primitive tactics.

5.2.1 Abelian Groups

Hodes' algorithm, a decision procedure for DAG, operates by eliminating quantifiers from the following class of formulae

$$\begin{array}{rcl} prfm & := & fm|(\exists var)prfm|(\forall var)prfm \\ fm & := & atom|\neg fm|fm \vee fm|fm \wedge fm|fm \supset fm|fm \equiv fm \\ atom & := & tm = tm|tm < tm|tm > tm|tm \leq tm|tm \geq tm \\ tm & := & var|rat| - tm|tm + tm|tm - tm|rat \times var \\ var & := & x|y|z|x_1|y_1|z_1|\ldots \\ rat & := & 0|nat^*|rat/nat^*| - rat \\ nat^* & := & 1|2|3|\ldots \end{array}$$

where *prfm* stands for *prefixed formula*[3]. It explores theorem i, according to which a theory T admits quantifier elimination in $\mathcal{L}$ if and only if it admits quantifier elimination for the set of $\mathcal{L}$-formulae of the form

$$(\exists v)(\gamma_1 \wedge \cdots \wedge \gamma_m) \quad (*)$$

where each γ_i is a literal.

[3] The algorithm actually deals with the full class of formulae of $\mathcal{L}_{DAG}$. The above subclass has been chosen to simplify its representation through proof planning. For the original procedure, see [Hodes 71].

This and similar procedures are decomposable into three main segments, one for the reduction of subformulae into expressions of the form $(*)$, another for the elimination of the existential quantifier from $(*)$, and a final one to recognise valid and invalid (in T) quantifier-free sentences. The first two stages have to be successively applied for the complete removal of quantifiers, followed then by the decision procedure for quantifier-free sentences. Assuming that $plan_1$, $plan_2$ and $plan_3$ are the special-purpose plans that respectively model each of these stages, the final plan corresponds to

```
repeat (plan1 then repeat plan2)
then plan3
```

$Plan_1$ includes the following sequence of transformations.

(i) the complete removal of universal quantifiers, biconditionals and conditionals, in this order,

(ii) the stratification of negations with respect to the remaining connectives (conjunctions and disjunctions),

(iii) repeated applications of the *remove* rule for negations,

(iv) the stratification of conjunctions beneath disjunctions, and

(v) the stratification of existential quantifiers under disjunctions.

It is represented as

```
try remove(univer) then try remove(neg)
then try remove(bicond)
     then try remove(cond) then try remove(neg)
          then try stratify(neg,[disj,conj]) then try remove(neg)
               then try stratify(conj,[disj])
                    then try stratify(exist,[disj])
```

which is similar to the plan devised for putting quantifier-free formulae into conjunctive normal form.

The distinctive components of a normaliser for a decidable theory that admits elimination of quantifiers are $plan_2$ and $plan_3$. Both involve rules that deal with the proper symbols of the subjacent language. $Plan_2$ has to put literals into canonical form to simplify the elimination of the innermost existential quantifier. Simplification is accomplished through the removal of $\geq$, $> \leq$ and negations, by means of the additional *remove* rules

$$
\begin{array}{rcl}
v_1 \geq v_2 & \Rightarrow & v_2 \leq v_1 \\
v_1 > v_2 & \Rightarrow & v_2 < v_1 \\
v_1 \leq v_2 & \Rightarrow & (v_1 < v_2) \vee (v_1 = v_2) \\
\neg(v_1 = v_2) & \Rightarrow & (v_1 < v_2) \vee (v_2 < v_1) \\
\neg(v_1 < v_2) & \Rightarrow & (v_1 = v_2) \vee (v_2 < v_1)
\end{array}
$$

Since the removal of $\leq$ and negation has as side effect the reintroduction of disjunctions inside a disjunct, the matrix has to be put back into disjunctive normal form. For efficiency reasons, a new initial plan, $plan'_1$, has been devised to include the removal of these additional symbols from the original class.

```
try remove(univer) then try remove(neg)
then try remove(bicond)
     then try remove(cond) then try remove(neg)
          then try remove(geq)
               then try remove(great)
                    then try remove(leq)
then try stratify(neg,[disj,conj]) then try remove(neg)
     then try stratify(conj,[disj])
          then try stratify(exist,[disj])
```

The removal of negations is performed in three steps: after the removal of universal quantifiers, the removal of conditionals (both of which introduce new occurrences of negations), and the stratification of negations over conjunctions and disjunctions. The stratification step, however, does not render the set of *remove* rules for negation complete for the resulting class: occurrences of negations in the scope of literals can be all eliminated, since atoms are dominated by either equality or $<$, but there may still be other occurrences in the prefix, introduced by the elimination of universal quantifiers. In any case, after the third call of *remove*(*neg*), no occurrence of this symbol is left in the matrix. The transformed class can be built in two stages. A somewhat larger class, defined by the rules,

$$
\begin{array}{rcl}
tfm_0 & := & tfm_1 | \neg tfm_1 | (\exists var) tfm_0 | \neg (\exists var) tfm_0 \\
tfm_1 & := & tfm_2 | tfm_1 \vee tfm_1 \\
tfm_2 & := & tfm_3 | (\exists var) tfm_2 \\
tfm_3 & := & atom' | tfm_3 \wedge tfm_3 \\
atom' & := & tm = tm | tm < tm
\end{array}
$$

where the set of terms is inherited unchanged from the original class, is initially taken into account. It includes inadequate formulae, as for instance those where

(i) negations dominate disjunctions,

$$\neg(x = y \vee z < 2x + 1)$$

(ii) negations dominate conjunctions,

$$\neg(z < 2y \wedge x < -z + 3)$$

(iii) negations dominate atoms,

$$\neg(x = x)$$

(iv) existential quantifiers dominate disjunctions,

$$(\exists x)((\exists y)(x = y) \vee (\exists z)(x < z))$$

To exclude them, additional strata are necessary. The rule

$$tfm'_1 \quad := \quad (\exists var)tfm_2 | tfm'_1 \vee tfm'_1$$

eliminates the undesirable cases involving negations, since all formulae it generates have the existential quantifier as dominant symbol. Concerning the case of existential quantifiers, the rules

$$\begin{array}{rcl} tfm'_0 & := & tfm''_0 | (\exists var)tfm'_0 | \neg(\exists var)tfm'_0 \\ tfm''_0 & := & tfm_2 | \neg tfm'_1 \end{array}$$

generate a subclass that does not contain any formula that presents a disjunction in the immediate scope of a quantifier. The desired transformed class can then be represented as

$$\begin{array}{rcl} fm' & := & tfm_1 | \neg tfm'_1 | tfm'_0 \\ tfm'_0 & := & tfm''_0 | (\exists var)tfm'_0 | \neg(\exists var)tfm'_0 \\ tfm''_0 & := & tfm_2 | \neg tfm'_1 \\ tfm_1 & := & tfm_2 | tfm_1 \vee tfm_1 \\ tfm'_1 & := & (\exists var)tfm_2 | tfm'_1 \vee tfm'_1 \\ tfm_2 & := & tfm_3 | (\exists var)tfm_2 \\ tfm_3 & := & atom' | tfm_3 \wedge tfm_3 \\ atom' & := & tm = tm | tm < tm \end{array}$$

The next stage consists of the elimination of the innermost quantifier from prefixed disjuncts, i.e. formulae of the form

$$(\exists v_1)\ldots(\exists v_n)(\gamma_1 \wedge \cdots \wedge \gamma_m) \quad (*)$$

where γ_i is an atom. All the syntactic operations examined so far were related to constant symbols of the underlying language, including logical, predicate, function and individual constant symbols. In the new stage, however, the operations involve mainly individual variables. Even though the elimination and reorganisation of occurrences of variables are similar to the operations involving constants, the current set of tactics and methods is inadequate for such purposes, for two main reasons. First, unlike constant symbols, individual variables do not admit specific rewrite rules to perform syntactic operations, since free variables in a rule can be instantiated to any term. For instance,

i. No *remove* rule can be restricted to perform the removal of a particular individual variable: whereas $v_1 - v_2 \Rightarrow v_1 + (-v_2)$ is a *remove* rule for binary $-$ only, $v \times 0 \Rightarrow 0$ removes occurrences of $\times$ and any term that substitutes for v, e.g. $x, y, z, \ldots$. Hence, assuming that x (and only x) has to be removed from

$$x \times 0 = y + (z \times 0)$$

this task cannot be accomplished by the tactic *remove*, since the exhaustive application of $v \times 0 \Rightarrow 0$ eliminates occurrences of both x and z.

ii. $nv + mv \Rightarrow (n + m)v$, where $n, m \in \mathbb{N}$, is a *reduce* rule not only for v, but also for any term that substitutes for v. Given the formula

$$(2x + 5x) + (3y + 5y) = z$$

if the simplification to be performed is the reduction of occurrences of x, it would not be achieved by the exhaustive application of the above rule, which would generate $(2 + 5)x + (3 + 5)y = z$.

The second inadequacy is the inability of the current tactics to restrict the application of rewrite rules to chosen subexpressions of a conjecture, such as the matrix in the

Class	Symbols	
	Constants	Variables
Elimination	*remove* *reduce*	*collect*
Reorganisation	*stratify* *reorder*	*attract* *isolate*

Table 5.1: Methods for Normalisation

scope of an existential quantifier. Transformations involving variables, however, must be frequently limited in this way.

To overcome the first limitation, rule variables in the second example above have to be first properly instantiated, and the newly introduced term, x, has to be dealt with as a new individual constant, to prevent further instantiations. As a result, the *exhaustive* application of $nx + mx \Rightarrow (n + m)x$ to the conjecture results in

$$(2 + 5)x + (3y + 5y) = z$$

For variables, therefore, checking a rule for applicability does not guarantee that only the required transformation will be performed. Preventing unnecessary or undesirable applications of a rule is an important feature of controlled rewriting. The above mechanism of controlled variable instantiation is absent from tactics such as *remove*, *reduce*, *stratify* and *reorder*. Finally, for the second limitation, it suffices to add an argument to the tactic, to highlight the target subexpression.

Methods for equation and inequality solving incorporated in the PRESS system satisfy both additional requirements. *Attract* applies rules that reduce the distance between occurrences of a variable inside the symbolic tree that represents the conjecture. *Collect* tries to reduce the number of occurrences of a variable. *Isolate* reorders terms inside an equation (or inequality), placing the assigned variable as sole symbol in one of its sides. As indicated in table 5.1, both *attract* and *isolate* belong to the class of *reorganisation* methods, together with *stratify* and *reorder*. *Collect*, *remove* and *reduce* are *elimination* methods[4].

[4] PRESS is described in [Bundy & Welham 81].

The procedure for the removal of $(\exists v_n)$ from $(*)$ has two components. Equations and inequalities are first reduced to canonical form by means of the *removal* of subtraction, the *stratification* of unary minus over multiplication and sum, the partial *removal* of occurrences of unary minus, the *stratification* of multiplication over sum, and the *reordering* of occurrences of summands. The second component is restricted to operations over variables, starting with the *attraction* of occurrences of v_n inside each atom, through the application of rules[5] such as

$$\begin{array}{rcl}
t_1[v] + (t_2[\not v] + t_3[v]) & \Rightarrow & (t_1[v] + t_3[v]) + t_2 \\
t_1[v] + (t_2[v] + t_3[\not v]) & \Rightarrow & (t_2[v] + t_1[v]) + t_3 \\
(t_1[v] + t_2[\not v]) + t_3[v] & \Rightarrow & (t_1[v] + t_3[v]) + t_2 \\
(t_1[\not v] + t_2[v]) + t_3[v] & \Rightarrow & (t_3[v] + t_2[v]) + t_1
\end{array}$$

For the *collection* of occurrences of v_n, rules

$$\begin{array}{rcl}
k \times v + v & \Rightarrow & (k+1) \times v \\
v + k \times v & \Rightarrow & (k+1) \times v \\
k \times v + (-v) & \Rightarrow & (k + (-1)) \times v \\
(-v) + k \times v & \Rightarrow & (k + (-1)) \times v \\
v + v & \Rightarrow & 2 \times v \\
v + (-v) & \Rightarrow & 0 \times v \\
(-v) + v & \Rightarrow & 0 \times v \\
(-v) + (-v) & \Rightarrow & (-2) \times v \\
k_1 \times v + k_2 \times v & \Rightarrow & (k_1 + k_2) \times v
\end{array}$$

are required. The *isolation* of v_n inside an atom, once the number of occurrences of v_n has been reduced to one in every equation and inequality, needs rules such as

$$\begin{array}{rcl}
t_1 + t[v] = t_2 & \Rightarrow & t[v] = (-t_1) + t_2 \\
(-v) = t & \Rightarrow & v = (-t) \\
0 \times t[v] = u & \Rightarrow & 0 = u \\
t[v] + t_1 < t_2 & \Rightarrow & t[v] < (-t_1) + t_2 \\
t_1 + t[v] < t_2 & \Rightarrow & t[v] < (-t_1) + t_2 \\
(-v) < t & \Rightarrow & (-t) < v \\
0 \times t[v] < u & \Rightarrow & 0 < u
\end{array}$$

as well as conditional rules

$$\begin{array}{lcl}
k_1 \neq k_2 \ \rightharpoonup \ k_1 \times v + t_1 = k_2 \times v + t_2 & \Rightarrow & v = (k_1 - k_2)^{-1} \times t_2 - (k_1 - k_2)^{-1} \times t_1 \\
k_1 = k_2 \ \rightharpoonup \ k_1 \times v + t_1 = k_2 \times v + t_2 & \Rightarrow & t_1 = t_2 \\
k \neq 0 \ \rightharpoonup \ k \times t[v] = u & \Rightarrow & t[v] = (1/k) \times u \\
k > 0 \ \rightharpoonup \ k \times t[v] < u & \Rightarrow & t[v] < (1/k) \times u \\
k < 0 \ \rightharpoonup \ k \times t[v] < u & \Rightarrow & (1/k) \times u < t[v]
\end{array}$$

---

[5] In its current implementation, the preconditions of *attract* include a test which ensures that a variable v occurs in the terms identified in the rules as $t[v]$, and that it does not occur in the others. *Attract* rules therefore represent conditional rules whose conditions remain implicit in the method.

The next step involves the repeated application of a two-stage process, consisting of the exhaustive *attraction* of atoms that contain v_n,

$$\begin{array}{rcl}
(\phi_1[v] \wedge \psi) \wedge \phi_2[v] & \Rightarrow & (\phi_1[v] \wedge \phi_2[v]) \wedge \psi \\
(\psi \wedge \phi_1[v]) \wedge \phi_2[v] & \Rightarrow & (\phi_2[v] \wedge \phi_1[v]) \wedge \psi \\
\phi_1[v] \wedge (\psi \wedge \phi_2[v]) & \Rightarrow & \psi \wedge (\phi_1[v] \wedge \phi_2[v]) \\
\phi_1[v] \wedge (\phi_2[v] \wedge \psi) & \Rightarrow & \psi \wedge (\phi_2[v] \wedge \phi_1[v])
\end{array}$$

and the subsequent *collection* of their occurrences, by means of

$$\begin{array}{rcl}
v = t_1 \wedge v = t_2 & \Rightarrow & v = t_1 \wedge t_1 = t_2 \\
v = t_1 \wedge v < t_2 & \Rightarrow & v = t_1 \wedge t_1 < t_2 \\
v = t_1 \wedge t_2 < v & \Rightarrow & v = t_1 \wedge t_2 < t_1 \\
v < t_2 \wedge v = t_1 & \Rightarrow & v = t_1 \wedge t_1 < t_2 \\
t_2 < v \wedge v = t_1 & \Rightarrow & v = t_1 \wedge t_2 < t_1 \\
v < t_1 \wedge v < t_2 & \Rightarrow & v < min(t_1, t_2) \\
t_1 < v \wedge t_2 < v & \Rightarrow & max(t_1, t_2) < v
\end{array}$$

At this stage, the *partial stratification* of existential quantifiers over conjunctions requires the application of conditional rules,

$$\begin{array}{rcl}
(\exists v)(\phi[v] \wedge \psi[\not v]) & \Rightarrow & \psi \wedge (\exists v)\phi[v] \\
(\exists v)(\psi[\not v] \wedge \phi[v]) & \Rightarrow & \psi \wedge (\exists v)\phi[v]
\end{array}$$

Finally, existential quantifiers are removed by a set of six rules,

$$\begin{array}{rcl}
(\exists v)(v = t[\not v]) & \Rightarrow & \top \\
(\exists v)(v < t[\not v]) & \Rightarrow & \top \\
(\exists v)(t[\not v] < v) & \Rightarrow & \top \\
(\exists v)(t_1[\not v] < v \wedge v < t_2[\not v]) & \Rightarrow & (t_1 < t_2) \\
(\exists v)(v < t_1[\not v] \wedge t_2[\not v] < v) & \Rightarrow & (t_2 < t_1) \\
(\exists v)\psi[\not v] & \Rightarrow & \psi
\end{array}$$

$Plan_2$ is indicated below.

```
try remove(subtr)
then try stratify(unaminus,[times,plus]) then try remove(unaminus)
     then try stratify(times,[plus])
          then reorder(sum)
then try attract(Var,Var:pnat#(_#_))
     then try collect(Var,Var:pnat#(_#_))
          then try isolate(Var,Var:pnat#(_#_))
               then try repeat(attract(Var,Var:pnat#(_#_))
                               then collect(Var,Var:pnat#(_#_)))
then try stratify(exist,[conj])
     then remove(exist)
```

After repeated applications of $plan_2$ to remove all innermost quantifiers, the first plan is recalled to generate new prefixed disjuncts, until the prefix of the formula is empty. Since there are *collect* rules which introduce two new function symbols, max and min, $plan'_1$ has to be modified to include steps for their complete removal as well. The final version for the global plan is

```
repeat (plan''1 then repeat plan2)
then plan3
```

The variable-free formulae of the final transformed quantifier-free class are generated by

$$\begin{array}{rcl} fm'' & := & disj' | fm'' \vee fm'' \\ disj' & := & atom'' | disj' \wedge disj' \\ atom'' & := & tm' = tm' | tm' < tm' \\ tm' & := & cst | tm' + tm' | cst \times tm' \\ cst & := & rat | -rat \\ rat' & := & 0 | nat^* | rat/nat^* \\ nat^* & := & 1|2|3| \ldots \end{array}$$

The validity of these sentences in DAG is established by a decision procedure for the set of closed atoms ($atom''$) and their boolean combinations, which could be also represented as a special-purpose $plan_3$. The complete plan has been implemented in *Clam*, and a few formulae have been supplied to it. Time results are reported in the next example.

Example 5.2.1

i. The formula

$$(\exists x)(x < 0 \wedge 0 < x)$$

can be immediately simplified to the atom $0 < 0$ *by the application of the subplans* plan''_1 *and* plan_2.

ii. The same subplans can convert the formula

$$(\exists x)(\exists y)(\exists z)(((((x < z \wedge x < y) \wedge y < z) \wedge x = y + z) \wedge y = 0) \wedge 0 < x)$$

to $\max(0,0) < 0 \wedge 0 < 0 \wedge 0 = 0 \wedge 0 = 0$ *in 92.1 seconds.*

iii. Given the lemma

$$(k)(l)(mn)(mx)((l \leq mn \wedge 0 < k \wedge mn \leq mx) \supset l < mx + k)$$

which has been derived from a problem described by Boyer and Moore[6]*, the global plan, which includes* $plan_3$*, required 742.1 seconds to reduce it to* $\top$*.* □

The inefficiency of the above special-purpose plan can be illustrated by the fact that, for the third conjecture of example 5.2.1, Hodes' algorithm requires less than one second to identify it as a DAG-theorem[7]. Efficiency, however, as argued in later chapters, is not the primary concern of the proof planning approach to normalisation. Some additional elements may be nevertheless added to the final plan to improve its performance.

i. A numerical evaluator can simplify or evaluate numerical expressions, removing therefore all the occurrences of $+, \times$ and $-_1$ from variable-free terms.

ii. $Plan_3$ could be called after the elimination of each quantifier, to remove closed atomic formulae. In the absence of this step, closed atoms are propagated to the whole conjecture when the stratification of conjunctions over disjunctions takes place, thus increasing the size of rewritten expressions and the number of occurrences of existential quantifiers. For instance, given the conjecture,

$$(\exists x)(\exists y)\neg(\exists z)((2x < y + 3 \wedge y = 2 - x) \vee z < 0)$$

the elimination of $\exists z$ leads to

$$(\exists x)(\exists y)\neg((2x < y + 3 \wedge y = 2 - x) \vee \top)$$

The removal of $\top$ would reduce this formula to $\bot$, otherwise it would be expanded to

$$\begin{array}{c}(\exists x)(\exists y)(y + 3 < 2x \wedge \bot) \vee (\exists x)(\exists y)(y + 3 = 2x \wedge \bot) \\ \vee \\ (\exists x)(\exists y)(y < 2 - x \wedge \bot) \vee (\exists x)(\exists y)(2 - x < y \wedge \bot)\end{array}$$

[6] See [Boyer & Moore 88].

[7] This test has been conducted with the implementation of Hodes' algorithm that is embedded in the linear procedure of the Boyer and Moore prover.

5.2.2 Closed Fields

The theory of *algebraically closed fields* (ACF), formulated in $\mathcal{L}_{ACF} = \{0, 1, -_1, +, \times\}$, also admits quantifier elimination. Besides the usual axioms for commutative fields, an axiom of the form

$$(\exists x)(x^k + x_{k-1}x^{k-1} + \cdots + x_1 x + x_0 = 0)$$

has to be added for each $k \in \mathbb{N}, k > 1$. Although ACF is not negation complete, completeness can be achieved with the inclusion of either an axiom of the form $p = 0$, or all the axioms of the form $p \neq 0$, where p is a prime number[8]. Given a sentence of $\mathcal{L}_{ACF}$, after it is submitted to $plan_1$, described in section 5.2.1, or any suitable variation, the elimination of existential quantifiers from a formula of form

$$(\exists v_1)\ldots(\exists v_n)(\gamma_1 \wedge \cdots \wedge \gamma_m) \quad (*)$$

where γ_i is an atom, requires a rule for *reordering* occurrences of terms inside literals,

$$t_1 = t_2 \;\Rightarrow\; t_1 - t_2 = 0, \quad \text{if } t_2 \overset{s}{\neq} 0$$

The *attraction* and *collection* of negated atoms is achieved by the application of

$$\begin{array}{rcl} (\psi_1[\![\neg\phi]\!] \wedge \delta) \wedge \psi_2[\![\neg\phi]\!] & \Rightarrow & (\psi_1[\![\neg\phi]\!] \wedge \psi_2[\![\neg\phi]\!]) \wedge \delta \\ \psi_2[\![\neg\phi]\!] \wedge (\psi_1[\![\neg\phi]\!] \wedge \delta) & \Rightarrow & \delta \wedge (\psi_1[\![\neg\phi]\!] \wedge \psi_2[\![\neg\phi]\!]), \quad \text{etc} \\ t_1 \neq 0 \wedge t_2 \neq 0 & \Rightarrow & t_1 t_2 \neq 0 \end{array}$$

where $t_1 \neq t_2$ is an abbreviation for $\neg(t_1 = t_2)$. After this step, a matrix originally with m conjuncts is reduced to either

$$t_1 = 0 \wedge t_2 = 0 \wedge \cdots \wedge t_m = 0$$

or

$$t_1 = 0 \wedge t_2 = 0 \wedge \cdots \wedge t_{r-1} = 0 \wedge t_r \neq 0,$$

with $1 \leq r \leq m$. The *attraction, collection* and *reordering* of occurrences of v_n inside each literal converts its left-hand side to the normalised polynomial

[8] The axioms for commutative fields can be found in [Kreisel & Krivine 67b], p. 57.

$$t_s v_n^s + \cdots + t_1 v_n + t_0,$$

such that v_n does not occur in t_i; *stratify* and *reorder* rules might be also necessary at this stage. A *recurrence* rule,

$$\begin{aligned}(\exists v)(t_{s_1} v^{s_1} + t' = 0 \wedge t_{s_2} v^{s_2} + t'' = 0 \wedge \phi) \quad \Rightarrow \quad & [t_{s_2} = 0 \wedge (\exists v)(t_{s_1} v^{s_1} + t' = 0 \wedge t'' = 0 \wedge \phi)] \\ & \vee \\ & [t_{s_2} \neq 0 \wedge (\exists v)(t_{s_2} t' - t_{s_1} t'' v^{s_1 - s_2} = 0 \wedge \\ & \qquad \wedge t_{s_2} v^{s_2} + t'' = 0 \wedge \phi)]\end{aligned}$$

for $s_1 \geq s_2$, then reduces the *rank* of v_n in each system of equations, where

(a) the *degree* of a variable v_n in a polynomial p is the highest power of v_n in p, and

(b) the *rank* for a variable v_n in a system of equations (and inequalities) is the sum of the degrees of v_n in each individual equation and inequality.

The exhaustive application of this rule leaves at most one equation in v_n in the scope of $(\exists v_n)$. The matrix is then reducible to an expression of one of the forms

$$t[v_n] = 0 \wedge \psi[\not v_n]$$

$$t[v_n] \neq 0 \wedge \psi[\not v_n]$$

$$t_1[v_n] = 0 \wedge t_2[v_n] \neq 0 \wedge \psi[\not v_n]$$

The *partial stratification* of existential quantifiers over conjunctions,

$$\begin{aligned}(\exists v)(\phi[v] \wedge \psi[\not v]) \quad &\Rightarrow \quad (\exists v)\phi[v] \wedge \psi \\ (\exists v)(\psi[\not v] \wedge \phi[v]) \quad &\Rightarrow \quad \psi \wedge (\exists v)\phi[v]\end{aligned}$$

is followed by the *removal* of sum by the application of two partial non-dominant rules,

$$\begin{aligned}(\exists v)(t_s v^s + t = 0) \quad &\Rightarrow \quad t_s \neq 0 \vee (t_s = 0 \wedge (\exists v)(t = 0)) \\ (\exists v)(t_s v^s + t \neq 0) \quad &\Rightarrow \quad t_s \neq 0 \vee (t_s = 0 \wedge (\exists v)(t \neq 0))\end{aligned}$$

and the *removal* of existential quantifiers by the application of

$$\begin{aligned}(\exists v)(t_1[v] = 0 \wedge t_2[v] \neq 0) \quad &\Rightarrow \quad \rho \\ (\exists v)(t_s v^s \neq 0) \quad &\Rightarrow \quad t_s \neq 0 \\ (\exists v)(t_s v^s = 0) \quad &\Rightarrow \quad \top \\ (\exists v)\psi[\not v] \quad &\Rightarrow \quad \psi\end{aligned}$$

where ρ is a quantifier-free formula in which v does not occur[9].

[9] The construction of ρ is described in [Kreisel & Krivine 67b], p. 58-9.

The rule for the reduction of the rank of a system of equations and/or inequalities does not seem to fit any of the current classes of rules, since it does not remove symbols, terms or other proper subexpressions, and it is not clear whether any particular stratification or reordering step (for symbols, terms or subexpressions) takes place either. To allow its use under the proof planning approach, one possible option is the introduction of a new method for the *reduction of the complexity of an expression*, measured according to a given parameter, as for instance its rank. The arguments of the new method would have to specify the chosen measure for each case. The new method would then be required in the implementation of $plan_2$ for ACF.

5.3 General Purpose Plans

The special-purpose plans that represent decision procedures for DAG and ACF confirmed the relevance of *remove*, *stratify* and *reorder*, as well as the PRESS methods, in normalisation. The study of the inner structure of both plans may supply additional elements for the definition of more abstract plans, from which new normalisers are generated by parameter instantiation. Particularly important for theorem proving is the representation of generic decision procedures and reduction functions.

5.3.1 Decision Procedures

Although quantifier elimination could, in principle, be recognised as the deep structure of Hodes' algorithm, this procedure is actually an instance of a mechanism that encompasses an even larger rewrite-based family of normalisers. Since any decision procedure for a (negation-complete) theory reduces a sentence of the subjacent language into a propositional constant, *the axis of rewriting is the removal of logical, predicate and function symbols.* All the remaining operations, such as the *stratification* and *reordering* of occurrences of symbols, have a subsidiary role with respect to the application of partial *remove* rules. Under this viewpoint, the decision procedure for DAG is a mechanism that gradually *removes* all the logical $(\neg, \vee, \wedge, \supset, \equiv, \forall, \exists)$, predicate $(=, <, \leq, >, \geq)$ and function symbols $(-_1, +, -_2, \times)$ from any sentence of $\mathcal{L}_{DAG}$. The removal of individual variables and constants happens as a side effect, since they occur only in the scope of function or predicate symbols.

A complete set of *remove* rules for each of these symbols would suffice for the representation of the decision procedure. Such rules are explicitly available, however, only for $\equiv, \supset, \forall, \geq, >, \leq$ and binary $-$,

$$\begin{array}{rcl}
\phi \equiv \psi & \Rightarrow & (\neg\phi \vee \psi) \wedge (\neg\psi \vee \phi) \\
\phi \supset \psi & \Rightarrow & \neg\phi \vee \psi \\
(\forall v)\phi & \Rightarrow & \neg(\exists v)\neg\phi \\
t_1 \geq t_2 & \Rightarrow & t_1 = t_2 \vee t_2 < t_1 \\
t_1 > t_2 & \Rightarrow & t_2 < t_1 \\
t_1 \leq t_2 & \Rightarrow & t_1 = t_2 \vee t_1 < t_2 \\
t_1 - t_2 & \Rightarrow & t_1 + (-t_2)
\end{array}$$

which, as indicated in the rhs expressions of the above rules, are eliminable in terms of some of the remaining symbols ($\neg, \exists, \wedge, \vee, <, =, +, \times, -_1$, min and max). The application of complete sets of *remove* rules at the beginning of rewriting allows the following stages to be entirely concentrated on partially removable symbols, if none of the partial rules contains totally removable symbols: when this is not the case, total rules have to be applied to partial rules before rewriting starts.

In the absence of complete sets of *remove* rules, the simplest rewriting strategy consists of the application of partial rules, provided that, when no rule is applicable, there is a mechanism to change the context in which the symbol to be removed occurs. A strategy of *context transformation* minimises search when based on *backwards* reasoning (i.e. from the set of propositional constants to the initial set of formulae). The partial *remove* rules that have a propositional constant as left-hand side expression are those for existential quantifiers,

$$\begin{array}{rcl}
(\exists v)(v = t[\not v]) & \Rightarrow & \top \\
(\exists v)(v < t[\not v]) & \Rightarrow & \top \\
(\exists v)(t[\not v] < v) & \Rightarrow & \top \\
(\exists v)\psi & \Rightarrow & \psi
\end{array}$$

for conjunctions,

$$\begin{array}{rcl}
\phi \wedge \bot & \Rightarrow & \bot \\
\psi \wedge \top & \Rightarrow & \psi \\
\bot \wedge \phi & \Rightarrow & \bot \\
\top \wedge \psi & \Rightarrow & \psi
\end{array}$$

for disjunctions,

$$\begin{array}{lcl} \psi \vee \bot & \Rightarrow & \psi \\ \phi \vee \top & \Rightarrow & \top \\ \bot \vee \psi & \Rightarrow & \psi \\ \top \vee \phi & \Rightarrow & \top \end{array}$$

for negations,

$$\begin{array}{lcl} \neg\top & \Rightarrow & \bot \\ \neg\bot & \Rightarrow & \top \end{array}$$

and for predicate symbols,

$$\begin{array}{lcl} (0 < 0) & \Rightarrow & \bot \\ (0 = 0) & \Rightarrow & \top \end{array}$$

where ψ is a propositional constant and ϕ is a formula of $\mathcal{L}_{DAG}$. As already mentioned, the application of a partial rule for a symbol may have as side effect the removal of others as well: for instance,

$$(\exists v)(v = t[\not{v}]) \quad \Rightarrow \quad \top$$

removes both existential quantifier and equality. This ambiguity is harmless, since in the end all symbols have to be eliminated. Before partial rules are applied, symbols usually have to be ordered to avoid the risk of their reintroduction in the course of rewriting. The above rules introduce only propositional constants, hence any order is acceptable.

For existential quantifiers, since the lhs expressions of the corresponding rules are dominated by the quantifier, and the subexpressions it dominates contain either $=$ or $<$ as main symbol, all the other logical symbols still possibly present in a conjecture $(\neg, \wedge, \vee)$ must be first removed from the scope of existential quantifiers[10]. If negation is chosen in the first place, since there are no stratification rules relating occurrences of

[10] Function symbols still present in the current subclass of formulae — $+, \times, -_1$, min and max — are allowed in the atoms dominated by the quantifier, since the lhs expression of all rules contains a variable over terms. To determine which symbols may still occur in the scope of a partially removable symbol, a normalisation plan has to take into account that

(a) if all partial rules contain variables over formulae, any symbol can occur in the expression to be rewritten,

(b) if only variables over atomic formulae are present, only predicate and function symbols are allowed in the expression, whereas

(c) variables over terms allow only the occurrence of function and individual constant symbols.

existential quantifiers to occurrences of negations, it is necessary to examine the rules for their partial removal. Rules

$$\begin{array}{rcl} \neg(t_1 = t_2) & \Rightarrow & ((t_1 < t_2) \vee (t_2 < t_1)) \\ \neg(t_1 < t_2) & \Rightarrow & ((t_1 = t_2) \vee (t_2 < t_1)) \\ \neg(\neg\phi) & \Rightarrow & \phi \end{array}$$

allow only three logical and predicate symbols in the immediate scope of a negation: negation itself, $=$ and $<$. The remaining logical and predicate symbols that could occur in the scope of a quantifier, $\wedge$ and $\vee$, have to be removed from the scope of negations. This operation calls for a set of suitable total *stratify* rules, which are in fact available. Their application results in a rewritten expression having negations applied just to either atomic or negated formulae, for which partial *remove* rules are also available. As a result, the removal of negations is accomplishable.

Two undesirable logical symbols, disjunctions and conjunctions, still occur in the scope of existential quantifiers. For the first one, there are also suitable total *stratify* rules, but their application does not suffice for the complete removal of disjunctions, since occurrences of this symbol may take place in the scope of conjunctions. The stratification of conjunctions over disjunctions is then required; after that, the stratification of existential quantifiers over disjunctions is completed[11]. For the second symbol, there are only *partial* stratification rules,

$$\begin{array}{rcl} (\exists v)(\phi[v] \wedge \psi[\not v]) & \Rightarrow & \psi \wedge (\exists v)\phi[v] \\ (\exists v)(\psi[\not v] \wedge \phi[v]) & \Rightarrow & \psi \wedge (\exists v)\phi[v] \end{array}$$

The restriction for their application, however, is not related to logical, predicate or function symbols, but to occurrences of *variables* in the context of atomic formulae, i.e. equations and inequalities. This case requires an equational procedure based on methods such as *attract, collect* and *isolate.*

The removal of all undesirable logical symbols from the scope of existential quantifiers has been achieved. Quantifier elimination requires a final test to determine, for instance, that v does not occur in t in the application of the rule

[11] It is clear at this stage that a mechanism for checking occurrences of a symbol S inside an expression *after* the application of a partial *remove* rule for S is also necessary. In the event S still occurs in the expression, and stratification rules are not applicable any more, there are two possible actions: the removal of symbols that obstruct the application of a *remove* rule for S, or the stratification of occurrences of the remaining symbols over S.

$$(\exists v)(v = t[\not p]) \quad \Rightarrow \quad \top$$

Another equational plan is required to handle those cases with undesirable occurrences of variables in equations and inequalities. Once equational transformations are completed, *remove* rules for existential quantifiers are finally employed. A similar reasoning applies to $\wedge, \vee, <, =$ and $\neg$, the remaining symbols for which *remove* rules with propositional constants as rhs expressions are available. After their removal, since no other logical or predicate symbol is left, the final expression must be a propositional constant.

Even though the above description is strictly linked to Hodes' algorithm, it indicates nonetheless the main elements for the construction of a general-purpose plan for the representation of any rewrite-based decision procedure. New guidelines for the selection of relevant rules have to be included in each stage. As already suggested, complete sets of *remove* rules should precede partial *remove* rules. A mechanism of *context transformation* could be based on the following principles.

i. Total rules have priority over partial rules, since the former do not involve additional rewriting, unless they introduce additional deviant symbols, in which case a partial rule may represent a less complex choice.

ii. The preference for *total* over *partial* rules has to be qualified in the presence of *stratify*, since stratification tends to dramatically increase the size of the rewritten expression. Partial *remove* may therefore generate a less complex process than a total *stratify* rule.

iii. Whenever more than one type of total (partial) rule is applicable, *remove* should have precedence over *stratify*, and *stratify* over *reorder*.

iv. When an order for the removal of a set of symbols fails to be accomplished, alternative orderings may still achieve the desired goal, and must therefore be tested.

5.3.2 Extension of Decidable Classes

The implementation of a general-purpose plan that captures the structure of rewrite-based decision procedures is no simple task. As informally described in the previous section, not only the number of operations is very large, but also several additional mechanisms for the selection of methods and rules are needed at each phase of the process. The task can be nonetheless broken into simpler components that involve intermediate classes of formulae. In the particular case of Hodes' algorithm, one of the intermediate subclasses is implicitly defined by the set of terminal *remove* rules, i.e. those that have a propositional constant as rhs expression.

$$
\begin{array}{rcl}
fm & := & \neg fm | fm \wedge fm | fm \vee fm \\
fm' & := & atom | (\exists x_i) atom | (\exists x_i)(x_i = tm_i) | (\exists x_i)(x_i < tm_i) | (\exists x_i)(tm_i < x_i) \\
atom & := & \top | \bot | rat = rat | rat < rat \\
tm_i & := & var_i | rat | - tm | tm + tm | tm - tm | rat \times var_i \ \text{(for each } i \in \mathbb{N}^*) \\
var_i & := & x_1 | \ldots | x_{i-1} | x_{i+1} | \ldots \ \text{(for each } i \in \mathbb{N}^*) \\
var & := & x_1 | x_2 | x_3 | \ldots \\
rat & := & 0 | nat^* | rat/nat^* \\
nat^* & := & 1 | 2 | 3 | \ldots
\end{array}
$$

The remaining rules available to the procedure have to transform each formula of $\mathcal{L}_{DAG}$ into an element of the above class. As a result, the first component of the mechanism is an instance of the standard strategy of *subclass reduction* into a decidable domain[12]. *Remove* loses its prominence over the other methods, and the new plan has to take into account the structures of both initial and intermediate classes, to determine

i. the symbols of the initial class which are absent from the intermediate class, and the necessary *remove* rules for them.

ii. the distribution of the remaining symbols amongst the strata in each class, and the *stratify* rules required to make the symbol distribution pattern of the initial class match that of the intermediate class.

iii. the relative position of symbols inside each stratum of the initial and intermediate class, so that reorganisation rules could be employed until each one matches against the other[13].

---

[12] See section 2.1.

[13] Although only three basic methods have been taken into account, it seems that the main arguments above can be extended to other methods as well.

Since there may be more than one candidate intermediate subclass, there must also be guidelines for the choice of the most suitable one. If the intermediate classes $\Sigma_1, \ldots, \Sigma_n$ are partially ordered according to their sets of logical, predicate and function symbols, the closer a decidable class Σ_i is to the initial class, the simpler the rewriting process to reach it. Measures other than the simple comparison of alphabets can be also defined, as long as they effectively reflect the complexity of the rewriting process required to reach a subclass. A *weight* assigned to each symbol would take into account

(a) the number of rewriting steps necessary to remove the symbol from an expression, and

(b) the possible introduction of deviant symbols by the corresponding *remove* rules.

For instance, if a total *remove* rule for S does not introduce deviant symbols, the weight of S should be minimal, whereas partially removable symbols should be associated with higher weights. The decidable class with highest global weight is (at least in principle) the easiest accessible from the initial set of formulae, since it requires the least number of rewriting steps to be obtained.

A possible application of this normalisation strategy takes place in *PA* (Peano arithmetic), which admits two subclasses, respectively linked to *PrA* (Presburger arithmetic) and *SMA* (strictly multiplicative arithmetic), both relevant from the point of view of formula reduction processes. Since *PrA* is decidable and negation complete, *PA* is one of its conservative extensions. Hence, as established in theorem C.4.3, the set of formulae of $\mathcal{L}_{PrA}$, defined in BNF as

$$\begin{array}{rcl} fm' & := & atom' | \neg fm' | fm' \vee fm' | fm' \wedge fm' | fm' \supset fm' | fm' \equiv fm' | (\forall var) fm' | (\exists var) fm' \\ atom' & := & tm' = tm' \\ tm' & := & var | cst | \mathrm{s}(tm') | tm' + tm' \\ cst & := & 0 | 1 \\ var & := & x | y | z | \ldots \end{array}$$

represents a decidable subclass of type II for *PA*. Similar remarks apply to *SMA* and $\mathcal{L}_{SMA}$, whose formulae are generated by the productions

$$\begin{array}{rcl} fm'' & := & atom'' | \neg fm'' | fm'' \vee fm'' | fm'' \wedge fm'' | fm'' \supset fm'' | fm'' \equiv fm'' | (\forall var) fm'' | (\exists var) fm'' \\ atom'' & := & tm'' = tm'' \\ tm'' & := & var | cst | tm'' \times tm'' \\ cst & := & 0 | 1 \\ var & := & x | y | z | \ldots \end{array}$$

Given that the set of formulae of $\mathcal{L}_{PA}$ corresponds to

$$
\begin{array}{rcl}
fm & := & atom|\neg fm|fm \vee fm|fm \wedge fm|fm \supset fm|fm \equiv fm|(\forall var)fm|(\exists var)fm \\
atom & := & tm = tm \\
tm & := & var|cst|\mathrm{s}(tm)|tm + tm|tm \times tm \\
cst & := & 0|1 \\
var & := & x|y|z|\ldots
\end{array}
$$

the comparison of the structure of all three classes shows that $\mathcal{L}_{PrA}$ and $\mathcal{L}_{SMA}$ are both strictly definable in terms of $\mathcal{L}_{PA}$, once a group of *absent symbols* is identified, i.e.

$$
\begin{array}{rcl}
Fml_{\mathcal{L}_{PrA}} & = & Fml_{\mathcal{L}_{PA}} - \{\phi | \times \text{ occurs in } \phi\} \\
Fml_{\mathcal{L}_{SMA}} & = & Fml_{\mathcal{L}_{PA}} - \{\phi | \mathrm{s} \text{ and/or } + \text{ occur in } \phi\}
\end{array}
$$

The reduction of formulae from the initial into any of these decidable classes can be described in terms of the *removal* of occurrences of deviant symbols, $\{\times\}$ or $\{\mathrm{s}, +\}$, from elements of $\mathcal{L}_{PA}$. A simplified version of a general-purpose plan, *decide*, that supervises this transformation, is described below,

```
if   [(member(Cla),
       decidable(Cla,Thr))
      dec_pro(Cla,Thr)
      if [min_dec_cla(DecCla,DevSymLst,Thr)
          multiple_remove(DevSymLst)
          then dec_pro(DecCla,Thr)]]
```

where

decidable(DecCla, Thr)	iff	*DecCla* is a decidable subclass for the theory *Thr*
min_dec_cla(DecCla, DevSymLst, Thr)	iff	*DecCla* is the easiest accessible *Thr*-decidable class for the current conjecture, and *DevSymLst* are the deviant symbols of the language of *Thr* w.r.t. *DecCla*

and *dec_pro(DecCla, Thr)* is a special-purpose plan that represents a decision procedure for *Thr* w.r.t. *DecCla*. *Decide* checks whether a conjecture belongs to a decidable class, and, in the affirmative case, supplies it to a suitable decision procedure. Otherwise it identifies the most easily reachable decidable class, *DecCla*, and the list of symbols *DevSymLst* that should be removed from the conjecture, to convert it into an element of this class. For this purpose, *remove* is repeatedly called by the subplan *multiple_remove*

which, when successful, delivers the transformed conjecture to a decision procedure for *DecCla*.

The above simplified description, however, does not indicate, for a given conjecture, how the easiest reachable decidable class is recognised, how deviant symbols are ordered before their removal, or how *remove* rules are selected when there are several available.

5.4 Conclusion

The representation of two decision procedures based on quantifier elimination by means of special-purpose plans stressed the relevance of primitive methods such as *remove*, *stratify* and *reorder* for normalisation, and revealed the need for additional methods to represent the equational components of the procedures. The analysis of one of the plans provided the necessary elements for the construction of two generic general-purpose plans from which families of normalisers can be derived.

The first general-purpose plan has been designed to capture patterns common to all rewrite-based decision procedures. It is built around the *remove* method, and provides an instrument for the development of normalisers described as $\langle Stn_{\mathcal{L}}, \{\top, \bot\}\rangle$. Since its informal description suggests a highly complex structure, a simpler plan has been obtained to represent individual stages of the process, identifiable by means of intermediate syntactically defined subclasses of formulae. The second plan, therefore, is suitable for the development of normalisers described by means of pairs of classes, $\langle \Sigma_1, \Sigma_2\rangle$, and is important for the study of the extension of decidable subclasses.

A simplified general-purpose proof plan built according to the second guideline can model, for instance, reduction processes based on the *removal of deviant symbols* w.r.t. a reduction class. Theories such as Peano arithmetic, a conservative extension of two of its decidable subtheories, are potential domains for applications. Before more elaborate plans can be built, however, other properties of the process of decidable sublanguage extension must be investigated, to allow the definition of more efficient strategies concerning the ordering of decidable classes, deviant symbols and *remove* rules.

Chapter 6

Extension of Decidable Sublanguages

Normalisation tactics and methods are built from a combination of elementary syntactic operators. The classification of rewrite rules along guidelines set out by proof planning represents the first fragment of a strategy for the control of rewriting processes in general. Additional control constructs may be provided for each particular normalisation task, such as the extension of decidable sublanguages.

Section 6.1 presents theoretical results according to which the extension of decidable sublanguages is reducible to the application of *remove* rules. Section 6.2 describes control elements for noetherian sets of *remove* rules that do not affect the deductive strength of the system. Heuristic measure functions for the complexity of rewriting are described in section 6.3.

6.1 Decidable Sublanguages

When a decidable subclass for a theory T in $\mathcal{L}$ coincides with the set of formulae of a sublanguage $\mathcal{L}'$, $\mathcal{L}'$ is a *decidable sublanguage* for T. Their extensions can be generated by the strict application of *remove* rules for deviant symbols of $\mathcal{L}$ with respect to $\mathcal{L}'$, i.e. non-logical symbols of $\mathcal{L}$ that are absent from $\mathcal{L}'$. A proof for this result first requires establishing that the reduction of formulae to decidable sublanguages through rewriting must include *remove* rules for deviant symbols[1].

[1] Decidable sublanguages are discussed in appendix C.4.3.

Lemma 6.1.1 *Let $\mathcal{R}$ be a rewrite set. Given the expressions $\delta[\![S]\!]$ and $\epsilon[\![\not S]\!]$, if there is a rewriting sequence of the form*

$$\delta_0 \overset{R_1}{\Rightarrow} \delta_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_{n-1}}{\Rightarrow} \delta_{n-1} \overset{R_n}{\Rightarrow} \delta_n$$

where $R_i \in \mathcal{R}$, $\delta_0 \overset{s}{=} \delta$ and $\delta_n \overset{s}{=} \epsilon$, then there is a pair $\langle \delta_j, \delta_{j+1} \rangle$, $0 \leq j < n$, such that S occurs only in δ_j.

PROOF. (By induction on the length of the rewriting sequence)

Base case. Trivial, for the sequence is reduced to

$$\delta \overset{R_1}{\Rightarrow} \epsilon$$

and, by hypothesis δ contains S, whereas ϵ does not.

Step case. Assume that, for any n-step rewriting sequence involving an initial expression $\widehat{\delta}$ that contains S and a final expression $\widehat{\epsilon}$ free from occurrences of that symbol, there is a pair of contiguous expressions, $\langle \widehat{\delta_i}, \widehat{\delta_{i+1}} \rangle$, such that S occurs only in $\widehat{\delta_i}$. Let δ and ϵ be expressions for which the rewritten sequence has $n+1$ steps, i.e.

$$\delta \overset{R_1}{\Rightarrow} \delta_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_n}{\Rightarrow} \delta_n \overset{R_{n+1}}{\Rightarrow} \epsilon$$

Two possible cases may take place. If δ_n contains S, then $\langle \delta_n, \epsilon \rangle$ is the sought pair. Otherwise, according to the induction hypothesis, there must be a pair of rewritten expressions $\langle \delta_i, \delta_{i+1} \rangle$, $1 \leq i < n$, that satisfies the mentioned condition, since the rewriting sequence that links δ to δ_n has length n. ∎

Lemma 6.1.2 *Let*

(i) *T be an undecidable theory in $\mathcal{L}$,*

(ii) *$\mathcal{L}'$ be a decidable sublanguage for T,*

(iii) *$\mathcal{R}$ be a set of T-valid rewrite rules of the form*

$$\delta(v_1, \ldots, v_n) \Rightarrow \delta'(v_1, \ldots, v_n)$$

(i.e. the free variables that occur in the rhs expression of each rule are exactly those that also occur in its lhs expression), and

(iv) Σ *be the* $\mathcal{R}$*-extension of* $Fml_{\mathcal{L}'}$.

If Σ *is a proper extension, then* $\mathcal{R}$ *contains a* remove *rule for a* $\mathcal{L}$*-deviant symbol (w.r.t.* $\mathcal{L}'$*).*

PROOF. Let ϕ be a formula of $\Sigma - Fml_{\mathcal{L}'}$. As $\phi \notin Fml_{\mathcal{L}'}$, there must be an occurrence of a $\mathcal{L}$-deviant symbol S in ϕ. Also, since Σ is the $\mathcal{R}$-extension of $Fml_{\mathcal{L}'}$, there is a formula $\phi_n \in Fml_{\mathcal{L}'}$ and a finite rewriting sequence such that

$$\phi_0 \overset{R_1}{\Rightarrow} \phi_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_{n-1}}{\Rightarrow} \phi_{n-1} \overset{R_n}{\Rightarrow} \phi_n$$

where $\phi \overset{s}{=} \phi_0$. According to lemma 6.1.1, there must be contiguous formulae ϕ_i and ϕ_{i+1} in this sequence such that ϕ_i contains S and ϕ_{i+1} does not[2]. Since ϕ_{i+1} is generated from ϕ_i by the application of R_{i+1}. $\delta \Rightarrow \delta'$, there is a substitution σ such that ϕ_i and ϕ_{i+1} have respectively the forms $\psi[\![\sigma\delta]\!]$ and $\psi[\![\sigma\delta']\!]$. Considering that S occurs in ϕ_i but not in ϕ_{i+1}, there must be an occurrence of this symbol either in σ or in δ.

If S occurs in σ, there is a pair $^{t_i}/_{v_i} \in \sigma$ such that S occurs in t_i. Since by hypothesis each variable that occurs in the lhs expression of R_{i+1} also occurs in its rhs expression, then S also occurs in $\sigma\delta'$ and ϕ_{i+1}, thus contradicting the assumption that ϕ_{i+1} does not contain S. Hence S must occur in δ. Considering that it cannot occur in δ' (for otherwise it would also occur in ϕ_{i+1}), R_{i+1} is a *remove* rule for S. ∎

Even though extensions of decidable sublanguages require the existence of *remove* rules for deviant symbols, the existence of such rules does not guarantee that the corresponding extension is non-trivial. Proper extensions need that the rhs expression of at least one *remove* rule is a well-formed expression of the original sublanguage.

---

[2] *General remove rules* are described in chapter 11.

Example 6.1.1 *Let $\mathcal{L} = \{S_1, \ldots, S_{10}\}$ be a language, and let $\mathcal{L}' = \{S_8, S_9, S_{10}\}$ be one of its sublanguages. Given the* remove *rules*

$$\begin{array}{rcl} S_1(v_1, S_9) & \Rightarrow & S_2(v_1, v_1) \\ S_2(S_9, S_7) & \Rightarrow & S_4(S_9) \\ S_3(v_1, v_2, v_2) & \Rightarrow & S_2(v_2, v_2) \end{array}$$

although they refer to symbols that are deviant w.r.t. $\mathcal{L}'$, their application to formulae of $\mathcal{L}$ does not lead to a proper extension of $Fml_{\mathcal{L}'}$, considering that the rhs expressions of all three rules contain deviant symbols as well. □

Lemma 6.1.2 is required in the proof that the reduction of a formula to a decidable sublanguage can be limited to the application of *remove* rules for deviant symbols.

Lemma 6.1.3 *Let T be an undecidable theory in $\mathcal{L}$, $\mathcal{L}'$ be a decidable sublanguage for T and $\mathcal{R}$ be a set of T-valid rewrite rules of the form $\delta(v_1, \ldots, v_n) \Rightarrow \delta'(v_1, \ldots, v_n)$. If Σ is the $\mathcal{R}$-extension of $\mathcal{L}'$, then, for any $\phi \in (\Sigma - Fml_{\mathcal{L}'})$, there are T-valid rules $R_1, \ldots, R_m$, not necessarily in $\mathcal{R}$, and a formula $\psi \in Fml_{\mathcal{L}'}$ such that each R_i is a* remove *rule for one or more deviant symbols of $\mathcal{L}$ w.r.t. $\mathcal{L}'$, and*

$$\phi \overset{R_1}{\Rightarrow} \phi_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_m}{\Rightarrow} \psi$$

PROOF. Let ϕ be a formula of $Fml_{\mathcal{L}} - Fml_{\mathcal{L}'}$ whose reduction to a formula ψ of the sublanguage $\mathcal{L}'$ involves the rewrite sequence

$$\phi \overset{R_1}{\Rightarrow} \phi_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_{n-1}}{\Rightarrow} \phi_{n-1} \overset{R_n}{\Rightarrow} \psi$$

According to lemma 6.1.2, the sequence must contain $R_{i_1}, \ldots, R_{i_m}$, $1 \leq m \leq n$, which are *remove* rules for $\mathcal{L}'$-deviant symbols. It can then be transformed through the iteration of the following procedure: given the subsequence

$$\phi_i \overset{R_{i+1}}{\Rightarrow} \phi_{i+1} \overset{R_{i+2}}{\Rightarrow} \phi_{i+2}$$

if either R_{i+1} or R_{i+2} (but not both) is a *remove* rule for a deviant symbol, then the subsequence is replaced with $\phi_i \overset{R'_{i+1}}{\Rightarrow} \phi_{i+2}$, where R'_{i+1} is the most general rule derived

from R_{i+1} and R_{i+2}. Otherwise the subsequence is left unchanged. The exhaustive application of the procedure leads to

$$\phi \overset{R'_{i_1}}{\Rightarrow} \phi'_1 \overset{R'_{i_2}}{\Rightarrow} \ldots \overset{R_{i_{m-1}}}{\Rightarrow} \phi'_{m-1} \overset{R'_{i_m}}{\Rightarrow} \psi$$

where each R'_{i_k} is a derived *remove* rules for one or more deviant symbols. ∎

Lemma 6.1.3 has important implications for the proof planning approach to normalisation. As described in section 5.3.2, the reduction of a formula into a decidable sublanguage involves two tasks, the *removal of deviant symbols* and, when necessary, the *elimination of fatal disagreements* between formulae and *remove* rules. The second task involves operations other than the plain removal of symbols, whose modeling therefore requires more complex plans. The above lemma, however, guarantees that sublanguages can be extended by the strict application of *remove* rules to a conjecture, and indicates that disagreements may be dealt with strictly at the rule level. As a result, a generic plan to control the extension of decidable sublanguages can be entirely developed upon the method *remove* or any of its variations.

6.2 Basic Control

A rewrite set $\mathcal{R}$ controlled by proof plans is limited in the first place to the exhaustive application of certain classes of rules, such as *remove* rules, according to a preestablished order, similarly to the case of normal or deterministic Markov algorithms[3]. Effectiveness and improved efficiency, however, usually demand stronger control features. In the context of decidable sublanguage extension, at least two of the new control elements, based on properties of the rewritten formula and $\mathcal{R}$, do not jeopardise completeness with respect to exhaustive rewriting.

The first control guideline consists of checking, after each rewriting step, whether the resulting expression belongs to a decidable sublanguage, in which case rewriting halts, independently of the applicability of other rules. This restriction does not prevent any reducible formula from being identified: if ϕ can be transformed into an element of the

[3] See [Sommerhalder & Van Westrhenen 88], p. 266.

sublanguage by exhaustive rewriting, it can be transformed by restricted rewriting as well.

The second guideline is valid only for confluent and noetherian sets of *remove* rules. For these systems, there is no need to consider more than a single path of the rewriting tree, since any of them leads to the same normalised expression. In the event of failure for a path (i.e. when the normal form of a conjecture does not belong to a decidable sublanguage), no other has to be examined. Additional restrictions can be imposed on non-confluent systems, depending on the properties of the rewrite set.

6.2.1 Non-confluent Sets

When $\mathcal{R}$ is noetherian but non-confluent, priority can be given to the application of total over partial *remove* rules, without loss of completeness w.r.t. exhaustive rewriting. Provided that certain conditions are met, rewriting is decomposable into two independent stages, the first of which has only total rules. One of the conditions involves the notion of *normalisable* set of rules.

Definition 6.2.1 (Antecedent Relation for Rules)

Let $\mathcal{R} = \{R_1, \ldots, R_n\}$, *where* R_i *has the form* $\delta_{i,1} \Rightarrow \delta_{i,2}$, *be a set of* remove *rules for a set of symbols* $\mathcal{S}$ (*i.e. for each* $S_j \in \mathcal{S}$, *there is at least one* remove *rule in* $\mathcal{R}$).

i. $\mathcal{R}$ *is* normalised *w.r.t.* $\mathcal{S}$ *iff, for every rule* $R_i \in \mathcal{R}$, $\delta_{i,2}$ *does not contain any symbol of* $\mathcal{S}$.

ii. $\mathcal{R}$ *is* normalisable *w.r.t.* $\mathcal{S}$ *iff there is a set*

$$\mathcal{R}' \quad = \quad \left\{ (R'_i.\ \delta_{i,1} \Rightarrow \delta'_{i,2}) \mid \delta_{i,2} \stackrel{\mathcal{R}}{\Rightarrow} \delta'_{i,2}, 1 \leq i \leq n \right\}$$

where $\mathcal{R}'$ *is a set of* remove *rules for* $\mathcal{S}$ *that is normalised w.r.t.* $\mathcal{S}$.

iii. If $\mathcal{R}_i$ *and* $\mathcal{R}_j$ *are total* remove *rules respectively for* S_i *and* S_j, R_i antecedes R_j, *or* $R_i \sqsubset R_j$, *iff* S_j *occurs in the rhs expression of* R_i.

iv. $\sqsubseteq$ *is the reflexive and transitive closure of* $\sqsubset$.

Example 6.2.1

i. Given the rules,

$$\begin{array}{lrcl} R_1. & f(v_1, v_2) & \Rightarrow & k(v_1, v_1, l(v_2)) \\ R_2. & g(v_1, v_1) & \Rightarrow & l(v_1) \\ R_3. & g(v_1, h(v_1, v_2)) & \Rightarrow & m(v_1, l(v_2), v_2) \\ R_4. & h(g(v_1, v_2), g(v_2, v_1)) & \Rightarrow & l(k(v_1, v_2, v_2)) \\ R_5. & m(v_1, l(v_2), v_3) & \Rightarrow & k(v_3, v_1, v_2) \end{array}$$

R_1 is a remove *rule for f, R_2, for g, R_3, for g and h, R_4, also for g and h, and R_5, for m and l. R_1 is the only total rule of the group.*

(a) $\mathcal{R} = \{R_1, R_2, R_3, R_4\}$ is normalised w.r.t. $\{f, g, h\}$, for none of these symbols occurs in the rhs expression of any rule of $\mathcal{R}$.

(b) $\mathcal{R}' = \mathcal{R} \cup \{R_5\}$ is not normalised w.r.t. $\{f, g, h, m\}$, since m occurs in the rhs expression of R_3. However, since the application of $\mathcal{R}'$ to the rhs expression of R_3 generates a normalised rule,

$$R_3'. \quad g(v_1, h(v_1, v_2)) \Rightarrow k(v_2, v_1, v_2)$$

$\mathcal{R}'$ is normalisable w.r.t. $\{f, g, h, m\}$.

(c) $\mathcal{R}'$ is not normalisable w.r.t. $\{f, g, h, m, l\}$, considering that no rule of $\mathcal{R}'$ can remove the occurrences of l from the rhs expressions of R_2 and R_4.

ii. The total remove *rules*

$$\begin{array}{lrcl} R_1. & f(v_1, v_2) & \Rightarrow & h(v_1, g(v_2, v_2)) \\ R_2. & g(v_1, v_2) & \Rightarrow & f(h(v_2, a), v_1) \end{array}$$

are such that $R_1 \sqsubset R_2$ and $R_2 \sqsubset R_1$. $\mathcal{R} = \{R_1, R_2\}$ is not normalised w.r.t. $\mathcal{S} = \{f, g\}$, since at least one of these symbols occurs in the rhs expressions of R_1 and R_2. $\mathcal{R}$ is not normalisable w.r.t. $\mathcal{S}$ either, considering that the rewriting sequences for the rhs expressions of both R_1 and R_2,

$$\begin{array}{lllll} h(v_1, g(v_2, v_2)) & \overset{\mathcal{R}}{\Rightarrow} & h(v_1, f(h(v_2, a), v_2)) & \overset{\mathcal{R}}{\Rightarrow} & h(v_1, h(h(v_2, a), g(v_2, v_2))) \overset{\mathcal{R}}{\Rightarrow} \ldots \\ f(h(v_2, a), v_1) & \overset{\mathcal{R}}{\Rightarrow} & h(h(v_2, a), g(v_1, v_1)) & \overset{\mathcal{R}}{\Rightarrow} & h(h(v_2, a), f(h(v_1, a), v_1)) \overset{\mathcal{R}}{\Rightarrow} \ldots \end{array}$$

are such that no rewritten expression is free from occurrences of both f and g.

□

When totally removable deviant symbols are allowed in the rhs expression of a *remove* rule, its application must be later followed by the total rules for the newly introduced symbols. The normalisation of $\mathcal{R}$ erases such symbols once and for all from the rule level. Normalised sets of *remove* rules therefore prevent the superfluous introduction of deviant symbols in the course of rewriting. If the subset of total *remove* rules of a non-confluent set $\mathcal{R}$ is normalisable, the subset is noetherian, as proved below.

Lemma 6.2.1 *If $\mathcal{R}$ is a normalised set of total* remove *rules, then $\mathcal{R}$ is noetherian.*

PROOF. Let $\mathcal{S}$ be the set of symbols for which $\mathcal{R}$ provides total *remove* rules, and, given a formula ϕ, let $m_{\mathcal{S}}(\phi)$ be the list of natural numbers that identify the number of occurrences of elements of $\mathcal{S}$ in each layer of this formula[4]. Termination for rewriting is ensured once it is established that

(i) $m_{\mathcal{S}}$ decreases (according to the lexicographic order for lists) after the application of each rule of $\mathcal{R}$, and

(ii) the introduction of new layers in the rewritten formula has an upper bound[5].

Concerning (i), since any total *remove* rule that is normalised w.r.t. $\mathcal{S}$ has the form

$$S(v_1, \ldots, v_n) \Rightarrow \delta\{v_1, \ldots, v_n\}$$

where no symbol of δ is removable by any element of $\mathcal{R}$, the application of a total *remove* rule for S to ϕ replaces an occurrence of S in a certain layer with another symbol that does not belong to $\mathcal{S}$, thus reducing by one the number of occurrences of $\mathcal{S}$-symbols in that layer[6]. Even though new occurrences of $\mathcal{S}$-symbols may be introduced

---

[4] The *layers* of an expression are defined in section 8.1.1.

[5] The second requirement derives from the fact that there is no minimal element for the lexicographic order of lists, e.g.

$$[1] > [0, 1] > [0, 0, 1] > \ldots$$

As a result, the reduction of the number of occurrences of a symbol at higher layers does not guarantee termination for the application of a generic rewrite rule. For instance,

$$R.\ f(v) \Rightarrow k(f(v))$$

reduces the number of occurrences of f at the top layer of the rewritten subexpression, but its exhaustive application does not halt, since the lhs expression is a subexpression of the rhs expression.

[6] This remark excludes total *remove* rules for identity and projection functions, which have the forms

in lower layers (since variables of [normalised] rules may be instantiated to expressions that contain $\mathcal{S}$-symbols), the removal of a single higher occurrence of a $\mathcal{S}$-symbol outweighs the introduction of any number of new occurrences below it, according to the lexicographic order.

Concerning (ii), let

(a) n be total number of occurrences of $\mathcal{S}$-symbols in ϕ,

(b) m be the maximum number of layers in a rhs expression of any rule of $\mathcal{R}$,

(c) L be the total number of layers of ϕ.

The removal of each occurrence of S causes the introduction of a maximum number of $(m-2)$ new layers in the rewritten formula, as illustrated in figure 6.1. The maximum expansion for ϕ takes place when all the occurrences of $\mathcal{S}$-symbols in ϕ are nested, with each occurrence taking place at a distinct layer, with or without other interleaving symbols: in this case, $n \times (m-2)$ new layers are introduced (i.e. $(m-2)$ layers for each occurrence of a $\mathcal{S}$-symbol in the original formula)[7].

Since the number of new layers is finite, and since $m_{\mathcal{S}}$ decreases after each rule application, $\mathcal{R}$ is noetherian. ∎

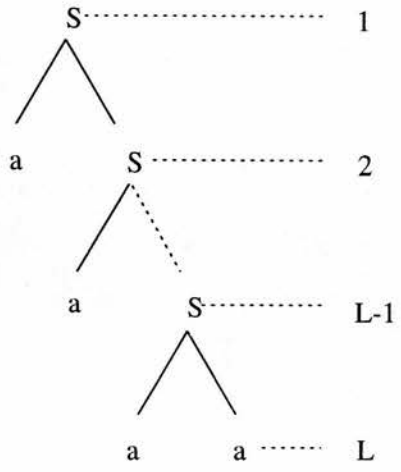
A termination measure is obtained from $m_{\mathcal{S}}$ once the maximum number of symbols N that can occur in any layer of the rewritten formula, ϕ', is taken into account. Since the maximum number of layers of ϕ' is $L + n \times (m-2)$, where L, n and m have been defined in the above lemma, N depends on the greatest arity of any symbol that occurs in $\{\phi\} \cup \mathcal{R}$. If r is such arity, then

$$N = r^{L+n\times(m-2)-1}$$

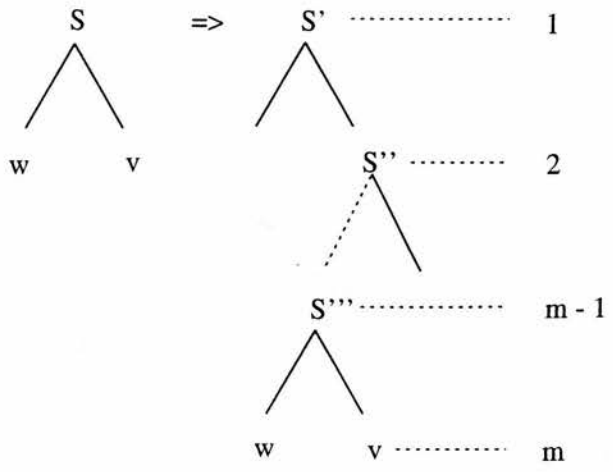
$$\begin{aligned} i(v) &\Rightarrow v \\ f_{p_i}(v_1, \ldots, v_n) &\Rightarrow v_i \end{aligned}$$

Termination for the application of these rules can be ensured based on the fact that, in both cases, the rhs expression is a proper subexpression of the lhs expression.

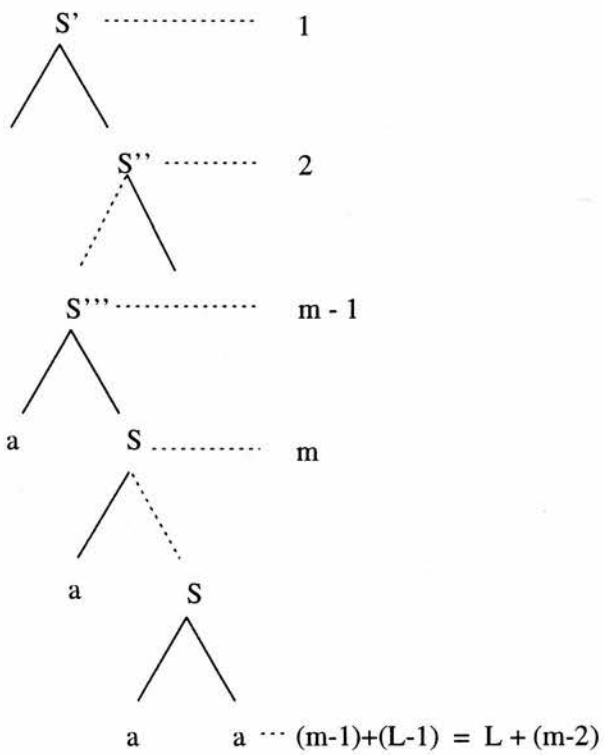
[7] See appendix E for an inductive proof of this result.



Conjecture



Total *remove* rule



Rewritten conjecture

Figure 6.1: Expansion of rewritten formulae

If n_i is the number of occurrences of totally removable symbols in the layer of depth i, the measure defined as

$$m_t(\phi) = \sum_{i=1}^{l} n_i \times N^{L+n\times(m-2)+1-i}$$

assigns a heavier weight to occurrences of $\mathcal{S}$-symbols that take place at higher layers. Clearly, m_t assumes values only in N, and it decreases after the application of any rule of $\mathcal{R}$. However, since it is guaranteed that the expansion of the rewritten formula has an upper bound, the original list measure $m_{\mathcal{S}}$ can be adopted instead of its more complex arithmetical version, as illustrated in the example below.

Example 6.2.2 *The atomic formula* ϕ,

$$p(g(x,a), f(h(g(y,b), g(x,y), b)), h(c,x,z), a)$$

has five layers,

$$\begin{array}{ll} \ell_1 & p \\ \ell_2 & g, f, h, a \\ \ell_3 & x, a, h, c, x, z \\ \ell_4 & g, g, b \\ \ell_5 & y, b, x, y \end{array}$$

If a, g *and* h *are deviant symbols for which total* remove *rules are available, then* $m_t(\phi) = [0, 3, 2, 2, 0]$. *If the total* remove *rule*

$$h(v_1, v_2, v_3) \Rightarrow k(v_1, v_1, f(v_2), v_3)$$

is applied to ϕ *for the removal of the occurrence of* h *that takes place at* ℓ_3, *the rewritten formula* ϕ' *is*

$$p(g(x,a), f(k(g(y,b), g(y,b), f(g(x,y)), b)), h(c,x,z), a)$$

Given the new distribution of symbols amongst its layers,

$$\begin{array}{ll} \ell_1 & p \\ \ell_2 & g, f, h, a \\ \ell_3 & x, a, k, c, x, z \\ \ell_4 & g, g, f, b \\ \ell_5 & y, b, y, b, g \\ \ell_6 & x, y \end{array}$$

it follows that $m_t(\phi') = [0,3,1,2,1,0]$. *The number of occurrences of totally removable deviant symbols has been reduced by one in* ℓ_3, *and a new occurrence has been introduced at* ℓ_5. *Since a reduction in the number of occurrences of totally removable deviant symbols took place at a higher layer, the measure* m *has been reduced, i.e.*

$$[0,3,1,2,1,0] < [0,3,2,2,0]$$

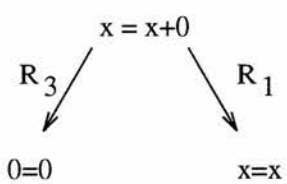
□

Local confluence for normalised sets of total *remove* rules is an immediate consequence of the Knuth and Bendix theorem. Since they are also noetherian, their confluence then follows[8]. Sets of partial *remove* rules, on the other hand, may be non-confluent.

Example 6.2.3 *The set of partial* remove *rules for* $+$, *made up of*

$$\begin{array}{lrcl} R_1. & v + 0 & \Rightarrow & v \\ R_2. & v_1 + v_2 = v_1 + v_3 & \Rightarrow & v_2 = v_3 \\ R_3. & v_1 = v_1 + v_2 & \Rightarrow & 0 = v_2 \\ R_4. & v_1 + v_2 = v_1 & \Rightarrow & v_2 = 0 \end{array}$$

is not confluent, since $x = x + 0$ *has two normal forms,*



□

As illustrated in example 6.2.1, not all sets of *remove* rules are normalisable. A sufficient condition for normalisability requires the antecedent relation for total rules to be a partial order.

Lemma 6.2.2 *Given a set* $\mathcal{R} = \{R_1, \ldots, R_n\}$ *of total* remove *rules for a set of symbols* $\mathcal{S} = \{S_1, \ldots, S_n\}$, $\mathcal{R}$ *is normalisable w.r.t.* $\mathcal{S}$ *provided that* $\langle \mathcal{R}, \sqsubseteq \rangle$ *is a partial order.*

---

[8] See [Knuth & Bendix 70] for a proof of this theorem. A restricted version of this proof, applied to normalised sets of rules, can be found in appendix E. For an account on properties of rewrite systems, see, for instance, [Bundy 83], p. 158-66.

PROOF. (By induction on the number of *remove* rules)

Base case. Trivial, since $\mathcal{R} = \{R_1\}$ is normalised w.r.t. $\mathcal{S} = \{S_1\}$.

Step case. Assume that $\mathcal{R} = \{R_1, \ldots, R_n\}$ is normalisable w.r.t. $\mathcal{S}$. As a result, there is a set $\mathcal{R}' = \{R'_1, \ldots, R'_n\}$ that is normalised w.r.t. $\mathcal{S}$. Let R_{n+1} be a total *remove* rule for a symbol S_{n+1}, such that $\langle \mathcal{R} \cup \{R_{n+1}\}, \sqsubseteq \rangle$ is a partial order. Two cases have to be considered.

i. R_{n+1} is minimal[9]. If S_{n+1} does not occur in any element of $\mathcal{R}'$, let R'_{n+1} be derived from R_{n+1} by the exhaustive application of $\mathcal{R}'$ to its rhs expression. According to lemma 6.2.1, the generation of R'_{n+1} is effective, and R'_{n+1} is free from occurrences of symbols from $\mathcal{S}$. Hence $\mathcal{R}' \cup \{R'_{n+1}\}$ is normalised w.r.t. $\mathcal{S} \cup \{S_{n+1}\}$, and $\mathcal{R} \cup \{R_{n+1}\}$ is normalisable w.r.t. $\mathcal{S} \cup \{S_{n+1}\}$.

ii. R_{n+1} is not minimal. If S_{n+1} occurs in rules of $\mathcal{R}'$, let $\mathcal{R}'' = \{R'_{i_1}, \ldots, R'_{i_m}\}$ contain all of such rules (and no other), hence $R'_{i_j} \sqsubset R_{n+1}$, $1 \leq j \leq m$ (and also $R'_{i_j} \sqsubseteq R_{n+1}$, given that $\sqsubset$ is a subrelation of $\sqsubseteq$). Since $\langle \mathcal{R} \cup \{R_{n+1}\}, \sqsubseteq \rangle$ is a partial order, the same applies to $\langle \mathcal{R}' \cup \{R_{n+1}\}, \sqsubseteq \rangle$. Since $R'_{i_j} \sqsubseteq R_{n+1}$, $1 \leq j \leq m$, it follows that $R_{n+1} \not\sqsubseteq R'_{i_j}$, hence $R_{n+1} \not\subset R'_{i_j}$ and $S_{i_1}, \ldots, S_{i_m}$ do not occur in R_{n+1}. Let R'_{n+1} be derived from R_{n+1} by the exhaustive application of $\mathcal{R}' - \mathcal{R}''$ to its rhs expression. Since $\mathcal{R}'$ is normalised w.r.t. $\mathcal{S}$, the same applies to $\mathcal{R}' - \mathcal{R}''$. Again, according to lemma 6.2.1, R'_{n+1} is effectively generated, and is free from any occurrence of symbols of $\mathcal{S} \cup \{S_{n+1}\}$. Let $R''_{i_1}, \ldots, R''_{i_m}$ be generated from $R'_{i_1}, \ldots, R'_{i_m}$ by the exhaustive application of R'_{n+1} to their rhs expressions. As a result,

$$(\mathcal{R}' - \mathcal{R}'') \cup \{R''_{i_1}, \ldots, R''_{i_m}, R'_{n+1}\}$$

is normalised w.r.t. $\mathcal{S} \cup \{S_{n+1}\}$, and $\mathcal{R} \cup \{R_{n+1}\}$ is normalisable w.r.t. $\mathcal{S} \cup \{S_{n+1}\}$.

∎

[9] Given a partially ordered set $\langle A, \subseteq \rangle$, an element $a \in A$ is *minimal* w.r.t. $\subseteq$ iff there is no element $b \in A$, $b \neq a$, such that $b \subseteq a$.

Based on lemma 6.2.1, it is possible to guarantee that the extension provided by a normalised set of total *remove* rules is also a decidable sublanguage.

Lemma 6.2.3 *Let T be an undecidable theory in $\mathcal{L}$, and let $\mathcal{L}'$ be a decidable sublanguage for T. If $\mathcal{R}$ is a finite and normalised set of total* remove *rules for deviant symbols of $\mathcal{L}$ w.r.t. $\mathcal{L}'$, the domain of $\mathcal{R}$ is a decidable sublanguage for T.*

PROOF. Let $S_1, \ldots, S_n$ be $\mathcal{L}$-symbols for which $\mathcal{R}$ provides total *remove* rules. The domain of $\mathcal{R}$ is the set of all formulae of $\mathcal{L}$ that contain at least one occurrence of S_i, $1 \leq i \leq n$. If $\mathcal{L}''$ is defined as the language whose set of non-logical symbols is $Sym_{\mathcal{L}'} \cup \{S_1, \ldots, S_n\}$, the domain of $\mathcal{R}$ can be defined as

$$\Sigma = Fml_{\mathcal{L}''} - Fml_{\mathcal{L}'}$$

Considering that

(i) any rewriting sequence under $\mathcal{R}$ is finite, since $\mathcal{R}$ is noetherian, and

(ii) $\mathcal{R}$ is finite (and therefore determining the existence of an applicable rule in $\mathcal{R}$ to an expression is effective),

it follows that the computation of the normal form of any expression under $\mathcal{R}$ is effective. Hence $\mathcal{L}''$ is a decidable sublanguage for T. ∎

This lemma can be extended to sets of rules containing complete sets of *remove* rules as well. The possibility of decomposing rewriting into two independent stages follows from the results obtained so far.

Lemma 6.2.4 *Let $\mathcal{R}$ be a set of total* remove *rules for $\mathcal{S}$ and $\mathcal{R}'$ be a set of partial* remove *rules for $\mathcal{S}'$, such that $\mathcal{R}$ and $\mathcal{R}'$ are normalised w.r.t. $\mathcal{S}$. Given any expression ϵ and a rewriting sequence under $\mathcal{R} \cup \mathcal{R}'$ of the form*

$$\epsilon \overset{\mathcal{R} \cup \mathcal{R}'}{\Longrightarrow} \epsilon'$$

there are (possibly empty) rewriting sequences under $\mathcal{R}$ and $\mathcal{R}'$ and an expression ϵ'' such that

$$\epsilon \overset{\mathcal{R}}{\Rightarrow} \epsilon'' \quad and \quad \epsilon'' \overset{\mathcal{R}'}{\Rightarrow} \epsilon'$$

PROOF. (By induction on the length of the rewriting sequence)

If $\epsilon \overset{\mathcal{R}\cup\mathcal{R}'}{\Longrightarrow} \epsilon'$, there is a finite rewriting sequence of the form

$$\epsilon \overset{R_1}{\Rightarrow} \epsilon_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_{n-1}}{\Rightarrow} \epsilon_{n-1} \overset{R_n}{\Rightarrow} \epsilon'$$

from which a sequence of the desired form can be derived, for any $n \in \mathbb{N}$.

Base case. $\epsilon \overset{\mathcal{R}\cup\mathcal{R}'}{\Longrightarrow} \epsilon$ (empty rewriting sequence). Trivial.

Step case. Assume that, for any rewriting sequence of length n, there is a sequence of the desired form. Let then

$$\epsilon \overset{R_1}{\Rightarrow} \epsilon_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_n}{\Rightarrow} \epsilon_n \overset{R_{n+1}}{\Rightarrow} \epsilon' \quad (*)$$

be a sequence of length $n+1$. If $R_{n+1} \in \mathcal{R}'$, according to the induction hypothesis, $\epsilon \overset{R_1}{\Rightarrow} \epsilon_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_n}{\Rightarrow} \epsilon_n$ can be reduced to a sequence $\boldsymbol{\epsilon}$ of the desired form, and $\boldsymbol{\epsilon} \overset{R_{n+1}}{\Rightarrow} \epsilon'$ is the new sequence for $(*)$. On the other hand, if $R_{n+1} \in \mathcal{R}$, let $\epsilon \overset{R_1}{\Rightarrow} \ldots \overset{R_n}{\Rightarrow} \epsilon_n$ be replaced by a rewriting sequence of the desired form, $\epsilon \overset{R'_1}{\Rightarrow} \ldots \overset{R'_p}{\Rightarrow} \epsilon_n$, where $p \in \mathbb{N}$. Given the subsequence

$$\widehat{\epsilon}_{n-1} \overset{R'_p}{\Rightarrow} \epsilon_n \overset{R_{n+1}}{\Rightarrow} \epsilon'$$

since R_{n+1} is applicable to ϵ_n, it must be also applicable to $\widehat{\epsilon}_{n-1}$, considering that

(a) R'_p is normalised w.r.t. $\mathcal{S}$ and does not introduce any occurrences of totally removable symbols, and

(b) the applicability of R_{n+1} to ϵ_n means that S_{n+1} occurs in ϵ_n.

From (a) and (b), it follows that there is an occurrence of S_{n+1} in ϵ_n which has not been introduced by the application of R'_p to $\widehat{\epsilon}_{n-1}$, hence S_{n+1} must occur in $\widehat{\epsilon}_{n-1}$. If $\widehat{\epsilon}_n$ is the expression generated by the application of R_{n+1} to $\widehat{\epsilon}_{n-1}$, there are yet again two cases to be taken into account.

i. If R'_p has been previously matched against a subexpression of $\widehat{\epsilon}_{n-1}$ that does not contain $S_{n+1} \in \mathcal{S}$, where R_{n+1} is a total *remove* rule for S_{n+1}, then R'_p can be matched against an identical subexpression of $\widehat{\epsilon}_n$.

ii. Otherwise let R'_p have the form $\delta_{n,1} \Rightarrow \delta_{n,2}$. As R'_p is normalised w.r.t. $\mathcal{S}$, $\delta_{n,1}$ is matchable against a subexpression β_{n-1} of $\widehat{\epsilon}_{n-1}$ that contains S_{n+1} iff there is a substitution such that

$$\left\{ {}^{t_1}/_{v_1}, \ldots, {}^{S_{n+1}\mathbf{u}}/_{v_i}, \ldots, {}^{t_n}/_{v_n} \right\} \delta_{n,1} \stackrel{s}{=} \beta_{n-1}$$

for some $i, 1 \leq i \leq n$. Then it is also possible to match $\delta_{n,1}$ against β_n, where $\beta_{n-1} \stackrel{R_{n+1}}{\Rightarrow} \beta_n$, by means of the substitution

$$\left\{ {}^{t_1}/_{v_1}, \ldots, {}^{\delta'_{n+1,2}}/_{v_i}, \ldots, {}^{t_n}/_{v_n} \right\}$$

such that $S_{n+1}\mathbf{u} \Rightarrow \delta'_{n+1,2}$ is an instance of R_{n+1}. ∎

Example 6.2.4 *Let $\mathcal{S} = \{s\}$ and $\mathcal{S}' = \{\times\}$, and let $\mathcal{R}$ and $\mathcal{R}'$ be unitary sets respectively composed of the following rewrite rules,*

$$\begin{array}{lrcl} R_1. & s(v) & \Rightarrow & v+1 \\ R_2. & v \times 0 & \Rightarrow & 0 \end{array}$$

The conjecture

$$(s(x+y^2)) \times 0 < z$$

can be rewritten under $\mathcal{R} \cup \mathcal{R}'$ as follows.

$$\underline{(s(x+y^2)) \times 0} < z \stackrel{\mathcal{R} \cup \mathcal{R}'}{\Longrightarrow} 0 < z$$

It can be also successively rewritten under $\mathcal{R}$, then $\mathcal{R}'$, in two alternative ways,

$$\underline{(s(x+y^2))} \times 0 < z \stackrel{\mathcal{R}}{\Rightarrow} \underline{((x+y^2)+1) \times 0} < z \stackrel{\mathcal{R}'}{\Rightarrow} 0 < z$$

and

$$(s(x+y^2)) \times 0 < z \stackrel{\mathcal{R}}{\Rightarrow} \underline{(s(x+y^2)) \times 0} < z \stackrel{\mathcal{R}'}{\Rightarrow} 0 < z$$

and the same rewritten formula is obtained in all three cases. It is worth noting that, in the third sequence, the rewriting sequence under $\mathcal{R}$ is empty. □

The removal of deviant symbols can therefore be first limited to totally removable symbols, which is then followed by the application of partial rules. For the first group of symbols, assuming that the set of total rules $\mathcal{R}$ is normalised, no additional control guideline is in principle necessary, due to the confluence of the set. As shown in figure 6.2, given an initial language $\mathcal{L}$ and a decidable sublanguage $\mathcal{L}'$, $\mathcal{R}$ defines an intermediate sublanguage, $\mathcal{L}''$, to which any formula of $\mathcal{L}$ can be reduced. The restriction of the second rewriting stage to the application of partial rules contracts the search space, thus improving efficiency.

6.2.2 Non-noetherian Sets

Termination is not an intrinsic property of sets of *remove* rules. Given a set of partial rules respectively for the removal of f, g and i,

$$\begin{array}{lrcl} R_1. & f(x,a) & \Rightarrow & h(g(x,b),x) \\ R_2. & h(g(x,y),z) & \Rightarrow & i(a,h(x,y),z) \\ R_3. & i(x,y,a) & \Rightarrow & f(a,a) \end{array}$$

where a and b are individual constants, rewriting does not halt e.g. for $f(a,a)$, since

$$f(a,a) \overset{R_1}{\Rightarrow} h(g(a,b),a) \overset{R_2}{\Rightarrow} i(a,h(a,b),a) \overset{R_3}{\Rightarrow} f(a,a)$$

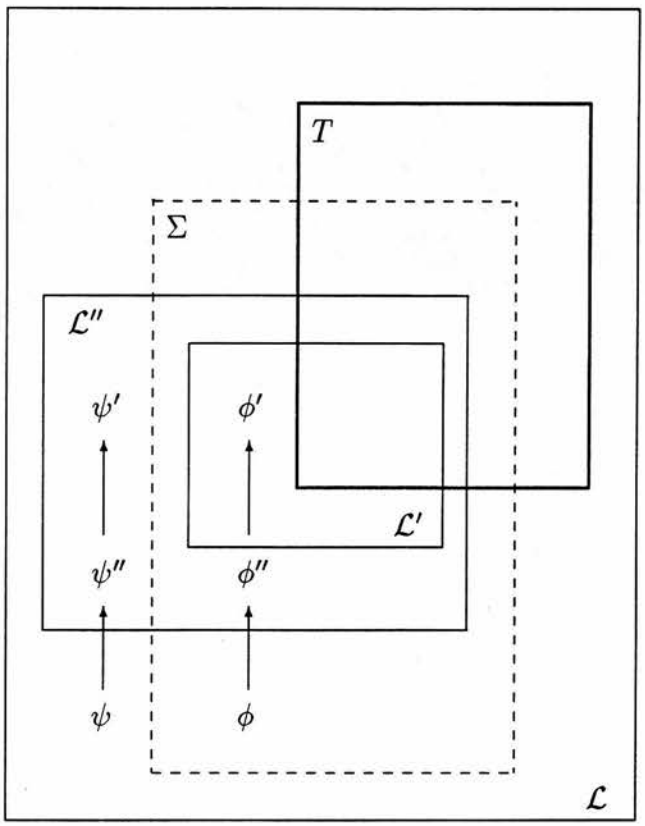
Termination can be imposed upon non-noetherian sets by means of special control devices, at the expense of completeness. There are at least three restrictions for the selection of *remove* rules, each of which integrates a distinct strategy for cycle prevention: given a conjecture ϕ, a rule $\delta_1 \Rightarrow \delta_2$ would be selected for application iff

(i) no deviant symbol occurs in δ_2, or

(ii) no deviant symbol absent from ϕ occurs in δ_2, or

(iii) no deviant symbol previously removed from ϕ occurs in δ_2.

Criteria (i) and (ii) are too strong, and would cause the exclusion of otherwise acceptable rules. Criterion (iii), in conjunction with a termination measure for the removal of each individual deviant symbol, ensures that the global process halts. An irreflexive relation for deviant symbols helps in the construction of an adequate ordering.



$\mathcal{L}$ Original language
(initial stage)
$\phi, \psi \in Fml_{\mathcal{L}}$

$\mathcal{L}''$ Intermediate sublanguage
(second stage – application of total *remove* rules)
$\phi'', \psi'' \in Fml_{\mathcal{L}''}$

$\mathcal{L}'$ Decidable sublanguage
(final stage – application of partial *remove* rules)
$\phi' \in Fml_{\mathcal{L}'}$, $\psi' \notin Fml_{\mathcal{L}'}$

Σ Extended decidable subclass
$\phi \in \Sigma$, $\psi \notin \Sigma$

Figure 6.2: Two-phased Removal of Deviant Symbols

Definition 6.2.2 (Irreflexive Relation for Symbols)

Let $\mathcal{S}$ be a set of symbols and $\mathcal{R}$ be a set of remove *rules for $\mathcal{S}$.*

i. Given $S_1, S_2 \in \mathcal{S}$,

$$S_1 \prec S_2$$

iff there is a remove *rule for S_1 in $\mathcal{R}$ whose rhs expression contains an occurrence of S_2.*

ii. $\overset{*}{\prec}$ is the transitive closure of $\prec$.

From the definition of *remove* rules, it follows that $\prec$ is irreflexive. If R_1 and R_2 are total *remove* rules respectively for S_1 and S_2 such that $R_1 \sqsubset R_2$, then $S_1 \prec S_2$. A sufficient condition for the existence of a termination ordering for a set of deviant symbols $\mathcal{S}$ is that $\langle \mathcal{S}, \overset{*}{\prec} \rangle$ is well-founded. Once a well-founded order is found, a total order can be always exhibited.

Theorem 6.2.1 *Let $\mathcal{R}$ be a set of* remove *rules for a set of symbols $\mathcal{S} = \{S_1, \ldots, S_n\}$ which are deviant w.r.t. a decidable sublanguage $\mathcal{L}'$. Let $\mathcal{R}_i$, the subset of $\mathcal{R}$ that contains all the* remove *rules for S_i, be noetherian, for each $i, 1 \leq i \leq n$. If $\langle \mathcal{S}, \overset{*}{\prec} \rangle$ is well-founded, then any extension of $\overset{*}{\prec}$ to a total order is a termination ordering for the removal of deviant symbols of $\mathcal{S}$.*

PROOF. (By strong induction on the number of deviant symbols)

Assume that, for any set $\mathcal{S}$ with up to n deviant symbols, any extension of $\langle \mathcal{S}, \overset{*}{\prec} \rangle$ to a total order ensures termination for the rewriting process. Let then S_{n+1} be a deviant symbol for which there is a noetherian set of *remove* rules, $\mathcal{R}_{n+1}$, such that $\langle \mathcal{S} \cup \{S_{n+1}\}, \overset{*}{\prec} \rangle$ is well-founded. Let also

$$\begin{aligned} \mathcal{S}_1 &= \{S_i \in \mathcal{S} \mid S_i \prec S_{n+1}\} \\ \mathcal{S}_2 &= \{S_j \in \mathcal{S} \mid S_{n+1} \prec S_j\} \\ \mathcal{S}_3 &= \{S_k \in \mathcal{S} \mid S_k \not\prec S_{n+1} \;\&\; S_{n+1} \not\prec S_k\} \end{aligned}$$

be a partition for $\mathcal{S}$. Clearly, no *remove* rule for a symbol of $\mathcal{S}_2$ introduces any of the symbols of $\mathcal{S}_3$, for otherwise there would be $S_j \in \mathcal{S}_2$ and $S_k \in \mathcal{S}_3$ such that $S_j \prec S_k$. From the definitions of $\mathcal{S}_2$ and $\overset{*}{\prec}$, it would then follow that $S_{n+1} \overset{*}{\prec} S_k$, in contradiction with the definition of $\mathcal{S}_3$. Also, no *remove* rule for a symbol of $\mathcal{S}_2$ introduces any of the symbols of $\mathcal{S}_1$, for otherwise there would be $S_j \in \mathcal{S}_2$ and $S_i \in \mathcal{S}_1$ such that $S_j \prec S_i$. From the definitions of $\mathcal{S}_2$ and $\overset{*}{\prec}$, it would then follow that $S_{n+1} \overset{*}{\prec} S_i$, in contradiction with the definition of $\mathcal{S}_1$ and the asymmetry of $\overset{*}{\prec}$ (given that $\overset{*}{\prec}$ is a well-founded relation and, therefore, an (irreflexive) partial order[10]).

As a result, no *remove* rule for S_{n+1} introduces any symbol of $\mathcal{S}_1 \cup \mathcal{S}_3$, and no *remove* rule for any symbol of $\mathcal{S}_2$ introduces a symbol of $\mathcal{S}_1 \cup \mathcal{S}_3 \cup \{S_{n+1}\}$. Hence, the removal of $\mathcal{S}_1 \cup \mathcal{S}_3$ may precede $\{S_{n+1}\}$, which in turn may precede $\mathcal{S}_2$. From the induction hypothesis, given that $\#(\mathcal{S}_1 \cup \mathcal{S}_3) \leq n$ and $\#(\mathcal{S}_2) \leq n$, there is a termination ordering for the removal of symbols from both sets. Let $\boldsymbol{S}'$ and $\boldsymbol{S}''$ denote possible total ordering respectively for $\mathcal{S}_1 \cup \mathcal{S}_3$ and $\mathcal{S}_2$. Then

$$\langle \boldsymbol{S}', S_{n+1}, \boldsymbol{S}'' \rangle$$

is a termination ordering for the removal of symbols of $\mathcal{S} \cup \{S_{n+1}\}$. ∎

To sum it up, given

(a) an undecidable theory T in $\mathcal{L}$,

(b) a set of deviant symbols $\mathcal{S} \cup \mathcal{S}'$ w.r.t. a decidable sublanguage $\mathcal{L}'$,

(c) a set of total *remove* rules $\mathcal{R}$ for $\mathcal{S}$, such that $\mathcal{R}$ is normalised w.r.t. $\mathcal{S}$, and

(d) a set of partial *remove* rules $\mathcal{R}'$ for $\mathcal{S}'$, such that $\mathcal{R}'$ is normalised w.r.t. $\mathcal{S}$,

rewriting control for the reduction of a formula $\phi \in Fml_{\mathcal{L}}$ into $\mathcal{L}'$ starts with the set of totally removable symbols. Since $\mathcal{R}$ is normalised w.r.t. $\mathcal{S}$, the exhaustive application

[10] The ordered pair $\langle \mathcal{A}, < \rangle$ is an *irreflexive partial order* iff

(i) $(v)\neg(v < v)$ (irreflexivity)

(ii) $(v_1)(v_2)(v_3)((v_1 < v_2 \wedge v_2 < v_3) \supset v_1 < v_3)$ (transitivity)

From (i) and (ii), it follows that $<$ is also asymmetric, i.e.

$$(v_1)(v_2)(v_1 < v_2 \supset \neg(v_2 < v_1))$$

of its rules effectively reduces ϕ to a formula $\phi' \in Fml_{\mathcal{L}''}$, where $Sym_{\mathcal{L}''} = Sym_{\mathcal{L}} - \mathcal{S}$. The order for the removal of symbols of $\mathcal{S}$ is immaterial, since $\mathcal{R}$ is confluent. Once a total ordering for the symbols of $\mathcal{S}'$ is built on top of a well-founded relation $\prec$, ϕ' is further rewritten by the ordered application of rules from $\mathcal{R}'$, provided that each subset $\mathcal{R}'_i$ for $S_i \in \mathcal{S}'$ is noetherian[11]. If all occurrences of each deviant symbol are removed, or if the complete removal of any deviant symbol according to the pre-established order fails, the process halts. There may be cases where more than one total ordering for $\mathcal{S}'$ can be built from $\prec$, in which case additional guidelines for the selection of an ordering become relevant.

When the set $\mathcal{R}$ of total *remove* rules is normalised (w.r.t. $\mathcal{S}$), and none of its elements introduces any of the partially removable deviant symbols of $\mathcal{S}'$, the order for the removal of deviant symbols can be reversed. Partially removable symbols would then come first, and totally removable symbols would be taken into account only if all symbols of the first group were successfully eliminated from a conjecture. The desired reduction would be always accomplishable once the first stage is completed, given that total rules are always applicable, provided that the corresponding symbols occur in the expression. Hence, in the event the removal of symbols of $\mathcal{S}'$ fails, no effort is wasted in the elimination of symbols of $\mathcal{S}$.

6.3 Additional Control Features

The control guidelines for noetherian rule sets examined in the previous sections cause no deductive losses vis-à-vis exhaustive rewriting. Further constraints, based on heuristic functions and measures, can be introduced, even though completeness may be affected. There are at least three new aspects to be explored: the selection of a decidable sublanguage, the choice of an ordering for partially removable symbols, and the choice of *remove* rules for these symbols.

Concerning decidable sublanguages, the number of deviant symbols present in a conjecture and the type of the *remove* rules (*total, partial dominant* or *partial non-dominant*) available for them have to be taken into account. The total number of steps required

[11] Noetherian sets of partial rules are examined in chapter 9.

to reduce a formula to a sublanguage can be estimated as the linear combination of the complexity measures for the removal of each deviant symbol.

Orderings for partially removable deviant symbols of a particular decidable subclass can then be identified, when more than one is permissible: properties of the partial rules have a decisive role at this stage. Thereafter, partial *remove* rules for each deviant symbol have to be organised, whenever there is more than one such rule. Their types (*dominant* or *non-dominant*) and the new symbols introduced in the rewritten conjecture are determinant factors. No additional loss of completeness occurs at any of these two stages when all possible orderings are examined[12].

Specific solutions regarding the above parameters can be based on a heuristic function that assesses the complexity for the removal of a symbol, once a rule set is given.

6.3.1 Complexity Measure Functions

The complexity of a computation can be measured from the viewpoint of either time or space properties. Time complexity determines the number of all atomic operations, or the number of repetitions of a particular operation or group of operations, that take place in a process. Space complexity assesses the sizes of the data structures involved in such operations[13].

Given a syntactic size function s: $Dom \rightarrow \mathbb{N}$, defined in the domain Dom of a procedure P, and a component operation q of P whose execution is to be counted, there are two measures for time complexity. *Worst case complexity* (wcc) for a size z establishes the maximum number of times q is executed when any object $o \in Dom$ with size $s(o) = z$ is supplied to P,

$$wcc(z) = max(\langle \#q, P(o) \rangle), \quad \text{for all } o \in Dom \mid s(o) = z$$

The *average case complexity* (acc) for z requires a probability distribution function p for Dom (i.e. a function p: $Dom \rightarrow [0,1]$ such that $\sum_{o \in Dom} p(o) = 1$). The measure for z is the weighted sum of the complexity measures for every object $o \in Dom$ of size z,

[12] There may already be a loss of completeness, however, when there is more than one occurrence of at least one of the deviant symbols in the conjecture. See example 6.3.4.

[13] See [Sommerhalder & Van Westrhenen 88], p. 198-201.

$$acc(z) = \sum_{s(o)=z} p(o) \times \langle \#q, P(o) \rangle$$

Complexity measures for the removal of deviant symbols are the primary element in the construction of measures for decidable classes and *remove* rules. Their main purpose is to determine the average (rather than the maximum) number of rewriting steps expected to take place in each particular task, e.g. the removal of a disagreement, the transformation of a conjecture into an element of a decidable subclass or the complete normalisation of an expression. These functions can be entirely based upon syntactic properties of the relevant objects (formulae, rules, or classes of formulae), and no rewriting step has to be performed for their evaluation.

Concerning the removal of symbols, the type of the rule directly influences the size of the transformation tree. *Total* rules require just one rewriting step to eliminate an occurrence of a symbol, whereas *partial rules* may also involve the elimination of disagreements, whenever the rule is not applicable and a context transformation mechanism is interfaced to the system. An initial estimate for a *dominant* rule could assume that one additional elimination step is required on the average (to cover one disagreement), whereas two additional steps would suffice for a *non-dominant* rule (to adjust both the expression in which the symbol occurs and the expression it dominates). The function m_e,

$$m_e(S, R) = \begin{cases} 1 & \text{if } R \text{ is total} \\ 2 & \text{if } R \text{ is partial dominant} \\ 3 & \text{if } R \text{ is partial non-dominant} \end{cases}$$

then provides an estimate for the average-case time complexity for formulae where S occurs only once, provided that R is a *remove* rule[14]. When more than one *remove* rule for S is available, a new function, m_s, takes into account their average effect,

$$m_s(S) \quad = \quad \frac{1}{n} \sum_{i=1}^{n} m_e(S, R_i)$$

[14] The values for dominant and non-dominant *remove* rules should reflect the average number of steps involved in the tentative application of R to a formula that contains a single occurrence of S. Both estimated values could be replaced by a more accurate measure based on a suitable probability distribution function, which could take experimental data into account.

where $R_1, \ldots, R_n$ are the *remove* rules for S.

Both m_e and m_s are purely concerned with the removal of an occurrence of a symbol, without any assessment of possible side effects. When a decidable sublanguage and, as a result, a specific set of deviant symbols are determined, the removal of S must also take into account the new occurrences of deviant symbols introduced through rewriting, and the additional steps required for their later removal. In the case of partial rules, another factor is the additional deviant symbols that may be removed as part of a single rewriting step. The function m_d, defined as

$$\begin{aligned} m_d(S) &= \frac{1}{n}\sum_{i=1}^{n}\left(m_e(S,R_i) - n_r(R_i) + \sum_{S_j \in \mathcal{S}_i} m_d(S_j)\right) \\ &= m_s(S) + \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{S_j \in \mathcal{S}_i} m_d(S_j) - n_r(R_i)\right) \end{aligned}$$

where

$R_1, \ldots, R_n$	*remove* rules for S
$n_r(R_i)$	number of additional occurrences of deviant symbols removed by R_i
$\mathcal{S}_i$	multiset of deviant symbols introduced by R_i

assesses all these factors.

For formulae of size greater than one (i.e. those containing more than one occurrence of S), a simplified average-case complexity function, indicative of the average number of transformation steps (including those for raising disagreements) involved in the complete removal of S, could be defined as $n \times m_d(S)$. The underlying assumption is that the removal of n occurrences of S amounts to solving n problems where only one occurrence of S takes place[15].

6.3.2 Order Relations

One of the possible measures for the reduction of a formula ϕ into a decidable sublanguage $\mathcal{L}'$ is defined in terms of the occurrences of deviant symbols in ϕ and the complexity measure for the removal of these symbols,

[15] Ideally, the number of transformation steps predicted by a measure function should apply not only to those cases where the removal of a symbol is possible, but also to those in which failure has to be eventually acknowledged.

$$m_c(\phi, \mathcal{L}') \quad = \quad \sum_{S_i \in \mathcal{S}} n_i \times m_d(S_i)$$

where $\mathcal{S}$ denotes the set of deviant symbols of the original language w.r.t. $\mathcal{L}'$, and n_i, the number of occurrences of S_i in ϕ, for each $S_i \in \mathcal{S}$. Classes are then put in an increasing order, according to their respective measures.

Example 6.3.1 *Let ϕ be the arithmetical conjecture,*

$$(x)(y)(\exists z)(\exists w)(x \times (y + w) = x \times z \supset z \times y = z \times w)$$

As it contains both $+$ and $\times$, it does not belong to either $\mathcal{L}_{PrA} = \{0, 1, s, +\}$ or $\mathcal{L}_{SMA} = \{0, 1, \times\}$, both of which are decidable for PA. The options for its reduction are

(i) *the removal of $\times$, with $\mathcal{L}_{PrA}$ as target sublanguage, or*

(ii) *the removal of $+$, for the generation of an expression of $\mathcal{L}_{SMA}$.*

Assuming that there are two rewrite rules

$$\begin{array}{lrcl} R_1. & v_1 \times (v_2 + v_3) = v_1 \times v_4 & \Rightarrow & v_1 = 0 \vee \psi \\ R_2. & v_1 \times v_2 = v_1 \times v_3 & \Rightarrow & v_1 = 0 \vee v_2 = v_3 \end{array}$$

where ψ is the formula[16]

$$s(s(v_2)s^2(v_4))\ s(s(v_3)s^2(v_4)) = s(s(s(v_2)s(v_3))s((s^2(v_4))^2))$$

it could then be taken into account that

(i) *both R_1 and R_2 are immediately applicable to the conjecture,*

(ii) *there are four occurrences of $\times$ against only one of $+$, and the complete removal of $+$ requires only one rewriting step,*

(iii) *the* remove *rule for sum, R_1, introduces a new $\mathcal{L}_{SMA}$-deviant symbol, s, whereas the rule for $\times$ does not introduce any new deviant symbol, and*

(iv) *there is no* remove *rule for s in the rule base.*

[16] See [Boolos & Jeffrey 89], p. 219.

Therefore, although both $+$ *and* $\times$ *can be completely eliminated from* ϕ*, due to the lack of* remove *rules for* s*, the measure* $m_d(s)$ *is* ∞*, and, as a result, the measure* m_c *for* $\mathcal{L}_{PrA}$ *is lower.* □

Once a decidable sublanguage is chosen, only *remove* rules for the corresponding deviant symbols are eligible for application. An additional loss of completeness w.r.t. exhaustive rewriting results then from the exclusion of rules for non-deviant symbols.

Example 6.3.2 *Let* $\mathcal{R}$ *be the following set of* remove *rules.*

$$\begin{array}{lrcl} R_1 & v_1 \times v_2 = v_1 & \Rightarrow & v_1 = 0 \vee v_2 = 1 \\ R_2 & v_1 + v_2 = 0 & \Rightarrow & v_1 = 0 \wedge v_2 = 0 \\ R_3 & v_1 + v_2 = 1 & \Rightarrow & (v_1 = 1 \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = 1) \end{array}$$

Given the formula ϕ*,*

$$(x^2 + y^2) \times (z^3 + w) = x^2 + y^2$$

it could be reduced to $\mathcal{L}_{SMA}$ *by the application of a* remove *rule for* $\times$*,* R_1*, followed by two* remove *rules for* $+$*,* R_2 *and* R_3*, resulting in*

$$(x^2 = 0 \wedge y^2 = 0) \vee ((z^3 = 1 \wedge w = 0) \vee (z^3 = 0 \wedge w = 1))$$

However, since the application of remove *rules is restricted to the deviant symbols w.r.t. either* $\mathcal{L}_{SMA}$ *or* $\mathcal{L}_{PrA}$ *(i.e. to the removal of either* $+$ *or* $\times$*, but not both),* ϕ *cannot be reduced under* $\mathcal{R}$ *to any of these sublanguages*[17]*.* □

After the choice of a subclass, a total order for partially removable deviant symbols must be built, based for instance on the termination arguments associated with theorem 6.2.1. When there is more than one possible ordering, preference should be given to those where the first symbols to be removed have rules that minimise the expansion of the rewritten formula, since the proliferation of subexpressions at the first rewriting stages may exaggerately multiply the number of occurrences of deviant symbols.

[17] One of the roles of disagreement elimination mechanisms is to compensate for this deductive loss. In this example, ϕ could still be reduced to $\mathcal{L}_{SMA}$ by the strict application of *remove* rules for $+$, provided that a modified version of R_1 is employed as elimination equation (rather than as a *remove* rule) to raise the disagreement between ϕ and R_3. See section 8.1.1.

Sensitive symbols (i.e. those for which only a reduced number of partial rules is available) should also have priority: in the presence of multiple rules, it is more likely that a symbol can be removed at various stages of the process, whereas when just a few contexts are covered, any transformation may prevent a removal.

Example 6.3.3 *Let T be a theory in $\mathcal{L} = \{a, f, g, h, P\}$, and ϕ be the conjecture*

$$P(x, f(y, z)) \wedge P(a, z) \wedge P(y, g(y, z)) \wedge P(g(y, z), h(x, y))$$

If $\mathcal{L}' = \{a, h, P\}$ is a decidable sublanguage for T, then f and g are the only deviant symbols w.r.t. $\mathcal{L}'$ that occur in ϕ. Assuming that the rule set $\mathcal{R}$ contains

$$\begin{array}{lrcl} R_1 & P(v_1, f(v_2, v_3)) & \Rightarrow & P(v_1, g(v_2, v_3)) \\ R_2 & P(g(v_1, v_2), v_3) & \Rightarrow & P(f(v_1, v_2), v_3) \\ R_3 & P(v_1, g(v_2, v_3)) \wedge P(g(v_2, v_3), v_4) & \Rightarrow & P(v_1, v_4) \end{array}$$

then

(i) *there are two rules, R_1 and R_3, which are immediately applicable to ϕ and respectively remove all the occurrences of f and g,*

(ii) *R_2 is also applicable, but it can remove just one of the occurrences of g, introducing an occurrence of f as side effect,*

(iii) *R_1 removes f at the expense of introducing another occurrence of g, already present in the conjecture, and*

(iv) *R_3 does not introduce any symbol in the rewritten formula.*

The main element in the current strategy for ordering deviant symbols is the requirement that a rule cannot reintroduce a deviant symbol previously removed from the conjecture. For the above rule set, both $f \prec g$ and $g \prec f$, in which case $\langle \mathcal{R}, \overset{*}{\prec} \rangle$ is not well-founded. For this reason, R_2 should be dropped from $\mathcal{R}$, and f should be removed before g. □

Since alternate removals are not allowed by the ordering mechanism and all the occurrences of a symbol must be completely erased before others are considered, there may be a loss of completeness when there are multiple occurrences of at least one of the deviant symbols of a formula.

Example 6.3.4 *Let* ϕ

$$x^{y^2+z^3+1} + y \times z = y \times z$$

be an arithmetical conjecture, and let

$$\begin{array}{lrcl} R_1 & v_1 + v_2 = v_2 & \Rightarrow & v_1 = 0 \\ R_2 & v_1 + v_2 = 0 & \Rightarrow & v_1 = 0 \wedge v_2 = 0 \\ R_3 & v_1^{v_2} = 0 & \Rightarrow & v_1 = 0 \wedge v_2 \neq 0 \end{array}$$

be the elements of the rule set $\mathcal{R}$. *With respect to* $\mathcal{L}_{SMA}$, ϕ *has two deviant symbols,* $+$ *and* exp. *In the absence of a disagreement elimination mechanism, none of the possible orderings for this pair of symbols allows the successful conversion of* ϕ *under* $\mathcal{R}$ *to a formula of* $\mathcal{L}_{SMA}$. *However, if the removal of occurrences of* $+$ *could be interleaved with* exp, *the above rules would suffice for the reduction, considering that*

$$x^{y^2+z^3+1}\boxed{+}y \times z = y \times z \overset{R_1}{\Rightarrow} x\boxed{\exp}(y^2 + z^3 + 1) = 0 \overset{R_3}{\Rightarrow}$$

$$\overset{R_3}{\Rightarrow} x = 0 \wedge \neg((y^2 + z^3)\boxed{+}1 = 0) \overset{R_2}{\Rightarrow} x = 0 \wedge \neg(y^2\boxed{+}z^3 = 0 \wedge 1 = 0) \overset{R_2}{\Rightarrow}$$

$$\overset{R_2}{\Rightarrow} x = 0 \wedge \neg(y^2 = 0 \wedge z^3 = 0 \wedge 1 = 0)$$

□

Finally, with respect to partial *remove* rules for deviant symbols, a possible measure for the complexity of the application of a rule R is based on the deviant symbols introduced by its rhs expression. A new function, m_r, is defined as

$$m_r(R) = \sum_{S_i \in \mathcal{S}} m_d(S_i) - n_r(R)$$

where $\mathcal{S}$ denotes the multiset of deviant symbols introduced by R, $n_r(R)$ represents the number of occurrences of additional deviant symbols that R eliminates as side effect, and m_d is the measure for deviant symbols defined in section 6.3.1. The set of rules for each symbol is then put in increasing order, according to m_r.

Example 6.3.5 *Given the rule set of example 6.3.3 and the conjecture*

$$(x)(y)(P(h(x,y),g(x,y)) \wedge P(g(x,y),y))$$

there are two applicable rules,

$$\begin{array}{lrcl} R_2 & P(g(v_1, v_2), v_3) & \Rightarrow & P(f(v_1, v_2), v_3) \\ R_3 & P(v_1, g(v_2, v_3)) \wedge P(g(v_2, v_3), v_4) & \Rightarrow & P(v_1, v_4) \end{array}$$

The first one removes a single occurrence of g, at the cost of introducing an occurrence of another deviant symbol, f. The second rule removes two occurrences of the same deviant symbol without introducing any other, being therefore the preferred choice. □

One of the guidelines concerning the ordering of symbols, which involves selecting those whose rules generate the smallest possible expansion in the course of rewriting, is also relevant from the point of view of rule selection. The expansion caused by a rule can be estimated based on the number of occurrences of variables in its lhs and rhs expressions.

Example 6.3.6 *Let ϕ be the formula*

$$\underbrace{x^2 + (0 + (0 + \cdots + (0 + y^3) \ldots))}_{n \text{ occurrences of } + \text{ \& } n-1 \text{ occurrences of } 0} = 1$$

and let

$$\begin{array}{lrcl} R_1. & 0 + v & \Rightarrow & v \\ R_2. & v_1 + v_2 = 0 & \Rightarrow & v_1 = 0 \wedge v_2 = 0 \\ R_3. & v_1 + v_2 = 1 & \Rightarrow & (v_1 = 1 \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = 1) \end{array}$$

be a set of partial remove *rules for $+$. Both R_1 and R_3 are applicable to ϕ. As a matter of fact, given additional propositional and atomic rewrite rules, such as*

$$\begin{array}{llllllllllll} 0 = 0 & \Rightarrow & \top & 0 = 1 & \Rightarrow & \bot & 1 = 0 & \Rightarrow & \bot & 1 = 1 & \Rightarrow & \bot \\ \top \wedge \psi & \Rightarrow & \psi & \top \vee \psi & \Rightarrow & \top & \bot \wedge \psi & \Rightarrow & \bot & \bot \vee \psi & \Rightarrow & \psi \end{array}$$

the extended rewrite set is confluent and noetherian, and, as a result, any rewriting path leads to the same normalised formula. The size of each path, however, may vary drastically. If R_1 is chosen in the first place, it is possible to reduce ϕ to $\mathcal{L}_{SMA}$ by $n-1$ applications of this rule, followed by a single application of R_3. However, if R_3 is first chosen, followed by applications of R_2 and R_3, the total number of rewriting steps is

$$\frac{n(n+1)}{2}$$

since

Base case. $x^2 + y^3 = 1 \overset{R_3}{\Rightarrow} (x^2 = 1 \wedge y^3 = 0) \vee (x^2 = 0 \wedge y^3 = 1)$, *and*

Step case. *Given an atom where* $+$ *has* $n+1$ *occurrences, the rewriting sequence has the form*

$$\overbrace{x^2 + (0 + (0 + \cdots + (0 + y^3) \ldots))}^{n+1 \text{ occurrences of } +} = 1$$

$$\Downarrow R_3$$

$$(x^2 = 1 \wedge \underbrace{(0 + (0 + \cdots + (0 + y^3) \ldots))}_{n \text{ occurrences of } +} = 0) \vee (x^2 = 0 \wedge \underbrace{(0 + (0 + \cdots + (0 + y^3) \ldots))}_{n \text{ occurrences of } +} = 1)$$

The first disjunct requires n *additional applications of* R_2 *till all occurrences of* $+$ *are eliminated, whereas, for the second disjunct, it is assumed as inductive hypothesis that* $\frac{n(n+1)}{2}$ *steps are required. The total number of steps in the presence of* $n+1$ *occurrences of* $+$ *then is*

$$1 + n + \frac{n(n+1)}{2} = \frac{2(n+1) + n(n+1)}{2} = \frac{(n+2)(n+1)}{2} = \frac{(n+1)((n+1)+1)}{2}$$

Hence, for the first path, the number of steps varies linearly with n*, whereas for the second path, it depends on* n^2*.* □

In the above example, R_1 and R_2 both preserve variables and their number of occurrences, while the rhs expression of R_3 has twice as many occurrences of variables as its lhs. Any increase in the number of occurrences of variables indicates the potential introduction of new occurrences of deviant symbols during rewriting, a factor that, according to the definition of the measure function m_r, should be also taken into account. This, however, would require the inspection of the conjecture itself, and the establishment of which variables of the rule must be instantiated to terms that contain deviant symbols. The additional matching step could be justified on the grounds that a shorter rewriting path may be eventually identified.

6.3.3 Decidable Sub-subclasses

From a strictly logical point of view, given a recursive extension Σ of a decidable sublanguage $\mathcal{L}'$ for an undecidable theory T in $\mathcal{L}$, any decision procedure for the decidable subtheory $T \cap Fml_{\mathcal{L}'}$ suffices to decide a formula of the extended subclass, once it has been rewritten into $\mathcal{L}'$. Decision procedures in this context are only extensionally relevant, and the actual process of reducing a sentence to a propositional constant is immaterial. For this reason, attention has been entirely focused on decidable subclasses rather than on possible procedures for them.

When the complexity of the overall decision process is taken into account, however, specific features of each procedure have to be examined before a class is selected. The complexity of a class Σ can be included in the measure function m_c under the form of an estimate for the complexity of the most efficient procedure available for Σ. The new measure then determines the expected number of transformation steps for the reduction of a sentence into a propositional constant, rather than just into an element of a chosen class.

Besides influencing the choice of a decidable subclass, efficiency constraints can affect the number of such classes and the selection of *remove* rules as well. For instance, whenever a subclass Σ' of a decidable class Σ admits more efficient procedures than Σ itself, Σ' may be included as an independent choice amongst the other classes. This is the case of certain decidable theories where it is simpler to assess the validity of a quantifier-free formula than of any other formula of the underlying language. Concerning *remove* rules, whenever the number of occurrences of quantifiers in a formula affects the complexity of the decision process, the introduction of quantifiers is a source of inefficiency. For such cases, priority must be given to rules whose rhs expressions are free from quantifiers.

The complexity of a procedure therefore interferes in three ways with the selection mechanisms described in the previous sections. Firstly, the measure function m_c has to be modified to take into account the complexity of the most efficient procedure for each subclass. Secondly, the number of decidable subclasses available to the system may increase due to the inclusion of sub-subclasses for which less complex procedures

are available. In the case of PA, it would then be possible to distinguish between reducing a formula to $\mathcal{L}_{PrA}$ or to the quantifier-free class of this sublanguage; for the new subclass, two logical symbols, $\forall$ and $\exists$, become deviant. Finally, the measure function for *remove* rules, m_r, has also to take into account the efficiency loss that results from the introduction of new occurrences of quantifiers in the rewritten formula, even when quantifiers do not represent deviant symbols w.r.t. the chosen class. The impact of the introduction of other symbols upon efficiency could be also examined for each particular theory[18].

6.4 Conclusions

In the course of the reduction of a formula into a decidable sublanguage, the removal of deviant symbols has to be performed at the conjecture level, and can be restricted to the application of *remove* rules for deviant symbols. The elimination of disagreements between rules and formulae, on the other hand, may be entirely addressed at the rule level. As a result, general-purpose proof plans for the extension of decidable sublanguages can be built exclusively upon the method *remove* and its various versions.

Normalisation plans can impose certain constraints upon the application of rewrite rules to prevent exhaustive rewriting: rules are first classified according to the syntactic operation they perform, and their application is then determined by a particular sequence of tasks specified by the plan. Other constraints, involving aspects such as the selection of decidable sublanguages, the ordering of deviant symbols and the ordering of *remove* rules, can be added to the initial control setting by means of heuristic functions.

The only missing element that still prevents general-purpose plans from being built is a generic mechanism of context transformation that eliminates fatal disagreements and supports the use of partial *remove* rules. This question is addressed in the coming two chapters.

---

[18] None of these three aspects has been fully addressed by the currently implemented heuristic functions. Chapter 9 though examines the complexity of decision procedures for arithmetical sublanguages and its effect on the selection of subclasses. These topics are further discussed in chapter 11.

Chapter 7

Difference Reduction Procedures

Special-purpose proof plans that reproduce decision procedures for theories such as *DAG* incorporate two major operations, the *elimination of disagreements* between conjectures and *remove* rules, and the *removal of deviant symbols* w.r.t. a particular decidable subclass. Although a specific procedure to eliminate disagreements has been built from the composition of primitive normalisation tactics, it is not suitable for a proof plan that represents a generic reduction function for symbol removal.

General disagreement elimination mechanisms can be built on top of any *difference reduction* procedure, which provides control for the application of paramodulation and other rules involving equality. *E-resolution* is described in section 7.1, followed by *RUE-resolution* in section 7.2 and *ECOP* in section 7.3. In section 7.4, E-resolution, RUE-resolution and ECOP are compared from the point of view of completeness and efficiency.

7.1 E-resolution

Paramodulation usually involves a large search space when employed in conjunction with resolution and factoring. To reduce the number of possible applications of the paramodulation rule, *E-resolution* restricts its domain of usage to pairs of potentially complementary literals[1]. Let Ω be a set of clauses to which two potentially resolvable elements,

[1] E-resolution is described in [Morris 69]. Paramodulation is defined in appendix D.1.

$$p(t_1, \ldots, t_n) \vee C_1 \qquad \neg p(s_1, \ldots, s_n) \vee C_2$$

belong. If they are unifiable and σ is their most general unifier (mgu), the corresponding E-resolvent coincides with an ordinary resolvent, $\sigma C_1 \vee \sigma C_2$. Otherwise there are only partial unifiers for them, i.e. given any substitution σ', $\sigma' t_i \stackrel{s}{\neq} \sigma' s_i$, for some i. A disagreement set $\mathcal{D} = \{\langle u_1, u_1' \rangle, \ldots, \langle u_m, u_m' \rangle\}$ for $\langle \sigma' p(t_1, \ldots, t_n), \sigma'(\neg p(s_1, \ldots, s_n)) \rangle$ can then be formed, and the expression

$$\sigma' C_1 \vee \sigma' C_2 \vee \bigvee_{j=1}^{m} (u_j \neq u_j')$$

is supplied to a side module of the mechanism, the *equality tree generator*[2]. No form of search control is available in this module, which relies on the exhaustive application of inference rules. If it solves all disagreements, the E-resolvent, defined as

$$\sigma'(\sigma C_1) \vee \sigma'(\sigma C_2) \vee \bigvee_{k} \delta_k$$

includes the residual literals δ_k introduced by paramodulation (whenever conditional equations have been employed) and the compound substitution σ' generated in the course of the removal of inequalities. As it prevents the indiscriminate introduction of paramodulated clauses in the refutation tree, the mechanism avoids unnecessarily expanding the derived clause set.

To ensure that the generation of equality trees halts, an upper bound for their depth, which is not sensitive to the input clause set, is determined in advance. If any disagreement is still left after the maximum tree size is reached, no E-resolvent is generated. Further restrictions may be imposed on other parameters as well. For instance, the *restricted* form of E-resolution takes into account only the left most general partial unifier (mgpu) for complementary literals, and the innermost disagreement set. Such restrictions, however, make the mechanism incomplete[3].

---

[2] Partial unifiers and disagreement sets are defined in appendix D. An overview of the difference reduction strategy is given in appendix D.1.

[3] See [Anderson 70].

Example 7.1.1 *Let ϕ be the conjecture*

$$a = b \wedge d = c \wedge (y)p(a, f(b,c), y)$$
$$\supset$$
$$(\exists x)p(x, f(x,d), e)$$

A constructive proof for the validity of ϕ must identify an instance of $p(x, f(x,d), e)$ which is a logical consequence of the hypotheses $\{a = b, d = c, (y)p(a, f(b,c), y)\}$, given that, for any finite set of sentences $\tau_1, \ldots, \tau_n$,

$$\models (\tau_1 \wedge \ldots \tau_n) \supset \psi \quad \textit{iff} \quad \{\tau_1, \ldots, \tau_n\} \models \psi$$

For this purpose, the negation of ϕ is first put into clausal form, and a set of clauses,

$$\Omega \quad = \quad \{a = b, d = c, p(a, f(b,c), y), \neg p(x, f(x,d), e)\}$$

is obtained. Since $p(a, f(b,c), y)$ and $\neg p(x, f(x,d), e)$ are the only complementary literals in Ω, they represent a candidate pair for the possible derivation of an E-resolvent. If the restricted version of the mechanism is chosen, given that the candidate pair is not unifiable, its left mgpu, $\sigma = \{^a/_x, ^e/_y\}$, is applied, and the innermost disagreement set,

$$\mathcal{D} = \{\langle b, a\rangle, \langle c, d\rangle\}$$

is obtained. An E-resolvent is derived as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{p(a, f(b,c), y) \qquad \neg p(x, f(x,d), e)}{b \neq a \vee c \neq d} \quad a = b}{b \neq b \vee c \neq d} \quad z = z}{c \neq d} \quad d = c}{c \neq c} \quad w = w}{\square}$$

Since all the inequalities have been raised, the resulting clause, $\square$, is an E-resolvent for the pair of complementary literals. All the intermediate steps are then erased. □

In summary, E-resolution has three main advantages with respect to the plain application of paramodulation.

(i) paramodulation is used only with the purpose of eliminating fatal disagreement

between complementary literals,

(ii) paramodulated clauses obtained in the course of the derivation of an E-resolvent are not all included in the main refutation tree, thus reducing the number of derived clauses, and

(iii) an additional rule present in E-resolution dispenses with the introduction of any equality axioms in the clause set.

The reduction of differences between terms, therefore, takes place only under the form of complete disagreement elimination.

7.2 Resolution by Unification and Equality

Even though inference steps in E-resolution require the complete elimination of disagreements between complementary literals, other procedures for the reduction of differences allow the introduction of intermediate resolvents as well. For instance, given a clause set Ω that contains $p(t_1, \ldots, t_n) \vee C_1$ and $\neg p(s_1, \ldots, s_n) \vee C_2$, it is sound to introduce an intermediate resolvent,

$$(\sigma t_1 = \sigma s_1 \wedge \ldots \wedge \sigma t_n = \sigma s_n) \supset (\sigma C_1 \vee \sigma C_2)$$

or, in clausal form,

$$\sigma t_1 \neq \sigma s_1 \vee \ldots \vee \sigma t_n \neq \sigma s_n \vee \sigma C_1 \vee \sigma C_2 \quad (*)$$

in the refutation tree, where σ is a partial unifier (not necessarily a mgpu) for the pair of literals. The desired resolvent, $\sigma C_1 \vee \sigma C_2 \vee D$, may or may not be later generated, depending on whether conditional equalities $D_i \vee (\sigma t_i = \sigma s_i)$ are derivable from clauses of Ω and resolved against $(*)$. $D \stackrel{s}{=} (D_1 \vee \cdots \vee D_n)$ is the residue clause generated in the course of the elimination of inequalities.

The presence of intermediate resolvents is one of the main features of *resolution by unification and equality* (RUE-resolution). Two new inference rules replace binary resolution and paramodulation. There are three versions for the procedure, each of which incorporates increasingly stronger control constructs[4].

[4] See [Digricoli & Harrison 86], p. 258 - 262.

7.2.1 Weak Version

Due to the presence of equality in a theory, resolvents may include the disjunction of inequalities obtained as a side effect of a resolution step. Inequalities may later be decomposed into subproblems, according to a chosen disagreement set. The NRF and RUE rules of inference respectively cover these cases. Additional concepts such as *RUE factors* are required to insert other properties of equality, e.g. symmetry, in the inference system.

Definition 7.2.1 (RUE factor)

Let $\ell_1(t_1, \ldots, t_n) \vee \ell_2(s_1, \ldots, s_n) \vee C$ *be a clause where* ℓ_1, ℓ_2 *are literals that contain the same predicate symbol and have the same sign.*

i. If ℓ_1, ℓ_2 *are non-equational literals, a* RUE factor *for the above clause has the form*

$$\ell_1(t_1, \ldots, t_n) \vee C \vee D$$

where D *is a disjunction of inequalities generated from a disagreement set* $\mathcal{D}$ *for the pair* $\langle \ell_1(t_1, \ldots, t_n), \ell_2(s_1, \ldots, s_n) \rangle$.

ii. If ℓ_1, ℓ_2 *are equational literals, e.g.* $t_1 = t_2$ *and* $s_1 = s_2$, *a* RUE factor *for* $t_1 = t_2 \vee s_1 = s_2 \vee C$ *is any clause of one of the forms*

$$t_1 = t_2 \vee C \vee t_1 \neq s_1 \vee t_2 \neq s_2$$
$$t_1 = t_2 \vee C \vee t_2 \neq s_1 \vee t_1 \neq s_2$$

Definition 7.2.2 (RUE/NRF Rules of Inference – Open Form)

i. A RUE resolvent *for the clauses* $p(t_1, \ldots, t_n) \vee C_1$ *and* $\neg p(s_1, \ldots, s_n) \vee C_2$, *where* p *is not equality, is a clause*

$$\sigma C_1 \vee \sigma C_2 \vee D$$

where σ *is a substitution for the free variables of both clauses, and* D *is a disjunction of inequalities generated from a disagreement set for the complementary pair* $\langle \sigma p(t_1, \ldots, t_n), \neg\, \sigma p(t_1, \ldots, t_n) \rangle$.

ii. *A* RUE resolvent *for the clauses* $(t_1 = t_2) \vee C_1$ *and* $(s_1 \neq s_2) \vee C_2$ *is either of the formulae,*

$$t_1 \neq s_1 \vee t_2 \neq s_2 \vee C_1 \vee C_2$$
$$t_2 \neq s_1 \vee t_1 \neq s_2 \vee C_1 \vee C_2$$

iii. *A* NRF (negative reflective function) resolvent *for the clause* $(t \neq s) \vee C$ *is a clause*

$$\sigma C \vee D$$

where σ *is a substitution for the free variables of* $(t \neq s) \vee C$, *and* D *is a disjunction of inequalities generated from a disagreement set for the pair* $\langle \sigma t, \sigma s \rangle$.

iv. *A RUE resolvent for* $p(t_1, \ldots, t_n) \vee C_1$ *and* $\neg p(s_1, \ldots, s_n) \vee C_2$ *is* minimal *iff the associated substitution* σ *minimises the number of disagreements between* $p(t_1, \ldots, t_n)$ *and* $p(s_1, \ldots, s_n)$.

v. *A NRF resolvent for* $(t \neq s) \vee C$ *is* minimal *iff the associated substitution* σ *minimises the number of disagreements between* t *and* s.

The NRF rule makes redundant the inclusion of the reflexive axiom for equality in the clause set. For instance, when it is applied to the clause

$$t \neq t$$

since, for any substitution σ, the only disagreement set for $\langle \sigma t, \sigma t \rangle$ is the empty set, the NRF resolvent is the empty clause, hence

$$\frac{t \neq t}{\square}$$

From the viewpoint of difference reduction, the NRF rule has the purpose of lowering disagreement sets. Hence, if the topmost disagreement set for a pair of complementary literals is chosen in the first place, any other disagreement set can be subsequently accessed by its application. This rule, as well as the RUE rule, is in *open* form given that the substitution σ and the disagreement set $\mathcal{D}$ are left unspecified. Both rules can be integrated into a complete inference system for the predicate calculus with equality.

Definition 7.2.3 (RUE-NRF Deduction)

i. Given a pair of clauses, C_1 *and* C_2*, an* extended RUE-resolvent *of* C_1 *and* C_2 *is a RUE-resolvent of* C_1' *and* C_2'*, where* C_i' *is either* C_i *or one of its RUE factors.*

ii. An extended NRF-resolvent *of a clause* C *is a NRF resolvent of* C *or of one of its factors.*

iii. Given a set of clauses Ω*, a* RUE-NRF deduction *of a clause* C *from* Ω *is a finite sequence of clauses*

$$C_1, \ldots, C_n$$

in which C_i *is either an element of* Ω*, or an extended RUE resolvent of a pair of preceding clauses in the sequence, or an extended NRF resolvent of a preceding clause, for* $1 \leq i \leq (n-1)$*, and* $C_n \stackrel{s}{=} C$*.*

Theorem 7.2.1 [Digricoli & Harrison 86] *If* Ω *is an unsatisfiable set of clauses in the predicate calculus with equality, there is a* RUE-NRF *deduction of the empty clause from* Ω*.*

7.2.2 Strong Version

There are guidelines for the selection of partial unifiers, disagreement sets and elimination equalities (i.e. those used to raise fatal disagreements between semantically or Ω-unifiable terms) which do not affect the completeness of the original inference system. They are based on a few additional concepts, which are introduced next[5].

Definition 7.2.4 (Viable disagreement set)

Let Ω *be a set of (conditional) equalities of a first-order language* $\mathcal{L}$*.*

[5] *Semantic unification* is discussed in [Dershowitz & Jouannaud 90], p. 282-4.

i. A disagreeing pair of terms[6] $\langle t_1, t_2 \rangle$ *is* potentially eliminable *w.r.t.* Ω *iff either* t_1 *and* t_2 *are unifiable, or there are (conditional) equations* $\phi_1 \vee s_1 = s_2$ *and* $\phi_2 \vee s_3 = s_4$ *in* Ω *and there are pairs* $\langle t_1, s_j \rangle$ *and* $\langle t_2, s_k \rangle$*, for some* $j, k, 1 \leq j, k \leq 4$*,* $j \neq k$*, each of which satisfies one of the conditions*

(a) t_i *unifies with* s_l*, or*

(b) $t_i \stackrel{s}{=} S(u_1, \ldots, u_n)$*,* $s_l \stackrel{s}{=} S(u'_1, \ldots, u'_n)$*, and there is a viable disagreement set for each pair* $\langle u_i, u'_i \rangle$*.*

ii. If ϵ_1 *and* ϵ_2 *are two* $\mathcal{L}$*-expressions that do not share variables, and* $\mathcal{D}$ *is a disagreement set for* $\langle \epsilon_1, \epsilon_2 \rangle$*,* $\mathcal{D}$ *is* viable *w.r.t.* Ω *iff each element of* $\mathcal{D}$ *is potentially eliminable w.r.t.* Ω*.*

Example 7.2.1 *Let*

$$\{\, f(g(a,b)) = f(b), \quad h(x) = c, \quad g(f(a), y) = f(c) \,\}$$

be a set of equations where a, b, c *are individual constants.*

i. The disagreement pair $\langle h(z), f(w) \rangle$ *is potentially eliminable, since each term of the pair is unifiable with either the rhs or lhs expression of one the equations above.*

ii. The pair $\langle f(f(z)), g(w, a) \rangle$ *is also potentially eliminable, for although* $f(f(z))$ *is not unifiable with any of the dominant terms in the equations above, it has the same dominant symbol as the term* $f(c)$*, and* $\langle f(z), c \rangle$ *is potentially eliminable.*

□

Considering that, for any pair of expressions, more than one disagreement set may be viable, the strong form of RUE-resolution selects the *topmost viable set.* There is a reduction in the search space, therefore, whenever the topmost disagreement set for a pair of expressions is not viable. When a clause set contains either a clause of the form $C \vee v_1 = v_2$, or a pair of clauses of the form $C_i \vee v_i = t_i$, $i \in \{1, 2\}$, where v_i is a

[6] Disagreeing pairs are defined in appendix D.

variable, any disagreement set is viable. In such cases, the selection criterion has no effect over the search space.

Concerning substitutions, the choice falls on the *RUE-unifier*, which is built from a mgpu $\sigma = \{^{u_1}/_{v_1}, \ldots, ^{u_m}/_{v_m}\}$ for a pair of expressions $\langle \epsilon_1, \epsilon_2 \rangle$ by the deletion of all components $^{u_i}/_{v_i}$, whenever v_i occurs in a viable disagreement set for $\langle \epsilon_1, \epsilon_2 \rangle$. For the selection of elimination equations, the *equality restriction*, which states that a RUE-resolvent can be generated from a pair of clauses

$$\langle\ s_1 \neq s_2 \vee C_1,\ \ t_1 = t_2 \vee C_2\ \rangle$$

iff s_i and t_j share the same dominant symbol, for some $i, j \in \{1, 2\}$, must be followed. The purpose of this restriction is to guarantee the generation of subproblems by a subsequent application of the NRF rule.

Example 7.2.2

i. Given the set of clauses

$$\Omega \ = \ \{p(f(x)) \vee q(x),\ \neg p(f(a)),\ b = c, f(a) = f(b)\}$$

once the pair $\langle p(f(x)) \vee q(x), \neg p(f(a)) \rangle$ is selected, its left mgpu is $\sigma = \{^a/_x\}$ and the disagreement sets for the pair, prior to partial unification, are

$$\begin{aligned} \mathcal{D}_1 &= \{\langle f(x), f(a) \rangle\} \\ \mathcal{D}_2 &= \{\langle x, a \rangle\} \end{aligned}$$

Both sets are viable: each element of the disagreeing pair in $\mathcal{D}_1$ can be unified with a distinct argument of an equation in Ω, whereas the disagreeing pair in $\mathcal{D}_2$ is unifiable. Since the variable x occurs in a viable disagreement set, it has to be dropped from the mentioned substitution to generate the RUE-unifier, which in this case is empty.

ii. For the clause set

$$\Omega \ = \ \{f(x, a) \neq b,\ g(y) = h(y, b),\ a = f(b, c)\}$$

the only pair of clauses eligible for the application of the RUE-rule, according to the equality restriction, is $\langle f(x,a) \neq b, a = f(b,c)\rangle$, *since at least one argument of one of the complementary literals shares its dominant symbol with an argument of the other literal.* □

All three notions have been incorporated in a stronger version of the RUE and NRF inference rules.

Definition 7.2.5 (RUE/NRF Rules of Inference – Strong Form)

i. The strong RUE resolvent *for the clauses* $p(t_1,\ldots,t_n)\vee C_1$ *and* $\neg p(s_1,\ldots,s_n)\vee C_2$ *is the clause*

$$\sigma C_1 \vee \sigma C_2 \vee D$$

where σ *is the RUE unifier of* $p(t_1,\ldots,t_n)$ *and* $p(s_1,\ldots,s_n)$, *and* D *is the disjunction of inequalities generated from the topmost viable disagreement set for* $\langle \sigma p(t_1,\ldots,t_n), \neg\, \sigma p(t_1,\ldots,t_n)\rangle$.

ii. The strong NRF (negative reflective function) resolvent *for* $(t \neq s) \vee C$ *is the clause*

$$\sigma C \vee D$$

where σ *is the RUE unifier of* t *and* s, *and* D *is the disjunction of inequalities generated from the topmost viable disagreement set for the pair* $\langle \sigma t, \sigma s\rangle$.

The topmost viable disagreement set and the equality restriction preserve the completeness of the open form of RUE resolution. The effect of the RUE unifier on completeness has not been formally established yet[7].

[7] See [Digricoli & Harrison 86], p. 277.

7.2.3 Heuristic Version

Further improvements in efficiency can be obtained at the expense of completeness. Heuristic control can be introduced at all three levels in which search is present, i.e. at the selection of partial unifiers, disagreement sets and elimination equalities. Concerning partial unifiers, a mgpu is chosen in the first place, although not necessarily the left mgpu[8]. When compared to the RUE-unifier, which does not exclude any viable disagreement set in order to preserve completeness, the heuristic choice may lead to a more compact refutation tree. For instance, given the pair of terms

$$\langle f(x, g(c, h(a, x))),\ f(b, g(y, h(c, d)))\rangle$$

assuming that the RUE unifier for this pair w.r.t. a set of clauses Ω is the empty substitution, the innermost disagreement set is

$$\{\langle x, b\rangle, \langle c, y\rangle, \langle a, c\rangle, \langle x, d\rangle\}$$

If the left mgpu is chosen instead, the new set,

$$\{\langle a, c\rangle, \langle b, d\rangle\}$$

can be reached from the first one after two applications of the NRF rule, which potentially increase the size of the first refutation tree.

The choice of a disagreement set favours the lowest one that does not contain an irreducible inequality, instead of the topmost viable. Irreducible inequalities are those whose negations (or rather instances of their negations) cannot be proven from the equality base, as for instance inequalities entirely formed of Skolem constants. Finally, the selection of elimination equations depends on the syntactic difference between terms.

Definition 7.2.6 (Degree of Unification)

Let σ be the left mgpu for $(t_1 \neq t_2)$ and the elimination equation $u_1 = u_2$.

[8] See [Digricoli & Harrison 86], p. 283.

i. The degree of unification w *between the above inequality and elimination equation is defined as*

$$w(t_1 \neq t_2, u_1 = u_2) \quad = \quad w(t_1, u_1 = u_2) + w(t_2, u_1 = u_2)$$

ii. The degree of unification *between a term t_i of the above inequality, $i \in \{1,2\}$, and the elimination equation is defined as follows.*

(a) If there is a term u_j, $j \in \{1,2\}$ and a substitution σ such that $\sigma t_i \stackrel{s}{=} \sigma u_j$, then $w(t_i, u_1 = u_2) = 50$.

(b) If t_i has the form $S(t'_1, \ldots, t'_n)$ and there is a term u_j, $j \in \{1,2\}$, of the form $S(u'_1, \ldots, u'_n)$, then $w(t_i, u_1 = u_2) = 20 + 30 \times \frac{m}{n}$, where m is the number of syntactically identical pairs $\langle t'_k, u'_k \rangle$.

(c) If neither u_1 nor u_2 has the same dominant symbol as t_i, then $w(t_i, u_1 = u_2) = 0$.

The measure 100 is obtained if and only if the complementary literals are unifiable, whereas 0 is obtained when the equality restriction for the selection of elimination equations has not been observed.

Example 7.2.3 *Assume it has to be proven that*

$$a^2 \times a^2 + (a^2 b + a^2 c) = a^2(a^2 + (c + b))$$

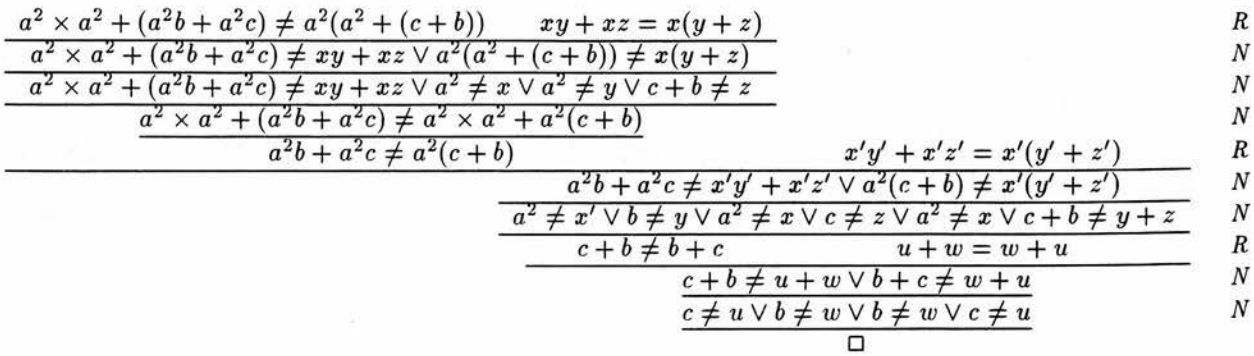
is a logical consequence of the commutativity of $+$ and the distributivity of $\times$ over $+$. The negation of the corresponding conjecture in clausal form is

$$\Omega \quad = \quad \{xy + xz = (x(y + z)),\ u + w = w + u,\ a^2 \times a^2 + (a^2 b + a^2 c) \neq a^2(a^2 + (c + b))\}$$

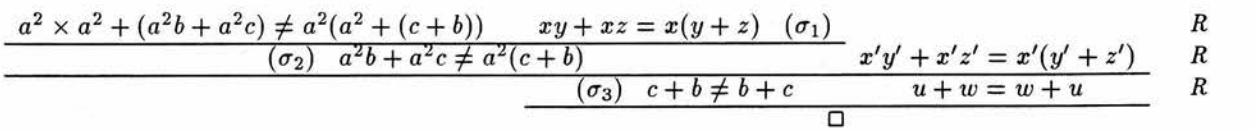
The equality restriction is satisfied e.g. for the pair

$$\langle\ a^2 \times a^2 + (a^2 b + a^2 c) \neq a^2(a^2 + (c + b)),\ xy + xz = (x(y + z))\ \rangle$$

since $a^2 \times a^2 + (a^2 b + a^2 c)$ and $xy + xz$ share the same dominant symbol. A RUE-derivation of the empty clause from Ω, both under the strong and the heuristic forms of RUE-resolution, is indicated in figure 7.1. □



Strong Form



Heuristic Form

σ_1 $\{^{a^2}/_x, ^{a^2}/_y, ^{(c+b)}/_z\}$
σ_2 $\{^{a^2}/_{x'}, ^{b}/_{y'}, ^{c}/_{z'}\}$
σ_3 $\{^{c}/_u, ^{b}/_w\}$

R RUE rule
N NRF rule

Figure 7.1: An Application of RUE-resolution

7.3 Equality Graphs

Planning in the context of equality reasoning can benefit from the usage of graphs for the representation of inference steps. In its pure form, an *equality graph* involves a pair of terms, $\langle t, u \rangle$, which is linked by means of equations,

$$\underbrace{t \longrightarrow (r_1}_{\sigma_1} = \underbrace{s_1) \longrightarrow (r_2}_{\sigma_2} = \ldots = \underbrace{s_{n-1}) \longrightarrow (r_n}_{\sigma_n} = \underbrace{s_n) \longrightarrow u}_{\sigma_{n+1}}$$

where each $r_i = s_i$ belongs to a set of hypotheses and σ_i is a unifier for each underbraced pair of terms. For notational simplicity, graphs are linearly represented as chains, whenever possible[9].

The existence of a local unifier for pairs of adjacent terms in a graph does not guarantee the existence of a substitution for the Δ-unification of $\langle t, u \rangle$, where Δ is a set of hypotheses. For instance, given the chain

$$\underbrace{f(a,b) \longrightarrow (f(x,y)}_{\sigma_1} = \underbrace{g(y,z)) \longrightarrow (g(a,z)}_{\sigma_2} = \underbrace{z) \longrightarrow c}_{\sigma_3}$$

even though each underbraced pair of terms can be respectively unified by

$$\begin{aligned} \sigma_1 &= \{^a/_x, {}^b/_y\} \\ \sigma_2 &= \{^a/_y\} \\ \sigma_3 &= \{^c/_z\} \end{aligned}$$

there is no common unifier for all three pairs. The chain, therefore, is *incompatible*. Compatible graphs correspond to executable sequences of paramodulation steps, and may lead to the resolution of non-unifiable complementary literals. There are three necessary (but not sufficient) conditions which characterise compatibility,

(a) all the component unifiers associated with each paramodulation link have to fuse into a most general unifier.

(b) access depths of terms must coincide.

(c) a graph that contains incompatible subgraphs is also incompatible[10].

---

[9] *Equality reasoning* designates any inference system for theories that contain equality; see appendix D.

[10] The access depth for a term is defined in [Bläsius 87]. The incompatibility of a graph containing incompatible subgraphs is mentioned in [Bläsius & Siekmann 88], p. 403.

In order to link the terminologies of RUE-resolution and ECOP, it is worth mentioning that viable disagreement sets, defined in section 7.2.2, and compatible graphs do not necessarily come together. Although the existence of a compatible graph linking terms t_1 and t_2 implies that $\langle t_1, t_2 \rangle$ has a viable disagreement set, the converse is not necessarily true, as indicated by the above example. Incompatible equality graphs, on the other hand, do not imply the existence of a viable disagreement set. Selecting viable disagreement sets and exhibiting equality graphs are two distinct strategies for the establishment of the Δ-unifiability of two terms: the first one proceeds in width, and actually indicates at least the first two steps of a compatible equality graph, whereas the second one operates in depth.

The *equality graph construction procedure* (ECOP) employs equality graphs to establish the unsatisfiability of sets of clauses containing equality. Unlike E- or RUE-resolution, no partial unifier is applied to the initial pair of complementary literals, and each original subproblem is solved independently from the others. Individual substitutions are checked for compatibility only at the end of the process.

Example 7.3.1 *Given the set Ω that contains the clauses*

$$\begin{array}{ccc} P(g(a,w),z) & & \neg P(f(e),b) \\ g(x,x) = h(x,b) & h(u,v) = h(v,u) & h(b,a) = f(b) \\ b = c & c = e & \end{array}$$

the pair of complementary literals, $\langle P(g(a,w),z), \neg P(f(e),b) \rangle$, determine two equalities subproblems, represented by its innermost (and sole) disagreement set,

$$\mathcal{D} \quad = \quad \{\langle g(a,w), f(e) \rangle, \langle z, b \rangle\}$$

While the second disagreement is solvable, the first one requires the construction of a chain,

$$g(a,w) \cdots\cdots f(e)$$

where the dotted line indicates that the graph is incomplete[11]*. The innermost disagreement set for these terms is simply represented as $\{\langle g, f \rangle\}$. Equations taken from Ω*

[11] See [Bläsius & Siekmann 88], p. 404-406.

are inserted in the chain with the purpose of eliminating disagreements. Amongst the possible potential graphs for $\langle g(a,w), f(e)\rangle$,

$$g(a,w)\cdot\cdot\cdot\cdot\cdot g(x,x) = h(x,b)\cdot\cdot\cdot\cdot\cdot h(b,a) = f(b)\cdot\cdot\cdot\cdot\cdot f(e)$$

is the shortest one, and is therefore the first to be examined. Three new subproblems,

$$\langle g(a,w), g(x,x)\rangle$$

$$\langle h(x,b), h(b,a)\rangle$$

$$\langle f(b), f(e)\rangle$$

are generated, and the same procedure has to be applied to each of them. In the particular case of $\langle h(x,b), h(b,a)\rangle$, *the pairs of the innermost disagreement set,* $\{\langle x,b\rangle, \langle b,a\rangle\}$, *cannot be linked by equations of* Ω. *However, a new link can be inserted in the initial graph by the introduction of a new equation between* $h(x,b)$ *and* $h(b,a)$,

$$h(x,b)\cdot\cdot\cdot\cdot\cdot h(u,v) = h(v,u)\cdot\cdot\cdot\cdot\cdot h(b,a)$$

After it is placed into the original graph,

$$g(a,w)\cdot\cdot\cdot\cdot\cdot g(x,x) = h(x,b)\cdot\cdot\cdot\cdot\cdot h(u,v) = h(v,u)\cdot\cdot\cdot\cdot\cdot h(b,a) = f(b)\cdot\cdot\cdot\cdot\cdot f(e)$$

the process is restarted for the extended version. Since subproblems are solved independently, none of the partial solutions have to be discarded. All the derived subproblems of the new chain admit a local solution, as indicated in the final graph (figure 7.2). As local unifiers can be composed into a global unifier, $\sigma = \{^{a}/_{x}, ^{a}/_{w}, ^{a}/_{u}, ^{b}/_{v}\}$, *the graph is compatible. Hence*

$$\Omega \models (\sigma g(a,w) = \sigma f(e))$$

A RUE-refutation from Ω *that corresponds to the above equality graph is*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(g(a,w),z)\quad \neg P(f(e),b)}{g(a,w)\neq f(e)\vee z\neq b}\,RUE}{g(a,w)\neq f(e)}\,NRF\ \sigma_1 \qquad g(x,x)=h(x,b)}{h(a,b)\neq f(e)}\,RUE\ \sigma_2 \qquad h(u,v)=h(v,u)}{h(b,a)\neq f(e)}\,RUE\ \sigma_3 \qquad h(b,a)=f(b)}{f(b)\neq f(e)}\,RUE}{b\neq e}\,NRF \qquad b=c}{c\neq e}\,RUE \qquad c=e}{\square}\,RUE$$

where $\sigma_1 = \{^b/_z\}, \sigma_2 = \{^a/_x, ^a/_w\}$ *and* $\sigma_3 = \{^a/_u, ^b/_v\}$.

□

One difficulty at the planning level is related to those pairs whose innermost disagreement set cannot be solved, as in the case of $\langle h(x,b), h(b,a)\rangle$ of the above example. Such cases require the expansion of the original graph by the introduction of new links and equations. It seems that their introduction can be limited in principle to the level of the immediate dominant symbol, in order to minimise the effect over other subproblems. For instance, given the disagreeing pair of terms

$$f(a, h(b, g(d)))$$

$$f(b, h(x, g(e)))$$

whose innermost disagreement set is $\mathcal{D} = \{\langle a,b\rangle, \langle b,x\rangle, \langle d,e\rangle\}$, if $\langle d,e\rangle$ cannot be raised, the set of subproblems identified by $\mathcal{D}$ has to be abandoned. ECOP would then suggest the closest alternative set, $\{\langle a,b\rangle, \langle b,x\rangle, \langle g(d), g(e)\rangle\}$. For the last pair, before global planning is resumed, new links between $g(d)$ and $g(e)$,

$$g(t_1) = F_1(t_1, \ldots, t_{n_1}) \cdot \cdots \cdot F_1(t'_1, \ldots, t'_{n_1}) = F_2(t''_1, \ldots, t''_{n_2}) \cdot \cdots \cdot$$

$$\cdots \cdot F_m(t^m_1, \ldots, t^m_{n_m}) = g(t')$$

have to be introduced. For each subproblem, the lowest set of disagreement pairs would be chosen and solved, whenever possible.

The unrestricted search for compatible graphs in ECOP has an order of magnitude similar to the search present in E- and RUE-resolution. One of the main problems in all of these approaches is the possibility of unifying variables with any term of an equality. Besides targeting variables or limiting search to the innermost disagreement set in the first instance, more complex problems may require additional heuristics, concerning

Weights and Limits. Time and size constraints upon the reduction of differences may be established by means of measures defined for the terms being transformed (e.g.

number of occurrences of certain symbols, nesting depth of their occurrences, etc). The decision of continuing or stopping the generation of a given graph is then based on the behaviour of such parameters.

Difference Measure. Additional measures may be defined to assess the level of difference between two terms. Such measures can be applied to the selection of equations, to determine whether their application reduces differences efficiently.

Conflict Solving. The global instantiation of multiply occurring variables may be prevented by incompatible local substitutions. Incompatibilities between partial unifiers may nonetheless be overcome by the construction of additional equality graphs.

7.4 Comparative Analysis

E-resolution, RUE-resolution and ECOP provide complete inference systems able to determine whether two expressions, or any of their instances, are equal under a set of hypotheses. All of them substitute semantic equality for syntactic identity at the stage of unifying expressions. From the point of view of proof strategies, they gradually eliminate disagreeing pairs (i.e. reduce differences) between non-unifiable expressions.

Search control in equality reasoning is provided along at least four lines. At the level of *global planning*, equality connection graphs supply ECOP with a mechanism for decomposing conjectures into subproblems, while E- and RUE-resolution seem to lack any similar structure. After decomposition, each subproblem is solved independently of the others, and no substitution adequate for a subproblem is propagated to any other until a solution for each of them is found. The preservation of partial solutions is one of the main advantages of ECOP over the two other procedures[12]. For *partial unification*, as already indicated, ECOP resorts to the empty substitution at the global level, and operates with most general unifiers locally, whenever the original problem does not allow full unification. The choice of a unifier in RUE-resolution, on the other hand, depends on the topmost viable disagreement set, while E-resolution may employ the left mgpu with loss of completeness.

[12] See [Bläsius & Siekmann 88], p. 408-9.

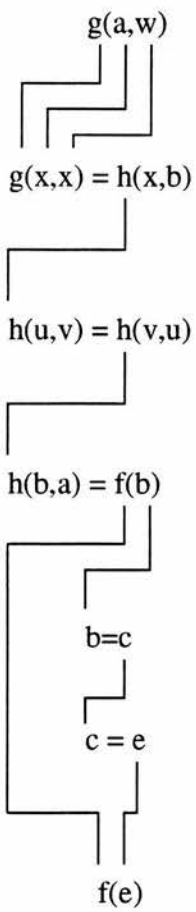


Figure 7.2: Compatible Equality Graph

Procedure	Variable Instantiation	Disagreement Set Selection	Elimination Equality Selection	Other Features
E-resolution (restricted version)	(left) mgpu	innermost	—	Incomplete Controlled use of paramodulation
E-resolution (full version)	—	—	—	Complete Controlled use of paramodulation
RUE-resolution (open form)	—	—	—	Complete RUE & NRF rules of inference (open form)
RUE-resolution (strong form)	RUE-unifier	topmost viable	equality restriction	Possibly complete RUE & NRF rules of inference (strong form)
RUE-resolution (heuristic form)	mgpu	lowest adequate	minimally disagreeing equality	Incomplete
ECOP	local mgu	innermost	minimally disagreeing equality	Independent solution for subproblems Complete Global search control

Table 7.1: Difference Reduction Procedures — Comparative Assessment

Disagreement sets in RUE-resolution are initially restricted to the topmost viable set, which is lowered by the application of the NRF rule, when necessary. ECOP starts with the lowest set, moving upwards only if required. The advantage of operating at a lower level is the comparatively smaller search space opened by the set. Concerning *elimination equalities*, there are two criteria for their selection in RUE-resolution, the equality restriction and the degree of unification, the second of which has apparently been borrowed by ECOP. These and other main properties of all three mechanisms are summarised in table 7.1.

7.5 Conclusions

The process of enlarging decidable subclasses is strengthened when mechanisms for the elimination of disagreements cooperate with the application of rewrite rules. A restricted form of disagreement elimination can be built as a special case of Δ-matching, a semantic process that involves determining whether there is an instance of a term that is equal to another one, under a set Δ of hypotheses[13].

Difference reduction procedures, on the other hand, are suitable for solving Δ-unification problems, which concern the existence of instances of an equation that are logical consequences of a set of formulae Δ. Guidance for the derivation of a refutation tree is provided along four dimensions: global planning, partial unification, disagreement set selection and elimination equation ordering. Procedures such as E-resolution, RUE-resolution and ECOP gradually remove disagreements between terms until syntactically identical expressions are obtained, whenever possible. All three approaches are complete, i.e. a proof for a Δ-unification problem (or rather a refutation for its negation) is eventually found if it is actually solvable. Termination and efficiency are affected as a result: given that the predicate calculus with equality is semidecidable, there are formulae for which the above procedures do not halt.

As even heuristic versions for these mechanisms involve a considerable amount of search in the average case, stronger control structures are required for the effective removal of disagreements in the context of rewriting. Additional control and the use of Δ-unification mechanisms for Δ-matching are discussed in the next chapter.

[13] The links between disagreement elimination and Δ-matching are examined in appendix D.4.

Chapter 8

The Rule Generation Mechanism

A generic mechanism for disagreement elimination can be derived from a difference reduction procedure once Δ-matching is recognised as a special case of Δ-unification. The *rule generation mechanism* (RGM), described in this chapter, addresses the specific problems related to formula rewriting. Section 8.1 describes the core of the mechanism and the guidelines it follows for the selection of elimination equations and partial unifiers. Section 8.2 presents a stronger version of RGM that provides a closer representation for the context transformation mechanism devised for *DAG*. A comparison between RGM and other difference reduction procedures follows in section 8.3.

8.1 Rule Generation

The inference systems examined in the previous chapter deal strictly with the unification of terms in the context of paramodulation or related rules. Nonetheless, similar procedures can be devised for expression matching as well. Fatal disagreements occur in rewriting if, given a rewrite rule $\delta \Rightarrow \delta'$ and an expression ϵ, where δ and ϵ do not share variables, δ cannot be syntactically matched against any subexpression of ϵ, i.e. there is no substitution σ or subexpression ϵ' such that $\sigma\delta \stackrel{s}{=} \epsilon'$. In the presence of a set of conditional equations Δ, these expressions may be Δ-matched, that is

$$\models \quad [\phi_1 \supset \epsilon_{1,1} = \epsilon_{2,1}] \wedge \cdots \wedge [\phi_n \supset \epsilon_{1,n} = \epsilon_{2,n}] \supset (\exists v_1) \ldots (\exists v_m)(\epsilon = \delta\{v_1, \ldots, v_m\}) \quad (*)$$

where $(\phi_i \supset \epsilon_{1,i} = \epsilon_{2,i}) \in \Delta$, $1 \leq i \leq n$. When the variables of ϵ are implicitly replaced

with individual constants, rule matching becomes an instance of unification[1].

Lemma 8.1.1 *Let $\epsilon(v_1, \ldots, v_n)$ be a $\mathcal{L}$-expression, and let $a_1, \ldots, a_n$ be individual constants that do not belong to $\mathcal{L}$. If R. $\delta_1 \Rightarrow \delta_2$ is a rewrite rule that does not share variables with ϵ, then*

$$\epsilon(v_1, \ldots, v_n) \stackrel{R}{\Rightarrow} \epsilon' \quad \textit{iff} \quad \sigma\epsilon(v_1, \ldots, v_n) \stackrel{R}{\Rightarrow} \sigma\epsilon'$$

where $\sigma = \{^{a_1}/_{v_1}, \ldots, ^{a_n}/_{v_n}\}$.

PROOF.

If R is applicable to ϵ, there must be a substitution τ and a subexpression $\widehat{\epsilon}$ of ϵ such that $\tau\delta_1 \stackrel{s}{=} \widehat{\epsilon}$. Then

$$\sigma\widehat{\epsilon} \stackrel{s}{=} \sigma(\tau\delta_1) \stackrel{s}{=} (\sigma\tau)\delta_1,$$

where $\sigma = \{^{a_1}/_{v_1}, \ldots, ^{a_n}/_{v_n}\}$. Hence there is a substitution, $\sigma\tau$, which matches R against $\sigma\widehat{\epsilon}$, a subexpression of $\sigma\epsilon$. Since $\epsilon \stackrel{R}{\Rightarrow} \epsilon'$ and $\epsilon' \stackrel{s}{=} \epsilon[\![^{\tau\delta_2}/_{\widehat{\epsilon}}]\!]$, then $\sigma\epsilon \stackrel{R}{\Rightarrow} \sigma\epsilon'$.

If R is applicable to $\sigma\epsilon$, there is a substitution τ and a subexpression $\sigma\widehat{\epsilon}$ of $\sigma\epsilon$ such that $\sigma\widehat{\epsilon} \stackrel{s}{=} \tau\delta_1$. Considering that none of the new constants a_i occurs in $\epsilon(v_1, \ldots, v_n)$, ϵ is syntactically derivable from $\sigma\epsilon$ through the replacement of each occurrence of a_i with v_i, an operation that can be applied to τ as well. Then

$$\widehat{\epsilon} \stackrel{s}{=} (\sigma\widehat{\epsilon})^* \stackrel{s}{=} (\tau\delta_1)^* \stackrel{s}{=} \tau^*\delta_1,$$

where δ^* is the expression that results from δ after the application of the replacement operation described above. Therefore R is also applicable to ϵ. ∎

Disagreement elimination procedures built on top of a difference reduction mechanism have to take two additional elements into account, (i) the selection of a subexpression

[1] Expression (*) above is actually a special case of the equational conjecture described in appendix D.1, provided that variables that occur in ϵ are regarded as new individual constant symbols.

from the conjecture and (ii) syntactic type checking for the selected subexpression. In the context of resolution, inference rules (with the exception of the NRF rule) apply only to complementary pairs of literals, and unification is limited to the corresponding pairs of arguments. The application of a rewrite rule, on the other hand, requires examining all subexpressions of an expression against which it has to be matched, hence an additional search component is present. Also, whereas resolution in first-order languages involves only the unification of terms, rewriting may involve either terms or formulae. Its representation in a resolution-based framework requires either the introduction of syntactic sorts such as *terms* and *formulae*, or the identification of two distinct processes, *term* and *formula rewriting*.

The *rule generation mechanism* (RGM), derived from both RUE-resolution and ECOP, addresses these and other specific requirements of Δ-matching, where Δ is a set of (conditional) equations. Besides matching a rule R against a conjecture, it also extracts from the rewriting process the most general applicable rule, which is Δ-equivalent to an instance of R. Also, in face of the inefficiency of complete mechanisms for difference reduction, even in the presence of the heuristic components, new control features have been added to prune the search tree even further[2]. Besides selecting disagreement sets, elimination equations and partial unifiers, RGM also chooses fatal disagreeing pairs for each disagreement set. Variable instantiation is restricted to three cases,

(a) multiply occurring variables,

(b) explicit introduction of disagreement pair in the elimination equation, and

(c) unification of elimination equations.

Finally, selected elimination equations have to preserve the upper context of a rule and cannot reintroduce previously eliminated disagreements.

8.1.1 Equation Selection

RGM adopts as a general guideline for the selection of elimination equations the requirement that a fatal disagreement has to be removed in each transformation step. It deals with both term and formula rewriting, applying therefore oriented equations and

[2] Search problems faced by difference reduction procedures are briefly described in [Cleve & Hutter 93], p. 1.

equivalences. The term *elimination equation* refers then to both types of expressions, which may be uniformly denoted as

$$\delta_1 = \delta_2$$

where $=$ represents either the equality or the biconditional symbol, depending on the syntactic type of δ_i. Given a disagreement pair and a disagreeing pair of expressions, an adequate elimination equation has to meet requirements that include its *partial applicability.*

Definition 8.1.1 (Upper Context)

Let ϵ be an expression where a symbol S occurs at position p.

i. The list $A(S, p, \epsilon)$ of upper symbols *for the occurrence of S at p in ϵ is recursively defined as follows.*

$$\begin{aligned} A(S, [\,], S) &= [\,] \\ A(S, [\,], S(t_1, \ldots, t_n)) &= [\,] \\ A(S, [i|p'], S'(\epsilon_1, \ldots, \epsilon_n)) &= [S'|A(S, p', \epsilon_i)] \end{aligned}$$

ii. The depth *of the occurrence of S at p in ϵ is the number of elements of $A(S, p, \epsilon)$ increased by one. The* depth *of a subexpression ϵ' at position p in ϵ is defined as the depth of its dominant symbol in ϵ.*

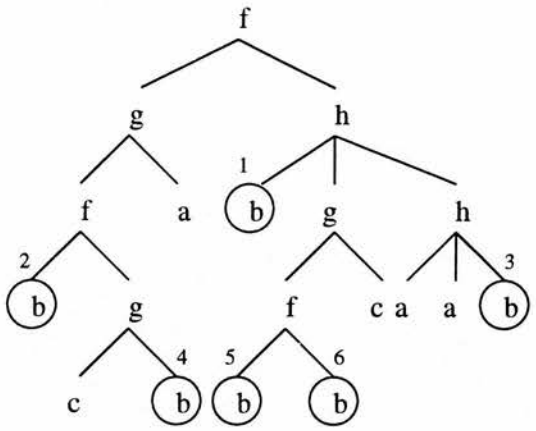
iii. The layer of depth d in ϵ, *denoted as $\ell(\epsilon, d)$, is the ordered (from left to right) list of symbols at depth d in ϵ.*

iv. If S occurs at depth d in ϵ, the upper context *of S in ϵ is the initial fragment of ϵ containing all its symbols of depth less than d.*

Example 8.1.1 *The term t,*

$$f(g(f(b, g(c, b)), a),\ h(b(g(f(b, b), c), h(a, a, b))))$$

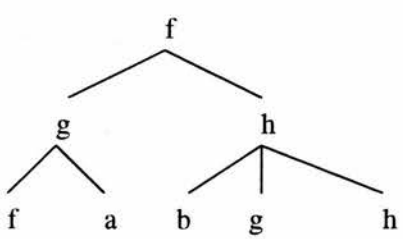
can be also represented as a tree of symbols,



i. As indicated in the above tree, there are six occurrences of the individual constant symbol b in t, each of which has a specific list of upper symbols.

occurrence	*upper symbol list*	*depth*
1	$[f, h]$	3
2	$[f, g, f]$	4
3	$[f, h, h]$	4
4	$[f, g, f, g]$	5
5	$[f, h, g, f]$	5
6	$[f, h, g, f]$	5

ii. The upper context for the second occurrence of b in t is



□

Definition 8.1.2 (Adequate Elimination Equation)

Let $\langle \epsilon_1, \epsilon_2 \rangle$ be a disagreement pair, where ϵ_2 is variable-free, and let $\langle S_1(\boldsymbol{u_1}), S_2(\boldsymbol{u_2}) \rangle$ be a fatal disagreeing pair between them.

i. An equation $\delta_1 = \delta_2$ is an adequate elimination equation *for $\langle \epsilon_1, \epsilon_2 \rangle$ with respect to S_1 and S_2 iff*

(a) *δ_1 is partially applicable to ϵ_1 w.r.t. S_1.*

(b) *S_1 has an occurrence in δ_1 at the same position as S_2 in δ_2.*

(c) *the upper context of S_1 in δ_1 is identical to the upper context of S_2 in δ_2.*

(d) *if ϵ_1 has a subexpression $S(t_1, \ldots, t_n)$ at position p, such that S was introduced as a result of a fatal disagreement elimination, then δ_1 has an occurrence of S at position p (i.e. symbols at positions corresponding to previously eliminated disagreements are not changed).*

ii. *An equation $\delta_1 = \delta_2$ is* partially applicable *to an expression ϵ w.r.t. a symbol S iff*

(a) *S has an occurrence in ϵ at the same position it occurs in δ_1.*

(b) *there is an instance δ_1' of δ_1 such that S has the same upper context in both ϵ and δ_1'.*

In particular, if S is the dominant symbol of both δ_1 and ϵ, then $\delta_1 = \delta_2$ is partially applicable to ϵ w.r.t. S. Also, any equation of the form

$$S_1(t_1, \ldots, t_{n_1}) \quad = \quad S_2(t_1', \ldots, t_{n_2}')$$

is an adequate elimination equation w.r.t. S_1 and S_2 for any pair of expressions of the form $\langle S_1(\boldsymbol{u_1}), S_2(\boldsymbol{u_2})\rangle$. Finally, whenever δ_1 is partially applicable to ϵ w.r.t. S, but δ_1 and ϵ are not unifiable, fatal disagreements are restricted to symbols of depth equal or greater than the depth of S in ϵ.

When an adequate elimination equation $\delta_1 = \delta_2$ is applicable to an expression ϵ, the disagreeing occurrence of S_1 in ϵ can be effectively replaced with S_2. Since upper contexts are preserved, there is no risk of introduction of higher disagreements in the course of such replacement. Also, as no disagreement can be introduced at a position from which another has already been eliminated, the search for new ones can be directed strictly to the same or lower layers w.r.t. the replaced occurrence of S_1.

Equation selection in RGM can be summarised as follows. Given a pair of disagreeing expressions, $\langle \delta, \psi \rangle$, a list $\{E_1, \ldots, E_n\}$ of adequate elimination equations is built, and

one of them, E_i, is selected for the removal of a fatal disagreement. If (the lhs or rhs expression of) E_i is unifiable with δ, δ is rewritten to δ' and a disagreeing pair is consequently removed. δ' is then checked for matchability w.r.t. ψ, and, in the negative case, a new subproblem, $\langle \delta', \psi \rangle$, has to be tackled. The restrictions established for equation selection prevent the reintroduction of previously removed disagreements.

Example 8.1.2 *Let ϕ be the arithmetical conjecture*

$$x^2 \neq 0 \quad \supset \quad x^2 \times x^2 \boxed{+} (x^2 \times y + x^2 \times y) = x \times x$$

R be the following remove *rule for $+$,*

$$v_1 + v_2 = 1 \quad \Rightarrow \quad (v_1 = 1 \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = 1)$$

and Δ be the set that contains the equations

$$\begin{array}{rcll} v_1 \times v_2 = v_1 & \equiv & v_2 = 1 & (v_1 \neq 0) \\ v_1 \times (v_2 + v_3) & = & v_1 \times v_2 + v_1 \times v_3 & \\ v_1 \times v_2 & = & v_2 \times v_1 & \\ (v_1 \times v_2) \times v_3 & = & v_1 \times (v_2 \times v_3) & \end{array}$$

Since R cannot remove the marked occurrence of $+$ in ϕ, there is a fatal disagreement between its lhs expression, δ, and every subexpression of ϕ of the same syntactic type as δ, i.e. ϕ itself and ψ, defined as $x^2 \times x^2 + (x^2 \times y + x^2 \times y) = x \times x$. With respect to the second formula, the disagreement sets are

$$\begin{array}{rcl} \mathcal{D}_1 & = & \{\langle x^2 \times x^2 + (x^2 \times y + x^2 \times y) = x \times x, v_1 + v_2 = 1 \rangle\} \\ \mathcal{D}_2 & = & \{\langle x^2 \times x^2 + (x^2 \times y + x^2 \times y), v_1 + v_2 \rangle, \langle x \times x, 1 \rangle\} \\ \mathcal{D}_3 & = & \{\langle x^2 \times x^2, v_1 \rangle, \langle x^2 \times y + x^2 \times y, v_2 \rangle, \langle x \times x, 1 \rangle\} \end{array}$$

The innermost disagreement set, $\mathcal{D}_3$, is chosen in the first place. The only fatal disagreeing pair is $\langle x \times x, 1 \rangle$, hence the mechanism tries to establish the equality $x \times x = 1$ from Δ. Given that variables of ϕ are dealt with as individual constants, they cannot be instantiated to reduce differences, and $x \times x = 1$ cannot be proven from the set of equations. $\mathcal{D}_2$ is excluded for the same reasons, and the topmost disagreement set, $\mathcal{D}_1$, has to be selected instead. Although $\langle x \times x, 1 \rangle$ still is the only fatal disagreeing pair, the subproblem to be tackled is

$$\Delta \models (\exists v_1)(\exists v_2)((x^2 \times x^2 + (x^2 \times y + x^2 \times y) = x \times x) \equiv (v_1 + v_2 = 1))$$

which can be schematically represented as

$\delta.$	v_1	$+$	v_2	$=$	$\boxed{1}$	
$\psi.$	$(x^2 \times x^2)$	$+$	$(x^2 \times y + x^2 \times y)$	$=$	x	$\boxed{\times}x$

The (conditional) equivalence $\delta_1 \rightleftharpoons \delta_2$,

$$(v_3 \times v_4) \times v_5 = v_3 \times v_4 \quad \equiv \quad v_5 = 1 \quad (v_3 \times v_4 \neq 0)$$

which is in fact an instance of an element of Δ,

$$v_1 \times v_2 = v_1 \quad \equiv \quad v_2 = 1 \quad (v_1 \neq 0)$$

satisfies the conditions for elimination equations, since

(a) δ_2 *is partially applicable to* δ *w.r.t. the marked occurrence of* 1 *in* δ, *as* 1 *occurs in the same position in both* δ *and* δ_2, *and the upper context for each of these occurrences, formed by a single symbol* $(=)$, *is the same,*

(b) *the function symbol* $\times$ *occurs in* δ_1 *at the same position as the constant* 1 *occurs in* δ_2, *and*

(c) $\times$ *is in the immediate scope of an equality in* δ_1, *and the same applies to the occurrence of* 1 *in* δ_2, *therefore the upper contexts are identical, as illustrated in figure 8.1.*

Hence, if the equivalence above becomes applicable to δ, *the marked occurrence of* 1 *is replaced with* $\times$, *and the fatal disagreeing pair between* δ *and* ψ *is raised. Since* δ *and* δ_2 *can be syntactically unified, with mgu* $\sigma_1 = \{(v_1 + v_2)/v_5\}$, δ *is replaced with* $\delta' \stackrel{s}{=} \sigma_1\delta_1$, *thus generating a new rule,*

$$R'. \quad v_3 \times v_4 \neq 0 \quad \rightarrow \quad (v_3 \times v_4) \times (v_1 + v_2) = v_3 \times v_4 \quad \Rightarrow \\ \Rightarrow \quad (v_1 = 1 \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = 1)$$

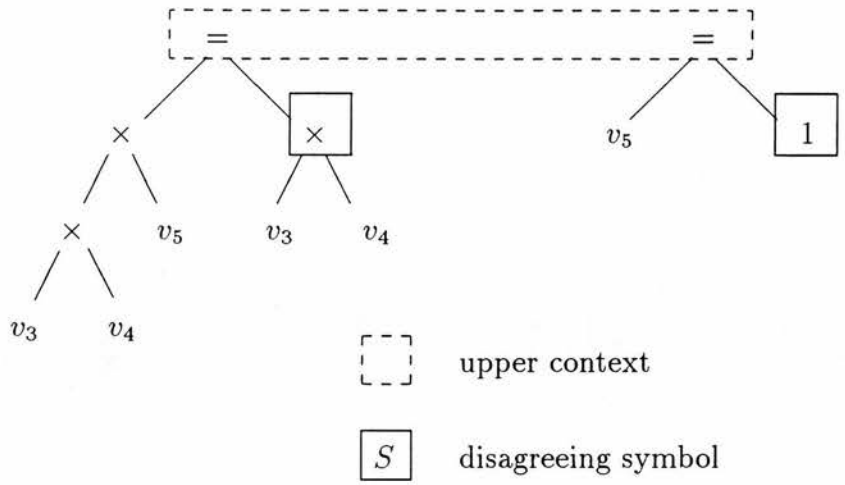


Figure 8.1: Example of an adequate elimination equation

δ' is still not matchable against ψ, therefore there must be a fatal disagreement between them. The same process has to be applied to the new pair of expressions.

$\delta'.$	$(v_3 \times v_4)$	$\boxed{\times}$	$(v_1 + v_2)$	$=$	v_3	$\times$	v_4
$\psi.$	$(x^2 \times x^2)$	$\boxed{+}$	$(x^2 \times y + x^2 \times y)$	$=$	x	$\times$	x

The innermost disagreement set,

$$\mathcal{D} \;=\; \{\langle x^2 \times x^2 + (x^2 \times y + x^2 \times y), (v_3 \times v_4) \times (v_1 + v_2)\rangle,\ \langle x^2, v_3 \times v_4\rangle\}$$

has a sole fatal disagreeing pair, $\langle x^2 \times x^2 + (x^2 \times y + x^2 \times y), (v_3 \times v_4) \times (v_1 + v_2)\rangle$. The elimination equation $\delta_1 = \delta_2$

$$v_6 \times v_7 + v_6 \times v_8 \;=\; v_6 \times (v_7 + v_8)$$

is adequate for this pair, since

(a) *δ_2 is partially applicable to $\widehat{\delta'}$ w.r.t. $\times$, where $\widehat{\delta'}$ is the term $(v_3 \times v_4) \times (v_1 + v_2)$, as $\times$ dominates both δ_2 and $\widehat{\delta'}$, and*

(b) *δ_1 and δ_2 are respectively dominated by $+$ and $\times$.*

A new substitution, $\sigma_2 = \{^{v_3 \times v_4}/_{v_6}, ^{v_1}/_{v_7}, ^{v_2}/_{v_8}\}$, unifies δ_2 and $\widehat{\delta'}$, hence δ'', defined as $\sigma_2\delta'[\![^{\delta_1}/_{\widehat{\delta'}}]\!]$, integrates a new rule,

$$R''.\quad v_3 \times v_4 \neq 0 \;\rightarrow\; (v_3 \times v_4) \times v_1 + (v_3 \times v_4) \times v_2 = v_3 \times v_4 \;\Rightarrow$$
$$\Rightarrow\; (v_1 = 1 \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = 1)$$

A final fatal disagreement can be detected between δ'' and ψ,

$\delta''.$	$(v_3 \times v_4)$	$\times$	v_1	$+$	$(v_3 \times v_4)$	$\boxed{\times}$	v_2	$=$	$v_3 \times v_4$
$\psi.$	x^2	$\times$	x^2	$+$	$x^2 \times y$	$\boxed{+}$	$x^2 \times y$	$=$	$x \times x$

The innermost disagreement set,

$$\mathcal{D} \;=\; \{\langle x, v_3\rangle,\ \langle x, v_4\rangle,\ \langle x^2, v_1\rangle,\ \langle x^2 \times y + x^2 \times y, (v_3 \times v_4) \times v_2\rangle\}$$

admits $\langle x^2 \times y + x^2 \times y, (v_3 \times v_4) \times v_2\rangle$ as fatal disagreeing pair. This disagreement can be in principle eliminated by a previously employed equation, with renamed variables,

$$v_9 \times (v_{10} + v_{11}) \;=\; v_9 \times v_{10} + v_{11} \times v_{12},$$

Once established that $\sigma_3 = \{^{v_{10}+v_{11}}/_{v_2}, {}^{v_3 \times v_4}/_{v_9}\}$ is the mgu for the disagreeing subexpression of δ'' and the left-hand side of the elimination equation, the third new rule,

$$v_3 \times v_4 \neq 0$$
$$\rightarrow$$
$$R'''.\quad (v_3 \times v_4) \times v_1 + ((v_3 \times v_4) \times v_{10} + (v_3 \times v_4) \times v_{11}) = v_3 \times v_4$$
$$\Rightarrow$$
$$(v_1 = 1 \wedge (v_{10} + v_{11}) = 0) \vee (v_1 = 0 \wedge (v_{10} + v_{11}) = 1)$$

is derived. For the subproblem $\langle \delta''', \psi\rangle$,

$\delta'''.$	$(v_3 \times v_4)$	$\times$	v_1	$+$	$((v_3 \times v_4) \times v_{10}$	$+$	$(v_3 \times v_4) \times v_{11})$	$=$	$v_3 \times v_4$
$\psi.$	x^2	$\times$	x^2	$+$	$x^2 \times y$	$+$	$x^2 \times y$	$=$	$x \times x$

the innermost disagreement set

$$\mathcal{D} = \{\langle x, v_3\rangle, \langle x, v_4\rangle, \langle x^2, v_7\rangle, \langle y, v_{10}\rangle, \langle y, v_{11}\rangle\}$$

has no fatal disagreements. $\mathcal{D}$ itself represents a substitution that matches δ''' against ψ, and R''' is the desired new rewrite rule. The full generation process can be represented

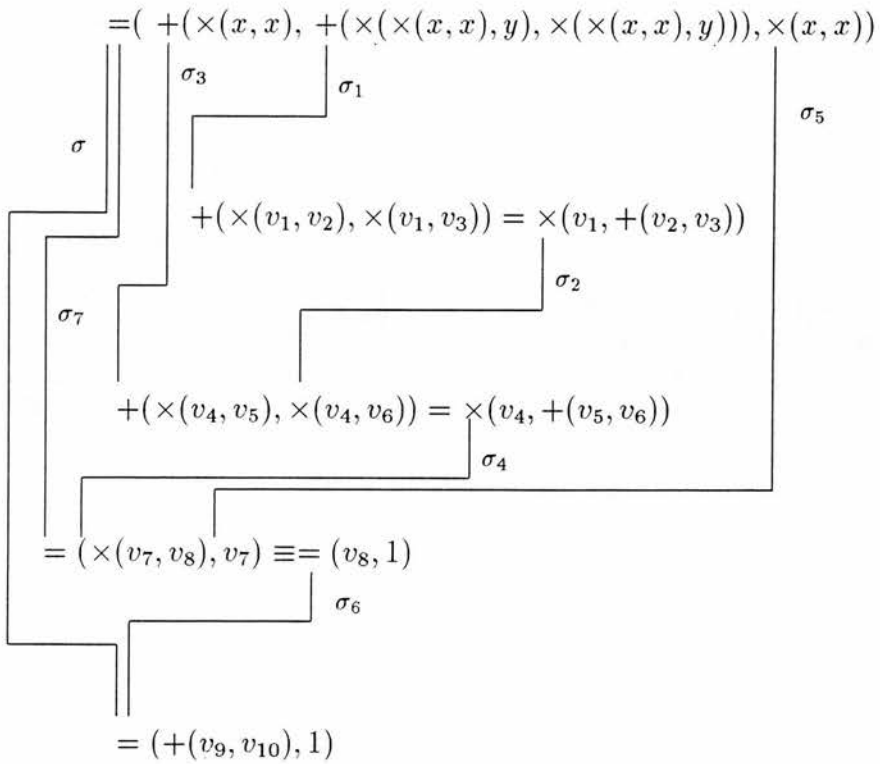


Figure 8.2: An Equality Graph for RGM

by the ordered list of intermediate expressions, the corresponding elimination equations and the unifiers, as indicated on page 221. Figure 8.2 presents the equality graph that describes the generation of δ''. □

Termination is guaranteed for two reasons,

(i) the number of stages is finite, since ψ is a finite expression, and the procedure either eliminates a fatal disagreement or fails: the maximum height of the disagreement between δ and ψ therefore decreases, and

(ii) the set of elimination equations selected at each stage of the process is finite.

Due to the requirement that the dominant symbols of a fatal disagreeing pair must respectively occur in the lhs and rhs of the equation, adequate elimination equations may have to be derived from the equation base by instantiation. For instance, in the

	Rule	Elimination Equation	Substitution
δ.	$\underline{\nu_1 \quad + \quad \nu_2} \quad \hat{=} \quad \boxed{1}$	$\nu_5 = \boxed{1} \equiv (\nu_3 \times \nu_4) \times \nu_5 = \nu_3 \hat{\times} \nu_4$	$\sigma_1 = \{^{\nu_1 + \nu_2}/_{\nu_5}\}$
δ'.	$\nu_3 \times \nu_4 \boxed{\times} \underline{(\nu_1 + \nu_2)} \hat{=} \nu_3 \hat{\times} \nu_4$	$\nu_6 \boxed{\times} (\nu_7 + \nu_8) = (\nu_6 \times \nu_7) \hat{+} (\nu_6 \times \nu_8)$	$\sigma_2 = \{^{\nu_3 \times \nu_4}/_{\nu_6}, ^{\nu_1}/_{\nu_7}, ^{\nu_2}/_{\nu_8}\}$
δ''.	$(\nu_3 \times \nu_4) \times \nu_1 \hat{+} \underline{(\nu_3 \times \nu_4) \boxed{\times} \nu_2} \hat{=} \nu_3 \hat{\times} \nu_4$	$\nu_9 \boxed{\times} (\nu_{10} + \nu_{11}) = (\nu_9 \times \nu_{10}) \hat{+} (\nu_9 \times \nu_{11})$	$\sigma_3 = \{^{\nu_3 \times \nu_4}/_{\nu_9}, ^{\nu_{10} + \nu_{11}}/_{\nu_2}\}$
δ'''.	$(\nu_3 \times \nu_4) \times \nu_1 \hat{+} ((\nu_3 \times \nu_4) \times \nu_{10} \hat{+} (\nu_3 \times \nu_4) \times \nu_{11}) \hat{=} \nu_3 \hat{\times} \nu_4$		

ψ.	$x^2 \times x^2 + x^2 \times y + x^2 \times y = x \times x$

$\boxed{S}$ Disagreeing symbol $\hat{S}$ Expected symbol $\underline{\epsilon}$ Selected subexpression

---

Successful generation of a new rule

problem examined above, the adequate elimination equation displayed in figure 8.1 is actually an instance of

$$v_1 \times v_2 = v_1 \quad \equiv \quad v_2 = 1 \quad (v_1 \neq 0)$$

which is not adequate for the disagreement pair in question. A special case of this principle involves fatal disagreeing pairs which include variables of a conjecture ϕ. According to the initial assumption in the construction of RGM, variables in ϕ have to be dealt with as new individual constants; as a result, any disagreeing pair of the form $\langle t, v \rangle$, where t is a composite term or individual constant, and v is a variable that occurs in ϕ, is fatal. Their removal also has to observe the guidelines for the selection of adequate elimination equations, and therefore v has to occur explicitly in the rhs expression of any adequate elimination equation for $\langle t, v \rangle$. Given the pair

$\delta.$	$(v \boxed{+} u)$	$+$	v
$t.$	$\boxed{x}$	$+$	$(y + x)$

since the disagreeing variable x must be dealt with as a new individual constant, $\mathbf{x}$, the equation

$$v_1 + (v_2 + v_3) \quad = \quad (v_1 + v_2) + v_3$$

has to be instantiated to

$$\mathbf{x} + (v_2 + v_3) \quad = \quad (\mathbf{x} + v_2) + v_3$$

to become adequate.

8.1.2 Variable Instantiation

When variables do not have multiple occurrences in an expression δ that has to be matched against ϵ, RGM deals with each fatal disagreeing pair of a disagreement set on an individual basis. Multiply occurring variables, however, may prevent the derivation of a global substitution, even when individual disagreements are eliminable. In such

cases, all the distinct expressions $\{\epsilon_1, \ldots, \epsilon_n\}$ that should replace different occurrences of a variable u have to be taken into account, and new subproblems are generated by the substitution of each ϵ_i for u in δ. $\langle \delta, \epsilon \rangle$ is then replaced with a series of new problems, $\langle \delta[^{\epsilon_1}/u], \epsilon \rangle, \ldots, \langle \delta[^{\epsilon_n}/u], \epsilon \rangle$, such that a positive solution to any of them amounts to a solution for the original problem as well.

Example 8.1.3 *Let Δ be the set of equations described in example 8.1.2. Given the expressions*

δ.	$\boxed{u}$	$\times$	v	$+$	$\boxed{u}$	$\times$	w
ϵ.	$\boxed{(\boldsymbol{x} \times \boldsymbol{x})}$	$\times$	y	$+$	$\boxed{\boldsymbol{x}}$	$\times$	z

their innermost disagreement set is

$$\mathcal{D} = \{\langle u, x \times x \rangle, \langle v, y \rangle, \langle u, x \rangle, \langle w, z \rangle\}$$

Each element of $\mathcal{D}$ is solvable, but it is not possible to derive from it a substitution that matches δ against ϵ, since there are two terms, $x \times x$ and x, that have to substitute for u. Two new problems have to be examined, involving ϵ and

δ'.	$(\boldsymbol{x} \times \boldsymbol{x})$	$\times$	v	$+$	$(\boldsymbol{x} \times \boldsymbol{x})$	$\times$	w
δ''.	$\boldsymbol{x}$	$\times$	v	$+$	$\boldsymbol{x}$	$\times$	w

The innermost disagreement set for $\langle delta', \epsilon \rangle$ is

$$\mathcal{D}' = \{\langle v, y \rangle, \langle x \times x, x \rangle, \langle w, z \rangle\}$$

for which no solution can be obtained from Δ; neither can it be found for any higher disagreement set, such as $\{\langle (x \times x) \times v, (x \times x) \times y \rangle, \langle (x \times x) \times w, x \times z \rangle\}$. The second subproblem, $\langle \delta'', \epsilon \rangle$, involves the innermost disagreement set

$$\mathcal{D}'' = \{\langle x, x \times x \rangle, \langle v, y \rangle, \langle w, z \rangle\}$$

No solution can be provided here either. However, moving to a higher disagreement set,

$$\mathcal{D}''' = \{\langle x \times v, (x \times x) \times y\rangle, \langle x \times w, x \times z\rangle\}$$

the subproblem

$\delta''.$	$\boxed{x}$	$\times$	v	$+$	$\boxed{x}$	$\times$	w
$\epsilon.$	$\boxed{(\boldsymbol{x} \times \boldsymbol{x})}$	$\times$	y	$+$	$\boxed{\boldsymbol{x}}$	$\times$	z

can be solved by the application of an equation of Δ, $(v_1 \times v_2) \times v_3 = v_1 \times (v_2 \times v_3)$, *or rather its instance,*

$$(\boldsymbol{x} \times v_2) \times v_3 = \boldsymbol{x} \times (v_2 \times v_3)$$

which successfully eliminates the disagreement between δ''' *and* ϵ. □

When the list of conflicting terms consists of variables $u_1, \ldots, u_n$ of the conjecture, instances of all the available equations, generated by the replacement of free variables by $u_1, \ldots, u_n$, have to be tested. Adequate instances, as defined in section 8.1.1, are checked against the conjecture, and additional fatal disagreements may have to be lifted to generate applicability. Since the disagreeing pairs have the form $\langle u_i, u_j\rangle$, $i \neq j$, a high number of adequate equations is expected, and may substantially increase the complexity of the elimination task. To control the search, at first equations are selected only when they are applicable to a subexpression that contains the occurrence of u_i to be eliminated, i.e. secondary disagreement elimination is discouraged.

Example 8.1.4 *Given the pair of terms,*

$\delta.$	$\boxed{u}$	$\times$	v	$+$	$\boxed{u}$	$\times$	w
$\epsilon.$	$\boxed{\boldsymbol{x}}$	$\times$	y	$+$	$\boxed{\boldsymbol{z}}$	$\times$	x

two instances of δ *must be examined,*

$$\delta'. \quad \boldsymbol{x} \times v + \boldsymbol{x} \times w$$
$$\delta''. \quad \boldsymbol{z} \times v + \boldsymbol{z} \times w$$

If attention is restricted to δ', elimination equations for the replacement of $\boldsymbol{x}$ with $\boldsymbol{z}$ must be identified. Equation $u \times v = v \times u$, in particular, is applicable to a subexpression of δ',

$$\delta'. \quad \boldsymbol{x} \quad \times \quad v \quad + \quad \boxed{\boldsymbol{x}} \times w$$

hence it can be checked for adequacy. For this example, its instance

$$\boldsymbol{z} \times \boldsymbol{x} \quad = \quad \boldsymbol{x} \times \boldsymbol{z}$$

has all the properties of an adequate elimination equation. □

The variable strategy completes the core of RGM. Table 8.1 has a summary of the algorithm[3].

8.1.3 Subexpression Selection

When a rule R is not applicable to a conjecture, if a disagreement elimination mechanism is available, a subexpression against which R could be semantically matched must be selected in the first place, as already stressed in section 8.1. The basic version of RGM described above, however, does not provide any guideline for ordering candidate expressions. Choices may be determined, for instance, by the *agreement depth* between rule and subexpression.

---

[3] The seventh step of the algorithm requires the notion of *RGM-unifiability* of a pair of expressions $\langle \delta, \epsilon \rangle$, which is defined similarly to RGM-matchability, with the proviso that variable instantiation and subexpression replacement may take place in both δ and ϵ. For this reason, termination is not ensured. For instance, given the expressions t and $t \times 1$, where t denotes a generic term of the underlying language, and the elimination equation $E.\ v = (v \times 1) \times 1$, the following infinite RGM-unification sequence,

$$\begin{array}{ccccccc} \boxed{t} & \xrightarrow{E} & (t\boxed{\times}1) \times 1 & & (\boxed{t} \times 1) \times 1 & \ldots \\ t\boxed{\times}1 & & \boxed{t} \times 1 & \xrightarrow{E} & ((t\boxed{\times}1) \times 1) \times 1 & \ldots \end{array}$$

where disagreeing symbols are indicated inside boxes, can be built. Additional restrictions can be nonetheless imposed to guarantee termination. The current implementation of RGM achieves it by operating with standard unification, which may be seen as a special case of RGM-unification in which the set Δ of equations is empty.

Let Δ be a set of (conditional) equations, $\langle \delta, \epsilon \rangle$ be a pair of expressions where ϵ is variable-free, and $\boldsymbol{D}$ be the set of all disagreement sets for $\langle \delta, \epsilon \rangle$.

i. $\langle \Delta, \delta, \epsilon \rangle$ is RGM-matchable iff $\langle \Delta, \delta, \epsilon, \boldsymbol{D} \rangle$ is RGM-matchable.

ii. $\langle \Delta, \delta, \epsilon, \boldsymbol{D} \rangle$ is RGM-matchable iff $\boldsymbol{D}$ has the form $[\mathcal{D}|\boldsymbol{D}']$, and

   (a) $\langle \Delta, \delta, \epsilon, \mathcal{D} \rangle$ is RGM-matchable, or

   (b) $\langle \Delta, \delta, \epsilon, \boldsymbol{D}' \rangle$ is RGM-matchable.

iii. $\langle \Delta, \delta, \epsilon, \mathcal{D} \rangle$ is RGM-matchable iff $\langle \Delta, \delta, \epsilon, D_i \rangle$ is RGM-matchable, for all $i, 1 \leq i \leq n$, where $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a disagreement set for $\langle \delta, \epsilon \rangle$.

iv. $\langle \Delta, \delta, \epsilon, D_i \rangle$ is RGM-matchable iff $\langle \Delta, \delta, \epsilon, D_i, d_{i,j} \rangle$ is RGM-matchable, for each $j, 1 \leq j \leq m_i$, where D_i is a disagreement pair for $\langle \delta, \epsilon \rangle$ and $\{d_{i,1}, \ldots, d_{i,m_i}\}$ is the set of fatal disagreeing pairs of D_i.

v. $\langle \Delta, \delta, \epsilon, D_i, d_{i,j} \rangle$ is RGM-matchable iff $\langle \Delta, \delta, \epsilon, \mathcal{D}_i, d_{i,j}, \mathcal{E}_{i,j} \rangle$ is RGM-matchable, where $\mathcal{E}_{i,j}$ is the set of all adequate elimination equations, obtained from Δ, for a fatal disagreeing pair $d_{i,j}$.

vi. $\langle \Delta, \delta, \epsilon, \mathcal{D}_i, d_{i,j}, \mathcal{E}_{i,j} \rangle$ is RGM-matchable iff $\mathcal{E}_{i,j}$ has the form $[E|\mathcal{E}']$ and

   (a) $\langle \Delta, \delta, \epsilon, \mathcal{D}_i, d_{i,j}, E \rangle$ is RGM-matchable, or

   (b) $\langle \Delta, \delta, \epsilon, \mathcal{D}_i, d_{i,j}, \mathcal{E}' \rangle$ is RGM-matchable.

vii. $\langle \Delta, \delta, \epsilon, \langle \delta', \epsilon' \rangle, d_{i,j}, e_1 = e_2 \rangle$ is RGM-matchable iff

   (a) δ' and e_1 are unifiable, with σ as mgu, and $\langle \Delta, \sigma\delta[\![^{e_2}/_{\delta'}]\!], \epsilon, \langle \sigma e_2, \epsilon' \rangle \rangle$ is RGM-matchable, or

   (b) δ' and e_1 are RGM-unifiable, with σ' as most general RGM-unifier, and $\langle \Delta, \sigma'\delta[\![^{e_2}/_{\delta'}]\!], \epsilon, \langle \sigma' e_2, \epsilon' \rangle \rangle$ is RGM-matchable,

   where $e_1 = e_2$ is an adequate elimination equation for a fatal disagreeing pair $d_{i,j}$ taken from a disagreement pair $\langle \delta', \epsilon' \rangle$ for $\langle \delta, \epsilon \rangle$.

Table 8.1: An algorithm for RGM

Definition 8.1.3 (Agreement & Disagreement Depth)

Let ϕ be a formula in which a symbol S occurs at position p, and let $\epsilon_1, \ldots, \epsilon_n$ be all the subexpressions of ϕ that contain the mentioned occurrence of S. Let $R_1, \ldots, R_m$ be remove *rules for S.*

i. For each syntactically compatible pair $\langle \delta_i, \epsilon_j \rangle$ (i.e. δ_i and ϵ_j have identical syntactic types), where δ_i is the lhs expression of R_i, its agreement depth *is the depth d of the lowest layer of ϵ_j, such that each layer of depth up to d of ϵ_j is syntactically identical to the corresponding layer of $\sigma\delta_i$, for some partial unifier σ.*

ii. The disagreement depth *for $\langle \delta_i, \epsilon_j \rangle$ is the difference between the number of layers of ϵ_j and the agreement depth for $\langle \delta_i, \epsilon_j \rangle$.*

From a heuristic viewpoint, the greater the disagreement depth, the larger the transformation process required to eliminate disagreements. For this reason, candidate disagreeing pairs are ordered according to this parameter. Provided that all possible pairs are checked, their ordering does not pose any additional threat to completeness. There is a similarity between this heuristic ordering and the *degree of unification* in RUE-resolution, defined in section 7.2.3, which, nonetheless, assesses essentially the agreement and ignores the disagreement depth.

Example 8.1.5

i. Let ϕ be the conjecture

$$x = 0 \wedge y \times (x + z) = z^2$$

If $+$ has to be removed from it, the list of candidate subexpressions consists of $x + z$, $y \times (x + z)$, $y \times (x + z) = z^2$, and ϕ itself. Given a set of remove *rules for $+$,*

$$\begin{array}{lrcl} R_1 & 0 + v & \Rightarrow & v \\ R_2 & v_1 + v_2 = v_1 + v_3 & \Rightarrow & v_2 = v_3 \end{array}$$

the list of compatible pairs has four elements,

$$\begin{array}{c} \langle\ 0 + v,\ \ x + z\ \rangle \\ \langle\ 0 + v,\ \ y \times (x + z)\ \rangle \\ \langle\ v_1 + v_2 = v_1 + v_3,\ \ y \times (x + z) = z^2\ \rangle \\ \langle\ v_1 + v_2 = v_1 + v_3,\ \ x = 0 \wedge y \times (x + z) = z^2\ \rangle \end{array}$$

The agreement depths for these pairs are respectively $1, 0, 1$ *and* 0*, whereas the disagreement depths are* $1, 3, 3$ *and* 5*, which then define the order according to which the rule generation mechanism (RGM) will try to semantically match a rule against the conjecture.*

ii. Let ϕ *be the formula*

$$x^2 + (y^2 + z^2) = 1 + 0$$

and let the rule base contain the following remove *rules for* $+$*,*

$$\begin{array}{lrcl} R_1 & v_1 + v_2 = 1 & \Rightarrow & (v_1 = 1 \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = 1) \\ R_2 & v_1 + v_2 = 0 & \Rightarrow & v_1 = 0 \wedge v_2 = 0 \\ R_3 & v_1 + v_2 = v_1 + v_3 & \Rightarrow & v_2 = v_3 \end{array}$$

Only one compatible pair can be identified for each rule.

$$\begin{array}{c} \langle\ v_1 + v_2 = 1,\ \ x^2 + (y^2 + z^2) = 1 + 0\ \rangle \\ \langle\ v_1 + v_2 = 0,\ \ x^2 + (y^2 + z^2) = 1 + 0\ \rangle \\ \langle\ v_1 + v_2 = v_1 + v_3,\ \ x^2 + (y^2 + z^2) = 1 + 0\ \rangle \end{array}$$

The agreement depths are respectively 1, 1 and 2, whereas the disagreement depths are 4, 4 and 3. The last pair therefore is the first chosen, even though the first is the only one that contains a raisable disagreement pair, $\langle 1, 1 + 0 \rangle$*.* □

When a proof plan for normalisation is interfaced to RGM, a new heuristic measure for *remove* rules, $m_{r\bullet}$, takes into account both the complexity of disagreement elimination and the introduction of new occurrences of deviant symbols. It is defined as

$$m_{r\bullet}(R, \epsilon) \quad = \quad m_r(R) + \boldsymbol{d}(\delta, \epsilon)$$

where ϵ is a subexpression of the conjecture, $R.\ \delta \Rightarrow \delta'$ is a *remove* rule, $\boldsymbol{d}(\delta, \epsilon)$ is the disagreement depth for the pair $\langle \delta, \epsilon \rangle$, and m_r is the measure function for *remove* rules, defined in section 6.3.2. There is an underlying assumption that each disagreeing layer requires one additional transformation step.

8.1.4 Most General Derived Rule

As already illustrated in a series of examples, RGM deals with rewriting problems where a quantifier-free rule $R.\ \delta_1 \Rightarrow \delta_2$ is not applicable to a formula ϕ, in which case there must be a disagreement pair $\langle \epsilon_1, \epsilon_2 \rangle$ between δ_1 and ϕ. Given an adequate elimination equation $E_i.\ \gamma_1 \rightleftharpoons \gamma_2$ for $\langle \epsilon_1, \epsilon_2 \rangle$ w.r.t. a disagreeing pair of symbols, $\langle S_1, S_2 \rangle$, in the event γ_1 and ϵ_1 admit a mgu σ, a new valid rule, $R'.\ \sigma\delta_1[\![^{\gamma_2}/_{\epsilon_1}]\!] \Rightarrow \sigma\delta_2$, which does not contain the disagreeing pair $\langle S_1, S_2 \rangle$, is eventually obtained.

Lemma 8.1.2 *Let $E_1.\ \gamma_1 \rightleftharpoons \gamma_2$ and $E_2.\ \delta_1 \rightleftharpoons \delta_2$ be quantifier-free valid equations of the same syntactic type (i.e. they are both derived from equations or equivalences) that do not share variables. If σ is a unifier for γ_1 and δ_1, then*

$$E.\ \sigma\gamma_2 \rightleftharpoons \sigma\delta_2$$

is also a quantifier-free valid equation.

PROOF. Since $\gamma_1 \rightleftharpoons \gamma_2$ and $\delta_1 \rightleftharpoons \delta_2$ are valid, any of their instances, e.g. $\sigma\gamma_1 \rightleftharpoons \sigma\gamma_2$ and $\sigma\delta_1 \rightleftharpoons \sigma\delta_2$ are valid as well. From the symmetry and transitivity of $\rightleftharpoons$, and since E_1 and E_2 have the same syntactic type, it follows that $\sigma\gamma_2 \rightleftharpoons \sigma\delta_2$ is valid. ∎

If γ_1 and ϵ_1 are not unifiable, disagreeing pairs may still be eliminable from $\langle \gamma_1, \epsilon_1 \rangle$. Their successful elimination, whenever possible, allows the construction of the desired new rule. In the case only most general unifiers are used in the process, the *most general composite rule* is derived.

Definition 8.1.4 (Composite Rules)

Let $R.\ \delta_1 \Rightarrow \delta_2$ be a rewrite rule, $\mathcal{E} = \langle E_1, \ldots, E_n \rangle$ be an ordered set of equations, such that E_i has the form $\delta_{i,1} \rightleftharpoons \delta_{i,2}$, and $\mathcal{P} = \{p_1, \ldots, p_n\}$ be a set of finite lists of natural numbers.

i. $R'.\ \delta'_1 \Rightarrow \delta'_2$ is a composite rule derived from *R, $\mathcal{E}$ and $\mathcal{P}$ iff there exists a unification chain*

$$\gamma_1, \ldots, \gamma_{n+1}$$

such that

(a) $\gamma_1 \stackrel{s}{=} \delta_1$ *and* $\gamma_{n+1} \stackrel{s}{=} \delta_1'$,

(b) $\gamma_{i+1} \stackrel{s}{=} \gamma_i[\![\langle \sigma_i \delta_{i,2}, \widehat{\gamma_i}, p_i \rangle]\!]$, *where* σ_i *is a unifier for* $\delta_{i,1}$ *and* $\widehat{\gamma_i}$, *the subexpression of* γ_i *at position* p_i *(i.e.* γ_{i+1} *is generated from* γ_i *when its subexpression at position* p_i *is replaced with* $\sigma_i \delta_{i,2}$*), and*

(c) $\delta_2' \stackrel{s}{=} \sigma \delta_2$, *where* $\sigma = \sigma_n \circ \cdots \circ \sigma_1$.

ii. R' *is the* most general composite rule *for* R, $\mathcal{E}$ *and* $\mathcal{P}$ *iff every composite rule* R'' *derived from* R, $\mathcal{E}$ *and* $\mathcal{P}$ *is an instance of* R'.

Equations, therefore, are not used as rewrite rules in the course of rule composition, since, rather than being matched against a selected rule R, they are unified with subexpressions of R, as illustrated below.

Example 8.1.6 *Let* R. $\delta_1 \Rightarrow \delta_2$ *be the arithmetical* remove *rule,*

$$v_1 \times v_2 = v_1 \times v_3 \quad \Rightarrow \quad v_1 = 0 \vee v_2 = v_3$$

and let

$$\mathcal{E} \quad = \quad \langle v_4 \times v_5 = v_5 \times v_4, \;\; v_6 \times (v_7 + v_8) = v_6 \times v_7 + v_6 \times v_8 \rangle$$

be an ordered set of equations. Given the unification chain,

$$\overbrace{\underline{v_1 \times v_2} = v_1 \times v_3}^{\delta_1}$$

$$v_4 \times v_5 \rightleftharpoons v_5 \times v_4 \qquad \sigma_1 = \left\{ {}^{v_1}/_{v_4}, {}^{v_2}/_{v_5} \right\}$$

$$v_2 \times v_1 = \underline{v_1 \times v_3}$$

$$v_6 \times (v_7 + v_8) \rightleftharpoons v_6 \times v_7 + v_6 \times v_8 \qquad \sigma_2 = \left\{ {}^{v_1}/_{v_6}, {}^{v_7 + v_8}/_{v_3} \right\}$$

$$\underbrace{v_2 \times v_1 = v_1 \times v_7 + v_1 \times v_8}_{\delta_1'}$$

where the underlined subexpressions are unifiable with the rhs expression of an equation of $\mathcal{E}$, the global substitution, $\sigma = \sigma_2 \circ \sigma_1$, is

$$\{v_1/v_6, v_7 + v_8/v_3, v_1/v_4, v_2/v_5\}$$

Hence, the rule $\delta_1' \Rightarrow \sigma\delta_2$,

$$v_2 \times v_1 = v_1 \times v_7 + v_1 \times v_8 \quad \Rightarrow \quad v_1 = 0 \vee v_2 = v_7 + v_8$$

is a composite rule derived from R, $\mathcal{E}$ and $\mathcal{P} = \{[1], [2]\}$. □

Theorem 8.1.1 *Given a rule R. $\delta_1 \Rightarrow \delta_2$ and an ordered set of equations $\mathcal{E} = \langle E_1, \ldots, E_n \rangle$, where each E_i has the form $\delta_{i,1} \rightleftharpoons \delta_{i,2}$, $1 \leq i \leq n$, and does not share variables with any other equation of $\mathcal{E}$, a rule R'. $\delta_1' \Rightarrow \delta_2'$ is the most general composite rule derived from R and $\mathcal{E}$ iff $\delta_1' \stackrel{s}{=} \sigma\delta_{n,2}$ and $\delta_2' \stackrel{s}{=} \sigma\delta_2$, where σ is the most general unifier for the pairs*

$$\langle \delta_1, \delta_{1,1} \rangle, \langle \delta_{1,2}, \delta_{2,1} \rangle, \langle \delta_{2,2}, \delta_{3,1} \rangle, \ldots, \langle \delta_{i,2}, \delta_{i+1,1} \rangle, \ldots, \langle \delta_{n-1,2}, \delta_{n,1} \rangle$$

PROOF.

$(\rightarrow)$

As R'. $\delta_1' \Rightarrow \delta_2'$ is a composite rule derived from R and $\mathcal{E}$, there must be formulae $\gamma_1, \ldots, \gamma_{n+1}$ and substitutions $\sigma_1, \ldots, \sigma_n$ such that $\gamma_1 \stackrel{s}{=} \delta_1$ and

$$\begin{array}{rcccl}
\sigma_1\delta_1 & \stackrel{s}{=} & \sigma_1\delta_{1,1} & \stackrel{s}{=} & \gamma_2 \\
\sigma_2(\sigma_1\delta_{1,2}) & \stackrel{s}{=} & \sigma_2\delta_{2,1} & \stackrel{s}{=} & \gamma_3 \\
\sigma_3(\sigma_2\delta_{2,2}) & \stackrel{s}{=} & \sigma_3\delta_{3,1} & \stackrel{s}{=} & \gamma_4 \\
 & \vdots & & \vdots & \\
\sigma_n(\sigma_{n-1}\delta_{n-1,2}) & \stackrel{s}{=} & \sigma_n\delta_{n,1} & \stackrel{s}{=} & \gamma_n \\
\sigma_n\delta_{n,2} & \stackrel{s}{=} & \delta_1' & \stackrel{s}{=} & \gamma_{n+1}
\end{array}$$

Since the equations of $\mathcal{E}$ do not share variables, the composite substitution $\sigma = \sigma_n \circ \cdots \circ \sigma_1$, is a common unifier for each pair $\langle \delta_{i,2}, \delta_{i+1,1} \rangle$, $1 \leq i \leq n$. R', therefore, has the form $\sigma\delta_{n,2} \Rightarrow \sigma\delta_2$. If σ were not the most general unifier for these pairs, there would be substitutions σ' and τ such that

$$\sigma = \tau \circ \sigma' \quad (\dagger)$$

where τ is a non-trivial substitution and σ' is also a unifier for $\langle \delta_{i,2}, \delta_{i+1,1} \rangle$. Therefore R'', defined as

$$\sigma' \delta_{1n,2} \Rightarrow \sigma' \delta_2, \quad (\ddagger)$$

would be also a composite rule for R and $\mathcal{E}$. From $(\dagger)$, $(\ddagger)$ and the definition of R', it follows that $R' \stackrel{s}{=} \tau R''$, in conflict with the assumption that R' is the most general composite rule derived from R and $\mathcal{E}$. Hence σ is the mgu for the above pairs of expressions.

$(\leftarrow)$

If σ is the mgu for $\langle \delta_{i,2}, \delta_{i+1,1} \rangle, 1 \leq i \leq n$, there is an associated composite rule, R'. $\sigma \delta_{n,2} \Rightarrow \sigma \delta_2$. If R' were not the most general composite rule for R and $\mathcal{E}$, there would be substitutions σ' and τ and another rule, R'', defined as

$$\sigma' \delta_{1,1} \Rightarrow \sigma' \delta_{n,2} \quad (\dagger)$$

such that

$$R' \stackrel{s}{=} \tau R'' \quad (\ddagger)$$

From $(\dagger)$, $(\ddagger)$ and the definition of R', it follows that $\sigma = \tau\sigma'$. As σ' is also a unifier for $\langle \delta_{i,2}, \delta_{i+1,1} \rangle$, it is not the mgu for them, in contradiction with the hypothesis. ∎

This result does not guarantee that a rule derived from R and $\langle E_1, \ldots, E_n \rangle$ is the most general rule for the full equation base from which each E_i has been taken. An additional condition for the rule base as a whole is the selection of adequate elimination equations which subsume other equally adequate options.

Example 8.1.7 *Let δ and ϵ,*

$\delta.$	v_1	$\boxed{+}$	v_2	$=$	v_1	$+$	v_3
$\epsilon.$	x^2	$\boxed{\times}$	$(y + zw)$	$=$	x^2y	$+$	$(x + z)$

be a pair of expressions such that δ has to be Δ-matched against ϵ, where Δ is the equation base. Assuming that Δ contains E_1 and E_2,

$$\begin{array}{lrcl} E_1 & v_4 \times v_5 + v_4 \times v_6 & = & v_4 \times (v_5 + v_6) \\ E_2 & (v_4 \times v_5) \times v_6 + (v_4 \times v_5) \times v_7 & = & (v_4 \times v_5) \times (v_6 + v_7) \end{array}$$

two transformed expressions, δ' and δ'',

$$\begin{array}{lrcl} \delta'. & v_4 \times (v_5 + v_6) & = & v_4 \times v_5 + v_3 \\ \delta''. & (v_4 \times v_5) \times (v_6 + v_7) & = & (v_4 \times v_5) \times v_6 + v_3 \end{array}$$

can be generated. Since E_2 is an instance of E_1, δ'' is an instance of δ' as well, in which case the most general rule for Δ requires the selection of the most general of the available adequate elimination equations, E_1. □

The selection of a most general adequate equation is relevant only in the context of rule generation and Δ-matching. For Δ-unification, with which RUE-resolution and ECOP are concerned, both solutions above are equally acceptable, since the lhs expression of both E_1 and E_2 are unifiable with a subexpression of δ, and their rhs expressions are unifiable with a subexpression of ϵ.

8.2 Extensions of RGM

At least two aspects of RGM are responsible for the incompleteness of the mechanism with respect to the exhaustive scrutiny of the full equational tree. First, since elimination rules have to contain the disagreeing pair split between the left and right-hand side expressions, in the event the equation is or becomes applicable to the conjecture, the disagreeing symbol is removed in a single rewriting step. When more than one equation would be necessary to achieve this effect, the mechanism fails. Also, due to the restriction concerning the reintroduction of disagreements, once a fatal disagreeing pair between a *remove* rule and a conjecture is identified, the current strategy does not allow either the introduction of any new higher disagreement, or the reintroduction of a disagreement at a position from which a disagreement has been previously removed. This requirement also affects the completeness of this mechanism w.r.t. the full rewriting tree.

Completeness is relevant in this context only if achieved under well-defined control guidelines, and it is usually necessary to find a compromise between completeness and efficiency. Nonetheless, it is also important to examine extensions of the mechanism which could reduce its incompleteness through the enlargement of the set of solvable Δ-matching problems $\langle \delta, \psi \rangle$ under Δ. Possible extensions include the elimination of disagreements along several steps, the introduction of additional subproblems and the enlargement of context for disagreement elimination.

8.2.1 Multiple-Step Rewriting

Fatal disagreeing pairs which cannot be eliminated by the application of a single equation may however be efficiently raised through planning strategies such as that available in ECOP, where an equality connection graph is built to link disagreeing subexpressions. For instance, if the disagreeing symbol + in the conjecture

$$x^2 \times x^2 + x^2 \times y + x^2 \times y = x \times 0 \boxed{+} y \times 0$$

has to be replaced by 0, and the set of equations is limited to

$$\begin{array}{lrcl} E_1. & v_1 \times v_2 + v_3 \times v_2 & = & (v_1 + v_3) \times v_2 \\ E_2. & v \times 0 & = & 0 \end{array}$$

the desired replacement cannot be completed, due to the absence of adequate elimination equations: no equation (or instance of equation) exhibits the dominant symbols of the disagreeing pair split between the left and right-hand side expressions, at the same positions. The connection graph in figure 8.3 shows however that the introduction of a single additional step in the process allows the elimination of the disagreeing pair.

The inclusion of this additional feature in RGM could be done in two alternative ways. The definition of adequate elimination equation could be replaced by the notion of *adequate elimination graph*, made up of equations taken from the equality base. Alternatively, the current definition of adequate elimination equation could be preserved, with the introduction of a parallel mechanism for the generation of new equations and their aggregation to the equation base.

In the second case, some of the new rules would be generated through the *transitive*

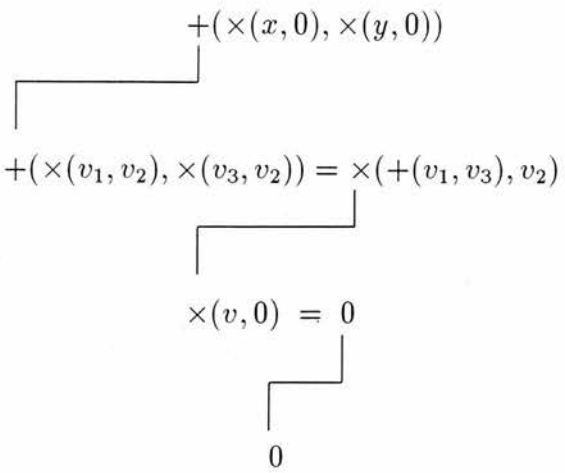


Figure 8.3: Multiple-Step Disagreement Elimination

closure of an equation base $\mathcal{E}$, i.e. a new set $\mathcal{E}', \mathcal{E} \subseteq \mathcal{E}'$ such that, whenever $E_1, E_2 \in \mathcal{E}$, $\phi \overset{E_1}{\Rightarrow} \phi'$ and $\phi' \overset{E_2}{\Rightarrow} \phi''$, there exists a rule $E_3 \in \mathcal{E}'$ such that

$$\phi \overset{E_3}{\Rightarrow} \phi''$$

Another solution would be the implementation of an additional mechanism for the generation of composite elimination equations, whenever a fatal disagreeing pair cannot be dealt with by the equational base. The first proposal generates an unmanageable amount of explicit information, which can make the system inefficient. The second solution keeps the necessary information implicit in the rule base and reveals it only when necessary. The effects of the second alternative on the performance of the system, however, are as yet unknown.

8.2.2 Intermediate Subproblems

The introduction of elimination equality graphs, as defined in the previous section, does not eradicate incompleteness from RGM. Even when a disagreeing pair does not admit an adequate graph, a solution may still be obtained after the introduction of intermediate equations and the generation of new subproblems.

Example 8.2.1 *Let Δ be a set made up of the following equations,*

$$\begin{array}{rcll} v_1 \times v_2 = v_1 & \equiv & v_2 = 1 & (v_1 \neq 0) \\ v_1 \times (v_2 + v_3) & = & v_1 \times v_2 + v_1 \times v_3 & \\ v_1 \times (v_2 \times v_3) & = & (v_1 \times v_2) \times v_3 & \\ v_1^{v_2} \times v_1^{v_3} & = & v_1^{v_2+v_3} & \end{array}$$

and let ϕ be the formula

$$x^{y+x} \neq 0 \supset x^{(y+x)+3}zy^2 + x^{y+x}y = x^{y+x}$$

which is equivalent (in PA) to $x^{y+x} \neq 0 \supset x^{y+x}(x^3zy^2 + y) = x^{y+x}$. It can therefore be rewritten to $x^{y+x} \neq 0 \supset x^3zy^2 + y = 1$, from which it is possible to remove the only occurrence of sum by the application of

$$v_1 + v_2 = 1 \quad \Rightarrow \quad (v_1 = 1 \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = 1) \quad (*)$$

To establish whether a rule for the elimination of + from ϕ can be generated by RGM from $(*)$, $\langle \psi, \delta \rangle$, where δ is the lhs expression of $(*)$ and ψ is the consequent of ϕ, is selected to be Δ-matched. Given its innermost disagreement set,

$$\mathcal{D}_1 \quad = \quad \{\langle t_1zy^2 + t_2y = x^{y+x}, v_1 + v_2 = 1 \rangle\}$$

where t_1 and t_2 respectively denote the terms $x^{(y+x)+3}$ and x^{y+x}, and the fatal disagreeing pair, $\langle x^{y+x}, 1 \rangle$,

$\psi.$	$t_1zy^2 + t_2y$	$=$	x	$\boxed{\exp}$	$(y+x)$
$\delta.$	$v_1 + v_2$	$=$		$\boxed{1}$	

there is an adequate elimination equation for it,

$$(v_3 \exp v_4) \times v_5 = v_3 \exp v_4 \quad \equiv \quad v_5 = 1 \quad (v_3 \exp v_4 \neq 0)$$

which is an instance of $v_3 \neq 0 \rightarrow v_3 \times v_4 = v_3 \equiv v_4 = 1$. Its application to δ generates δ',

$$(v_3 \exp v_4) \times (v_1 + v_2) = v_3 \exp v_4$$

For the second subproblem, $\langle \psi, \delta' \rangle$, given the disagreement set

$$\mathcal{D}_2 = \{\langle t_1 zy^2 + t_2 y, (v_3 \exp v_4) \times (v_1 + v_2)\rangle, \langle x, v_3\rangle, \langle y + x, v_4\rangle\}$$

and the fatal disagreeing pair $\langle t_1 zy^2 + t_2 y, (v_3 \exp v_4) \times (v_1 + v_2)\rangle$,

$\psi.$	$t_1 zy^2$	$\boxed{+}$	$t_2 y$	$=$	x	exp	$(y + x)$
$\delta'.$	$(v_3 \exp v_4)$	$\boxed{\times}$	$(v_1 + v_2)$	$=$	v_3	exp	v_4

equation $v_5 \times (v_6 + v_7) = v_5 \times v_6 + v_5 \times v_7$ is adequate for its elimination. The new expression, δ'',

$$((v_3 \exp v_4) \times v_1) + ((v_3 \exp v_4) \times v_2) = v_3 \exp v_4$$

introduces a third subproblem, $\langle \psi, \delta'' \rangle$, whose innermost disagreement set represents an incompatible substitution,

$$\mathcal{D}_3 = \{\langle x, v_3\rangle, \langle (y + x) + 3, v_4\rangle, \langle zy^2, v_1\rangle, \langle x, v_3\rangle, \langle y + x, v_4\rangle, \langle y, v_2\rangle, \langle x, v_3\rangle, \langle y + x, v_4\rangle\}$$

since $y + x$ and $(y + x) + 3$ are both assigned to v_4. Two new subproblems can be derived from this disagreement set, the first of which is $\langle \psi, \{^{y+x}/_{v_4}\}\delta''\rangle$. Some of the corresponding disagreement sets are

$$\begin{aligned}
\mathcal{D}_{4,1} &= \{\langle x, v_3\rangle, \langle y + x, y\rangle, \langle 3, x\rangle, \langle zy^2, v_1\rangle, \langle y, v_2\rangle\} \\
\mathcal{D}_{4,2} &= \{\langle x, v_3\rangle, \langle (y + x) + 3, y + x\rangle, \langle zy^2, v_1\rangle, \langle y, v_2\rangle\} \\
\mathcal{D}_{4,3} &= \{\langle x^{(y+x)+3}, v_3^{y+x}\rangle, \langle zy^2, v_1\rangle, \langle y, v_2\rangle, \langle x, v_3\rangle\} \\
\mathcal{D}_{4,4} &= \{\langle x^{(y+x)+3}(zy^2), v_3^{y+x} v_1\rangle, \langle y, v_2\rangle, \langle x, v_3\rangle\} \\
\mathcal{D}_{4,5} &= \{\langle x^{(y+x)+3}(zy^2) + x^{y+x} y, v_3^{y+x} v_1 + v_3^{y+x} v_2\rangle, \langle x, v_3\rangle\} \\
\mathcal{D}_{4,6} &= \{\langle x^{(y+x)+3}(zy^2) + x^{y+x} y = x^{y+x}, v_3^{y+x} v_1 + v_3^{y+x} v_2 = v_3^{y+x}\rangle\}
\end{aligned}$$

None of them has a solution under the current restrictions imposed on RGM, not even if elimination graphs are allowed. A solution under ECOP, on the other hand, starts examining disagreements from the innermost set, moving up when no compatible graph can be built. As indicated in figure 8.4, the first three sets do not admit solution for the current set of equations, Δ. The fourth disagreement set, however, admits a solution, represented in figure 8.5. This graph suggests an extension of RGM which can address this and similar equality problems. It requires the introduction of an intermediate equation,

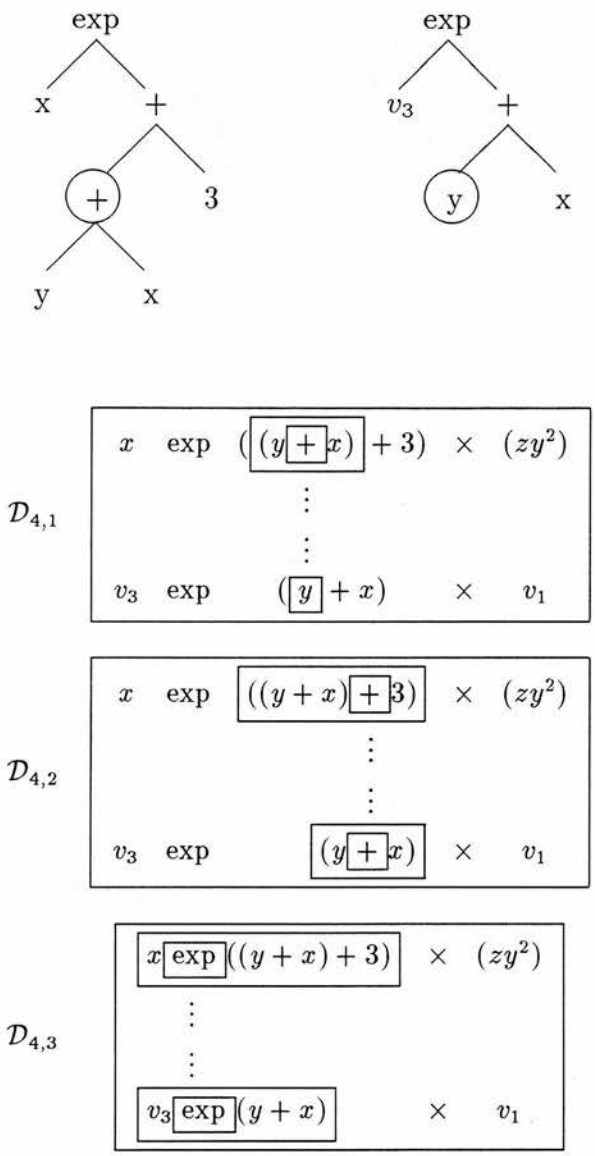


Figure 8.4: Δ-unsolvable links

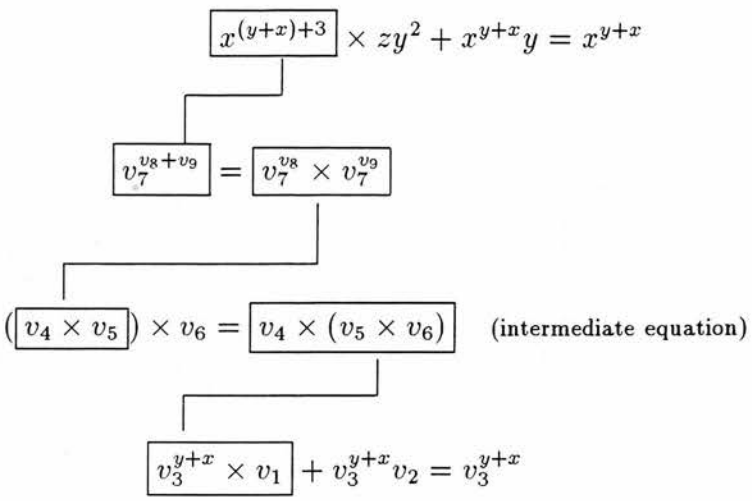


Figure 8.5: The ECOP solution

$$v_4 \times (v_5 \times v_6) = (v_4 \times v_5) \times v_6$$

which creates a new subproblem, $\langle x^{(y+x)+z}zy^2 + t_2y = t_2, (v_3^{y+x}v_5)v_6 + v_3^{y+x}v_2 = v_3^{y+x}\rangle$. *Given the disagreement set*

$$\mathcal{D}_5 \;=\; \{\langle x^{(y+x)+3}, v_3^{y+x} \times v_5\rangle, \langle zy^2, v_6\rangle, \langle y, v_2\rangle, \langle x, v_3\rangle\}$$

and the disagreement pair $\langle x^{(y+x)+3}, v_3^{y+x}v_5\rangle$,

$\psi.$	x	$\boxed{exp}$	$((y+x)+3)$	$\times$	zy^2	$+$	$x^{y+x}y$	$=$ x^{y+x}
$\delta'''.$	v_3^{y+x}	$\boxed{\times}$	v_5	$\times$	v_6	$+$	$v_3^{y+x}v_2$	$=$ v_3^{y+x}

the adequate elimination equation, $v_7^{v_8} \times v_7^{v_9} = v_7^{v_8+v_9}$, *finally allows the generation of* δ'''',

$$v_3^{(y+x)+v_9} \times v_6 + v_3^{y+x} \times v_2 = v_3^{y+x}$$

which can be matched against ψ, *as indicated in figure 8.6.* □

Rule	Elimination Equation	Substitution
$v_3 \quad \exp \quad (y+x) \quad \times \quad v_1$	$\boldsymbol{v_4 \times (v_5 \times v_6) = (v_4 \times v_5) \times v_6}$	σ_1
$(v_3^{y+x} \quad \times \quad v_5) \quad \times \quad v_6$	$(v_7 \exp v_8) \times (v_7 \exp v_9) = v_7 \exp(v_8 + v_9)$	σ_2
$v_3 \quad \exp \quad ((y+x)+v_9) \quad \times \quad v_6$		
$x \quad \exp \quad ((y+x)+3) \quad \times \quad zy^2$		

$\underline{\epsilon}$ selected subexpression

$\boldsymbol{\delta_1 \rightleftharpoons \delta_2}$ intermediate equation

σ_1 $\{v_3^{y+x}/v_4, v_5 \times v_6/v_1\}$

σ_2 $\{v_3/v_7, y+x/v_8, v_7^{v_9}/v_5\}$

Figure 8.6: Intermediate Subproblems in RGM

The elimination of the fatal disagreeing pair $\langle +, y \rangle$ *with the preservation of the upper context*, represented in figure 8.7, has been achieved only after the application of an intermediate equation, which actually changed the context. The introduction of intermediate subproblems does not amount to multiple-step disagreement elimination, described in section 8.2.1, since the latter involves the construction of a chain where the original disagreeing pair of symbols, $\langle S_1, S_2 \rangle$, is linked e.g. by a series of equations of the form

$$S(S_1 \boldsymbol{t_1}) \cdot \cdots \cdot S(S_1 \boldsymbol{u_1}) = S(S_i \boldsymbol{u}_j) \cdot \cdots \cdot S(S_j \boldsymbol{u}_j) = S(S_2 \boldsymbol{u_2}) \cdot \cdots \cdot S(S_2 \boldsymbol{t_2})$$

Intermediate subproblems, on the other hand, may break the sequence by the introduction of new disagreements at a higher level, e.g.

$$S(S_1 \boldsymbol{t_1}) \cdot \cdots \cdot S(S_1 \boldsymbol{u_1}) = S(S_i \boldsymbol{u}_j) \cdot \cdots \cdot \boxed{S}(S_j \boldsymbol{u}_j) = \boxed{S'}(S_k \boldsymbol{u_2}) \cdot \cdots \cdot S(S_2 \boldsymbol{t_2})$$

In the above sequence, the disagreeing pair of symbols of the last equality link, $\langle S', S \rangle$, occurs at a higher layer that the pair initially selected, $\langle S_1, S_2 \rangle$. With respect to this pair, the equation $S(S_j \boldsymbol{u}_j) = S'(S_k \boldsymbol{u_2})$ neither preserves upper symbolic layers nor contains the desired symbol, S_2, in the expected position. Intermediate subproblems require additional control structures, since intermediate equations may not have any relevant syntactic links with other expressions in the rewrite sequence.

8.2.3 Widened Rules

Δ-matching a rule against a subexpression of a formula ϕ is just a special case of the more general problem of matching a rule against subexpressions of formulae that are Δ-equivalent to ϕ.

Definition 8.2.1 (Strong semantic matching)

Given a set of equations Δ, an expression ϵ and a rewrite rule $R.\ \delta_1 \Rightarrow \delta_2$ that does not share variables with ϵ, R is strongly Δ-matchable *against ϵ iff there is a substitution σ and an expression ϵ' such that*

$$\Delta \models \epsilon = \epsilon'$$

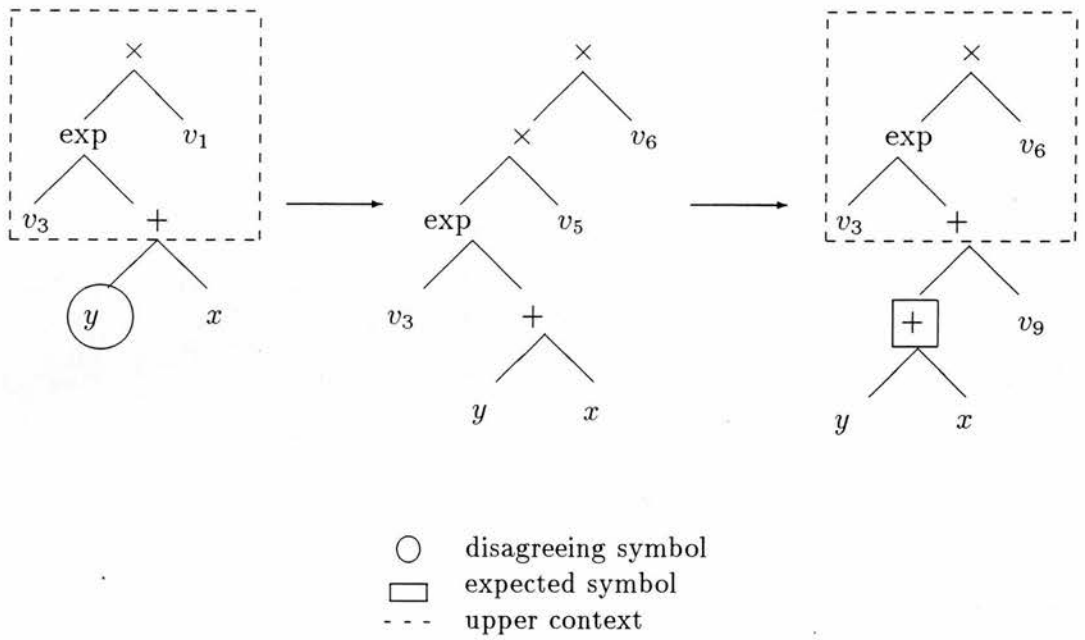


Figure 8.7: Extended Disagreement Elimination

and $\sigma\delta_1 \stackrel{s}{=} \epsilon''$, *where* ϵ'' *is a subexpression of* ϵ'.

Strong Δ-matchability is more general than standard Δ-matchability.

Lemma 8.2.1 *Let* Δ *be a set of (conditional) equations of* $\mathcal{L}$, *and let* ϵ *and* δ *be expressions of* $\mathcal{L}$ *that do not share variables. If* δ *is* Δ*-matchable against* ϵ, *then* δ *is also strongly* Δ*-matchable against* ϵ.

PROOF. Let ϵ' be a subexpression of ϵ such that $\Delta \models (\sigma\delta \rightleftharpoons \epsilon')$, for some substitution σ. Then $\Delta \models (\epsilon \rightleftharpoons \epsilon[\![^{\sigma\delta}/_{\epsilon'}]\!])$ and δ (syntactically) matches $\sigma\delta$. Hence δ strongly Δ-matches ϵ. ∎

Example 8.2.2 *The rule* $R.\ (a = b) \Rightarrow p(c)$, *where* a, b *and* c *are individual constants, cannot be* $\emptyset$*-matched against any of the subexpressions of* ϕ,

$$x = b \wedge a = x$$

given that

$$\begin{aligned} (x = b \wedge a = x) &\equiv (a = b) \\ (x = b) &\equiv (a = b) \\ (a = x) &\equiv (a = b) \end{aligned}$$

are all invalid. However, given an equivalent version of ϕ,

$$x = b \wedge a = b$$

it is possible to match R *against the second conjunct.* □

The strong version of Δ-matching can be partially incorporated to RGM through an extension of the definition of adequate elimination equation. Given the pair $\langle \delta, \epsilon \rangle$, where δ has to be Δ-matched against the subexpression ϵ of a conjecture, the selection of elimination equations does not have to be restricted to those applicable to ϵ. It would be also possible to consider equations which match higher layers of the formula from which ϵ has been taken. For this to take place, it becomes necessary to widen δ accordingly.

Definition 8.2.2 (Widened rewrite rule)

Let $R.\ \delta_1 \Rightarrow \delta_2$ *be a rule and* ϵ *be an expression where the subexpression* ϵ' *has a single occurrence.*

i. A generalised version *for* ϵ *w.r.t.* ϵ' *is an expression* ϵ^g *defined as follows.*

(a) If each term that occurs in ϵ *but outside* ϵ' *is either an individual variable or a composite term that contains* ϵ' *(in the event* ϵ' *is a term), then* $\epsilon^g \stackrel{s}{=} \epsilon$.

(b) If $p(t_1, \ldots, t_n)$ *occurs in* ϵ *and lies outside* ϵ' *(in the event* ϵ' *is a formula), and* $t_{i_1}, \ldots, t_{i_m}, i_j \in \{1, \ldots, n\}$, *are composite terms that do not contain* ϵ' *(if* ϵ' *is a term), then* $\epsilon^g \stackrel{s}{=} (\epsilon[\![^{v_1}/_{t_{i_1}}, \ldots, ^{v_m}/_{t_{i_m}}]\!])^g$, *where* $v_1, \ldots, v_m$ *are variables that do not occur in* ϵ.

ii. $R'.\ \epsilon[\![^{\delta_1}/_{\epsilon'}]\!] \Rightarrow \epsilon[\![^{\delta_2}/_{\epsilon'}]\!]$ *is the* primitive widened rule *for* R *w.r.t.* ϵ *and* ϵ'.

iii. $R''.\ \epsilon^g[\![^{\delta_1}/_{\epsilon'}]\!] \Rightarrow \epsilon^g[\![^{\delta_2}/_{\epsilon'}]\!]$ *is a* most general widened rule *for* R *w.r.t.* $\epsilon[\![\epsilon']\!]$ *iff* ϵ^g *is a* generalised version *for* ϵ *w.r.t.* ϵ'.

Lemma 8.2.2 *Let* $R.\ \delta_1 \Rightarrow \delta_2$ *be a valid rule and* Δ *be a set of equations. Let* ϕ *be a formula in which the expression* ϵ *has a single occurrence.*

i. Both primitive and most general widened rules for R *w.r.t.* ϕ *and* ϵ *are valid, provided that* ϵ *and* δ_1 *have the same syntactic type.*

ii. If a most general widened rule for R *w.r.t.* $\phi[\![\epsilon]\!]$ *is* Δ*-matchable against* ϕ*, then* R *is strongly* Δ*-matchable against* ϕ.

PROOF.

i. If $\delta_1 \Rightarrow \delta_2$ is valid, then $\phi[\![^{\delta_1}/_{\epsilon}]\!] \rightleftharpoons \phi[\![^{\delta_2}/_{\epsilon}]\!]$ is valid as well, for any formula $\phi[\![\epsilon]\!]$.

ii. If $\phi^g[\![^{\delta_1}/_{\epsilon}]\!]$ can be Δ-matched to ϕ, there is a substitution σ such that

$$\Delta \ \models\ \sigma\phi^g[\![^{\delta_1}/_{\epsilon}]\!] \equiv \phi$$

Since δ_1 can be matched against a subexpression of $\sigma\phi^g[\![^{\delta_1}/_{\epsilon}]\!]$ (namely, $\sigma\delta_1$), R is Δ-matchable to ϕ. ∎

The next example illustrates how extended elimination equations cover some of the cases where an expression can be Δ-matched against a proper subexpression of a transformed (Δ-equivalent) version of the conjecture, but not to any relevant subexpression of the conjecture itself.

Example 8.2.3

i. Given the arithmetical conjecture ϕ,

$$0 = x \ \wedge\ y^2 + z \times w^2 = x,$$

and a set of equations, Δ,

$$\begin{array}{rcl} v_1 + v_2 & = & v_2 + v_1 \\ v_1 \times v_2 & = & v_2 \times v_1 \\ v_1 = v_2 \ \wedge \ v_3 = v_1 & \equiv & v_1 = v_2 \ \wedge \ v_3 = v_2 \end{array}$$

the rule R. $v_1 + v_2 = 0 \Rightarrow (v_1 = 0 \wedge v_2 = 0)$ cannot be Δ-matched against either ϕ or its subformula $y^2 + z \times w^2 = x$: for the last case, given the disagreement set $\mathcal{D} = \{\langle y^2 + z \times w^2 = x, v_1 + v_2 = 0\rangle\}$, the disagreeing pair $\langle 0, x\rangle$,

y^2	$+$	$z \times w^2$	$=$	$\boxed{x}$
v_1	$+$	v_2	$=$	$\boxed{0}$

cannot be raised due to the nonexistence of adequate elimination equations. However, if a wider context, e.g. the full conjecture, is taken into account, the equivalence $(v_3 = v_4 \wedge v_5 = v_3) \equiv (v_3 = v_4 \wedge v_5 = v_4)$, or rather its instance

$$(0 = x \ \wedge \ v_5 = 0) \ \equiv \ (0 = x \ \wedge \ v_5 = x) \quad (\dagger)$$

can be adopted as an adequate elimination equation for $\langle 0, x\rangle$. After the primitive widened rule for R w.r.t. ϕ,

$$0 = x \ \wedge \ v_1 + v_2 = 0 \ \Rightarrow \ 0 = x \ \wedge \ (v_1 = 0 \wedge v_2 = 0)$$

is generated, the disagreeing pair is eliminated: the lhs expression of equation $(\dagger)$, $0 = x \ \wedge \ v_5 = 0$, is unified with the lhs of the widened remove *rule, and the final rule,*

$$0 = x \ \wedge \ v_1 + v_2 = x \ \Rightarrow \ 0 = x \ \wedge \ (v_1 = 0 \wedge v_2 = 0)$$

is applicable to the conjecture.

ii. As described in example 8.2.2, the rule R. $(a = b) \Rightarrow p(c)$ cannot be $\emptyset$-matched against any of the subexpressions of ϕ,

$$x = b \wedge a = x$$

This situation does not change even in the presence of the unitary set Δ containing the logically valid equivalence[4]

[4] It suffices to consider that, given a set of logically valid formulae Γ and a formula ψ, $\Gamma \models \psi$ iff $\models \psi$.

$$E.\quad (v_1 = v_2 \wedge v_1 = v_3) \equiv (v_3 = v_2 \wedge v_1 = v_3)$$

Let then

$$R'.\quad a = b \wedge a = x \Rightarrow p(c) \wedge a = x$$

be the primitive widened rule for R w.r.t. ϕ and its left conjunct. For the disagreeing pair $\langle a, x\rangle$ between R' and ϕ, an instance of E, $(a = v_2 \wedge a = x) \equiv (x = v_2 \wedge a = x)$, provides an adequate elimination equation. The rule R'' generated after the application of the above equivalence,

$$R''.\quad x = b \wedge a = x \Rightarrow p(c) \wedge a = x$$

is then applicable to ϕ. □

Standard and extended elimination equations can be jointly used in the generation of rules.

Example 8.2.4 *Rule*

$$R.\quad v_1^{v_2} = 1 \Rightarrow v_1 = 1 \vee v_2 = 0$$

is not (syntactically) matchable to ϕ,

$$x^y \times w = z \times w \wedge z = 1$$

To generate a new rule for this formula, the pair $\langle x^y \times w = z \times w, v_1^{v_2} = 1\rangle$ is chosen in the first place. The topmost disagreement set admits $\langle x^y \times w, v_1^{v_2}\rangle$ as fatal disagreeing pair,

$\delta.$	v_1	exp	v_2	$=$	1
$\psi.$	x^y	$\times$	w	$=$	$z \times w$

Given the set of conditional equations Δ, containing

$$\begin{array}{rcl} v_4 \neq 0 \;\rightarrow\; v_3 = v_5 & \equiv & v_3 \times v_4 = v_5 \times v_4 \\ v_5 = v_6 \times v_7 \wedge v_8 = v_6 & \equiv & v_5 = v_8 \times v_7 \wedge v_8 = v_6 \end{array}$$

an instance of one of its elements,

$$v_6^{v_7} = v_5 \;\equiv\; v_6^{v_7} \times v_4 = v_5 \times v_4 \quad (v_4 \neq 0)$$

is unifiable with δ, and $\sigma_1 = \{^{v_1}/_{v_6}, ^{v_2}/_{v_7}, ^{1}/_{v_5}\}$ is the mgu. For the new rule, R',

$$v_1^{v_2} \times v_4 = 1 \times v_4 \;\Rightarrow\; v_1 = 1 \vee v_2 = 0 \quad (v_4 \neq 0)$$

no adequate standard elimination equation is available for any of the subsequent subproblems, hence extended adequate elimination equations can be sought. Given the disagreement pair,

$\delta'.$	$v_1^{v_2} \times v_4$	$=$	$\boxed{1}$	$\times$	v_4	
$\psi.$	$x^y \times w$	$=$	$\boxed{z}$	$\times$	w	

the lhs expression of $(v_5 = v_6 \times v_7 \wedge v_8 = v_6) \equiv (v_5 = v_8 \times v_7 \wedge v_8 = v_6)$, or rather a subexpression of it, can be unified with δ', by means of $\sigma_2 = \{^{v_1^{v_2} \times v_4}/_{v_5}, ^{1}/_{v_6}, ^{v_4}/_{v_7}\}$, and a widened rule R'',

$$v_1^{v_2} \times v_4 = v_8 \times v_4 \;\wedge\; v_8 = 1 \;\Rightarrow\; (v_1 = 1 \vee v_2 = 0) \wedge v_8 = 1 \quad (v_4 \neq 0)$$

is obtained. The final subproblem

$\delta''.$	$v_1^{v_2} \times v_4$	$=$	v_8	$\times$	v_4	$\wedge$	$v_8 = 1$
$\phi.$	$x^y \times w$	$=$	z	$\times$	w	$\wedge$	$z = 1$

admits a disagreement set which represents a substitution that matches δ'' against ϕ.

□

8.3 RUE-resolution, ECOP & RGM

As in the case of other difference reduction procedures, RGM operates under the principle that syntactically distinct expressions can be proved (semantically) equal through the gradual elimination of disagreeing pairs. RGM nonetheless is specialised to dealing with rule matching and, unlike RUE-resolution and ECOP, it handles both terms and formulae. From the strategic viewpoint, it can be regarded as a restricted version of ECOP with a series of additional constraints to improve efficiency.

The restrictions imposed upon the selection of elimination equations substantially reduce the search space and reinforce the difference reduction principle. Each disagreeing pair has to be eliminated in a single transformation step, i.e. the disagreeing pair of symbols have to be split between lhs and rhs expressions of the equation. There is a similarity in this respect between RGM and E-resolution, since the latter allows the generation of a resolvent only when all differences between complementary literals are raised. At the level of equality tree generation, however, which establishes the Δ-unifiability of terms, paramodulation is applied exhaustively, irrespectively of the possible effect on difference reduction. E-resolution, therefore, provides effective difference reduction only at one stage of the process, whereas RGM sticks to this guideline permanently.

Elimination equations must also unify with the expression being rewritten and match at least the upper context of occurrence of the expected symbol in the reference expression. As a result, the degree of unification for an elimination equation, as defined in the heuristic version of RUE-resolution, must necessarily be at least 70. This condition is not sufficient for adequacy though, since an equation may have measure 70 without actually unifying with any term of the inequality.

As in the case of ECOP, the innermost disagreement set is usually selected in the first place. This choice however is also influenced by the set of elimination equalities. The existence of a single adequate equation determines a disagreement set, or at least the disagreement pair that contains the disagreeing symbol, considering that one of the sides of the equation has to be unified with a subexpression of the expression being rewritten. In such cases, the choice of a disagreement set ceases to be an independent

parameter.

A substitution (not necessarily a partial unifier) still has to be chosen for those cases in which there are multiply occurring variables, when the mechanism examines all possible partial unifiers restricted to such variables. The rewriting tree is also pruned as a result of the requirement that, once a disagreement has been eliminated, no alternative solution for that pair is sought. This restriction is deductively harmless in those cases where there is a single way of carrying out the replacement of a symbol with another one.

Completeness is incorporated to RGM when the decomposition of disagreement elimination into a series of subproblems is allowed. Extended versions of RGM may be based on bounded subproblem decompositions, according to a pre-established number of intermediate steps. A higher number of intermediate elimination equations would be also required to accommodate the introduction of elimination equality graphs. Heuristic elements present in ECOP, particularly the context-sensitive functions based on various syntactic measures, could be then employed to reduce the search space[5].

Both ECOP and RGM have the ability to preserve solutions for subproblems, respectively under the form of equality graphs and derived rules. A most general derived rule represents the solution for a family of subproblems, which can be retrieved from a rulebase, when required, instead of being repeatedly generated by the mechanism. This feature is particularly relevant in extended versions of RGM, where multiple-step disagreement elimination increases the number of subproblems. The storage of solutions under the form of rules rather than graphs is particularly advantageous in the context of Δ-matching, where variables are instantiated in one of the expressions only.

The restriction of disagreement elimination to the rule level makes RGM suitable to cooperate with proof plans for normalisation, given that plans limit transformations at the conjecture level to specific syntactic tasks, e.g. the removal of symbols, whereas the removal of disagreements involves a more complex sequence of operations, as discussed in chapter 5. The main properties of RGM are summarised in table 8.2.

[5] See [Bläsius & Siekmann 88], p. 410-1.

Variable Instantiation	Disagreement Set Selection	Elimination Equality Selection	Other Features
mgpu (multiply occurring variables)	equality-base sensitive	minimally disagreeing equality	linked solution for subproblems incomplete

Table 8.2: RGM - Main properties

8.4 Conclusions

RGM is a procedure for equality reasoning where each equality link potentially eliminates a fatal disagreeing pair of subexpressions. It provides a substantial reduction of the search space, under the strict observance of the difference reduction principle. Strategic control is mainly centered around the selection of adequate elimination equations, which, in certain cases, determines the choice of a disagreement set as well.

Given the evidence that complete difference reduction mechanisms are too inefficient for general use, RGM is then preferable to RUE-resolution or ECOP for applications such as Δ-matching. Since RGM is suitable for representing certain weak forms of disagreement elimination, it can cooperate with generic proof plans in the extension of decidable classes of formulae.

Chapter 9

Decidable Classes for Peano Arithmetic

Heuristic functions and the rule generation mechanism may be incorporated in general-purpose plans to enhance their deductive strength and performance. Peano arithmetic and its extensions are relevant domains for the application of the resulting plans, due to the role of these theories in the representation of verification conditions.

A series of plans for the extension of decidable sublanguages is examined in section 9.1. Before they can be applied to *PA*, *remove* rules and elimination equations have to be selected. A list of recursive functions and relations frequently found in verification conditions is introduced in section 9.2. *Remove* rules for deviant symbols follow in section 9.3. To ensure the effectiveness of the process of subclass enlargement, the halting problem for sets of partial *remove* rules is discussed in section 9.4. Limitations for the extension of arithmetical decidable subclasses and sets of *remove* rules are examined in section 9.5, whereas the computation of members of extended classes is discussed in section 9.6.

9.1 General Purpose Plans

Complexity measure functions and order relations defined for decidable subclasses, deviant symbols and *remove* rules provide heuristic guidance for the extension of decidable sublanguages. Conditional methodicals allow the inclusion of these functions and order relations among the preconditions of primitive normalisation methods, whose

original definition is then kept unchanged. General-purpose plans are defined below from the composition of such constructs.

9.1.1 The Deciders

Decide1/1 contains the core strategy for the extension of decidable sublanguages. Whenever more than one decidable sublanguage is known, the function m_c is called to organise them into decreasing order of preference. The subplan *red_dec_cla*/2 then tries to perform the reduction. If it succeeds, the rewritten conjecture is eventually supplied to the corresponding decision procedure. Additional subplans are responsible for calling the function m_d and ordering deviant symbols with respect to the chosen sublanguage. The subplans *rem_dev_sym*/3-5 in particular control the removal of symbols and the elimination of disagreements, employing for this purpose the rule generation mechanism[1].

A second plan, *decide2*/1, has the same search control as *decide1*/1 for the selection of decidable sublanguages, the hierarchisation of the deviant symbols and the choice of *remove* rules. It is interfaced to a stronger version of RGM that explores conjecture subexpressions other than the particular one directly involved in semantic matching, as described in section 8.2.3. Thanks to this extension, given the conjecture

$$(x)(y)(z)(x = 0 \wedge y^2 + z = x)$$

and the rule $R.\ v_1 + v_2 = 0 \Rightarrow v_1 = 0 \wedge v_2 = 0$, a new rule,

$$R'.\quad v_3 = 0 \wedge v_1 + v_2 = v_3 \quad \Rightarrow \quad v_3 = 0 \wedge v_1 = 0 \wedge v_2 = 0$$

can be generated to convert the conjecture into

$$(x)(y)(z)(x = 0 \wedge y^2 = 0 \wedge z = 0)$$

which belongs to $\mathcal{L}_{SMA}$.

[1] The Prolog code for this plan and all others described in this chapter is listed in appendix F.

9.1.2 The Simplifiers

Some of the decision procedures integrated into theorem provers have been extended to perform additional transformation tasks. In the *Stanford Pascal Verifier*, the decision procedure for the quantifier-free subclass of Presburger arithmetic behaves as a simplifier: given an expression ϵ of the domain, it returns **true** if ϵ is a theorem, **false** if it is unsatisfiable, and a normalised expression if it is either a satisfiable invalid formula or a term. In *Nqthm*, the domain of the simplifier is also larger than the domain of its core decision procedure: if the input clause belongs to the decidable subclass, it is reduced to a boolean value, otherwise the resulting transformed formula is supplied to an inductive prover[2].

Neither *decide1*/1 nor *decide2*/1 has this ability, since, when a formula cannot be rewritten into a propositional constant, no output is generated. Another plan, *simplify*/1, on the other hand, explores the fact that, even if a formula is not reducible to a decidable sublanguage, the rewriting process simplifies it, in the sense that the number of occurrences of deviant symbols (or another measure for such symbols, as the one described in lemma 6.2.1) decreases. When the reduction fails, the output of *simplify*/1 consists of the initial formula and one simplified formula for each sublanguage, corresponding to the failed reduction attempts. Since these expressions are all equivalent to each other, they can be supplied to an alternative proving strategy, e.g. induction. The choice of a potentially most adequate version of the conjecture has to suit the requirements of the new proving strategy, and may be based on length, number of occurrences of quantifiers, etc.

The new plan has two separate rewriting modules, as represented in figure 9.1. *Simplify1*/2 is not interfaced to any difference reduction procedure and is much faster and deductively less powerful as a result. When a formula cannot be transformed into a decidable sublanguage by this subplan, both this formula and its simplified version are supplied to *simplify2*/2, which chooses the best (i.e. simplest according to certain heuristic criteria) candidate for the final rewriting stage, where RGM is called. As one of the criteria concerns the number of occurrences of deviant symbols, the sim-

[2] See [Nelson & Oppen 79], p. 246-7, and [Boyer & Moore 88].

plified formula is usually chosen. Rewriting halts whenever an element of a decidable sublanguage is obtained, or no decidable sublanguage is left.

A second simplifier, *weak_simplify*/1, imposes additional restrictions on the selection of equations from equality bases for the elimination of disagreements. It prevents the application of any equation of the form

$$v = t \quad \text{or} \quad t = v$$

where v is a variable. The presence of such equalities in difference reduction procedures increases the search space dramatically, due to the fact that individual variables can be unified with any term in the course of disagreement elimination[3].

9.1.3 The Rewriters

The effect of each of the three control factors described in section 6.3 — ordering sublanguages, deviant symbols and *remove* rules — over efficiency can be empirically assessed by means of rewriters specially devised for this purpose. Two groups have been built according to the presence or absence of a difference reduction mechanism, and, in each group, six distinct cases have been classified according to their main control features.

Rewrite1/4 exhaustively applies all the *remove* rules available to the system, until the input expression is reduced to one of the decidable sublanguages, whenever possible. Rules are applied in the order in which they are stored in the rule base. Each rule is exhaustively applied before the next one is called.

Rewrite2/5 is an extension of *rewrite1/4* that includes RGM to raise disagreements. As a result, whenever a rule is not applicable to a subexpression of a conjecture, the system tries to eliminate the disagreements before considering the application of the following rule.

Rewrite3/5, another extension of *rewrite1*/4, selects a decidable sublanguage before starting to apply rules, and, once it is selected, only *remove* rules for its deviant symbols are tested. No heuristic measures are available though for the selection of

[3] See [Bläsius & Siekmann 88], p. 403.

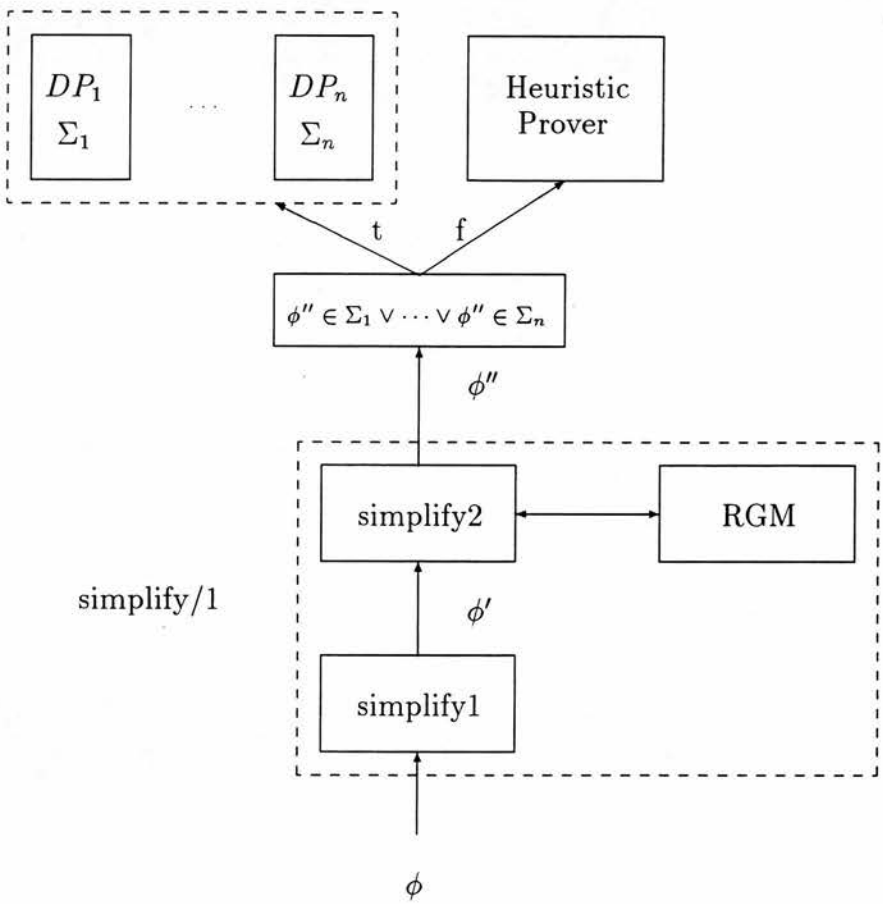


Figure 9.1: Plan *simplify*/1

sublanguages or rules. If the system fails to reduce a conjecture ϕ to a particular sublanguage, the next one is chosen and the process is repeated, until either ϕ is eventually reduced, or no decidable sublanguage is left. Since just one of the possible orderings for the removal of deviant symbols is examined, it is deductively less powerful than the rewriters that examine all of such orderings. *Rewrite3*/5, as well as the systems that follow, is interfaced to a difference reduction procedure.

Rewrite4/5 additionally restricts the application of *remove* rules to deviant subformulae, i.e. those containing a deviant symbol. Although it may look irrelevant at first, this criterion implicitly recognises the syntactic role of *remove* rules and prevents their use for any other purpose. Such a feature is *not* present in standard rewrite systems, where rules are not distinguished along syntactic lines and are tentatively applied to any subexpression of a conjecture. Since *rewrite4*/5 works in connection with RGM, when a subexpression of ϕ does not contain a deviant symbol, the application of the corresponding rule is not tried, and the mechanism of rule generation is not called either, thus considerably cutting down the time allocated to unfruitful disagreement elimination attempts. *Rewrite4*/5 examines every permissible ordering for the removal of deviant symbols that occur in a conjecture. As in the case of *rewrite3*/5, no heuristic criterion is used in the selection of a sublanguage: they are retrieved according to a fixed order.

Rewrite5/5, besides inheriting all the features of *rewrite4*/5, heuristically orders decidable sublanguages before rewriting starts. Decidable sublanguages are organised in decreasing order of complexity w.r.t. the conjecture. All the permissible permutations of deviant symbols that occur in the conjecture are tested, until the reduction is completed, whenever possible.

Finally, *rewrite6*/5 has all the features of *rewrite5*/5 and heuristically orders deviant symbols w.r.t. the selected decidable sublanguage.

The next system in this hierarchy includes heuristic functions to guide the selection of decidable sublanguages, deviant symbols and *remove* rules, and corresponds to the general-purpose proof plan *decide1*/1. All the rewriters, excluding *rewrite1*/4, have in principle the same deductive power as this plan, since they have access to the same rule base and are linked to RGM. Table 9.1 has a summary of the main features of the

Control Features	Disagreement Elimination	
	absence	presence
1. Strict application of *remove* rules	*rewrite1*/4	*rewrite2*/5
2. Decidable-class oriented rewriting	—	*rewrite3*/5
3. Deviant subexpression selection	—	*rewrite4*/5
4. Heuristically ordered decidable sublanguages	—	*rewrite5*/5
5. Heuristically ordered deviant symbols	—	*rewrite6*/5
6. Heuristically ordered *remove* rules	—	*decide*/1

Table 9.1: Rewrite Systems & Proof Plans for Normalisation

rewriters.

9.2 Subtheories & Extensions of *PA*

Since general-purpose plans capture the common structure of families of normalisers, they are applicable to more than a single theory. Plans devised for enlarging decidable sublanguages can be equally applied, therefore, to both *PA* and its conservative extensions. Among these extensions are those obtained by the introduction of new recursive functions and relations, which include user-defined operations. Before describing them, the complexity of specific decision procedures for arithmetical subtheories is examined.

9.2.1 Decidable Subtheories

A simple extension of *PA* is obtained when the symbol $<$ is added to the subjacent language to represent the standard (irreflexive) order relation over natural numbers. When the same symbol is added to the language of Presburger arithmetic, resulting in $\mathcal{L}_{PrA^<} = \{0, 1, s, +, <\}$, a new decidable theory, $PrA^<$, is obtained. Any decision procedure for *PrA*, $PrA^<$ or *SMA* is respectively adequate for any recursive subclass of formulae of $\mathcal{L}_{PrA}$, $\mathcal{L}_{PrA^<}$ or $\mathcal{L}_{SMA}$. Nevertheless, from the point of view of time and

space complexity, procedures valid for the same class may differ radically.

The most efficient known procedure for PrA, Cooper's algorithm, has deterministic time complexity of the order of $2^{2^{2^n}}$ for the worst case, where n denotes the length of the input formula. A more acceptable behaviour can be observed in certain subsets of formulae of $\mathcal{L}_{PrA}$. The decision process leads to the combinatorial expansion of the input formula, and depends on the number of alternate quantifiers in the prefix, among other factors.

Concerning PrA^*, the quantifier-free class of formulae of $\mathcal{L}_{PrA^*}$ admits a decision procedure based on the SUP-INF method, whose complexity is not worse than 2^n, n being the length of the formula. Even if restricted to this class, Cooper's algorithm seems to be less efficient than the SUP-INF based procedure, due to additional parameters that affect formula expansion[4].

For SMA, although there seems to be no study about its decision procedures or their complexity, there are at least two proofs for its decidability, one based on quantifier elimination and the other on the fundamental theorem of arithmetic. It is likely that a procedure directly obtained from the first proof is more complex than Cooper's algorithm[5].

Complexity considerations may influence the choice of both decidable subclasses and *remove* rules. Given the decidability of PrA^* and the availability of an efficient procedure for its quantifier-free subclass, the choice of $\mathcal{L}_{PrA^*}$ instead of $\mathcal{L}_{PrA}$ as decidable sublanguage eliminates $<$ from the list of deviant symbols. Moreover, for quantifier-free formulae involving $<$, given the options

(a) the reduction to $\mathcal{L}_{PrA}$ through the removal of $<$ by the application of a total *remove* rule,

$$v_1 < v_2 \quad \Rightarrow \quad (\exists v_3)(v_3 \neq 0 \wedge v_1 + v_3 = v_2)$$

[4] Cooper's algorithm is described in [Cooper 72]. Some remarks on the efficiency of this algorithm and of the SUP-INF procedure can be found in [Shostak 77], p. 530.

[5] None of the texts quoted in this dissertation makes any reference to decision procedures for SMA. Also, the decidability of SMA is seldom mentioned in the literature; amongst the few exceptions are [Boolos & Jeffrey 89], p. 219 and [Epstein & Carnielli 89], p. 202. Proofs for the decidability of SMA can be found in [Skolem 70] and [Mostowski 52].

which nonetheless introduces occurrences of quantifiers in the rewritten formula, and

(b) the reduction to $\mathcal{L}_{PrA^*}$, in which case $<$ does not have to be eliminated,

the first solution excludes the use of the SUP-INF procedure for the decision of the resulting quantified formula, whereas the second solution is still compatible with it. As a result, $\mathcal{L}_{PrA^*}$ and $\mathcal{L}_{SMA}$ become the new pair of decidable sublanguages.

Concerning *remove* rules, as already discussed in section 6.3.3, the use of those that introduce quantifiers in the course of rewriting should be discouraged, both w.r.t. PrA^* and SMA, due to the effect of quantifiers on complexity. For PrA^*, the presence of quantifiers excludes the use of decision procedures for its quantifier-free class. For SMA, which seems to require the use of procedures directly based on quantifier elimination, the number of quantifiers in a formula directly influences the size of its expansion.

9.2.2 Definitional Extensions

A *number theoretic function* (of arity m) is any function that has $\mathbb{N}^m$ as domain and $\mathbb{N}$ as range. A *number theoretic relation* (of arity m) is any subset of $\mathbb{N}^m$. Number theoretic functions and relations form in principle the universe from which extensions of PA are derived. Some of these functions and relations are nonetheless representable in PA.

Definition 9.2.1 (Weak & Strong Representability)

Let $\mathcal{L}'_{PA}$ be the expansion of $\mathcal{L}_{PA}$ generated by the inclusion of infinitely many individual constant, function and predicate symbols of each arity, and let $\mathfrak{N}'$ be a structure which expands $\mathfrak{N} = \langle \mathbb{N}, 0, 1, s, +, \times \rangle$ to $\mathcal{L}'_{PA}$. Let T be a theory in $\mathcal{L}_{PA}$.

i. A number-theoretic relation $p^{\mathfrak{N}'}$ (of arity j) is weakly representable *in T iff there is a formula $\phi(v_1, \ldots, v_j)$ in $\mathcal{L}_{PA}$ such that*

$$\mathfrak{N}' \models p(v_1, \ldots, v_j)[\alpha] \quad \textit{iff} \quad T \models \phi(s^{n_1}(0), \ldots, s^{n_j}(0))$$

for any assignment $\alpha = \{^{n_1}/_{v_1}, \ldots, ^{n_j}/_{v_j}\}$, with $n_1, \ldots, n_j \in \mathbb{N}$.

ii. A number-theoretic relation $p^{\mathfrak{N}'}$ *(of arity* j*) is* representable *in* T *iff there is a formula* $\phi(v_1, \ldots, v_j)$ *in* $\mathcal{L}_{PA}$ *such that*

$$\begin{array}{llll} \textit{if} & \mathfrak{N}' \models p(v_1, \ldots, v_j)[\alpha] & \textit{then} & T \models \phi(s^{n_1}(0), \ldots, s^{n_j}(0)) \\ \textit{if} & \mathfrak{N}' \not\models p(v_1, \ldots, v_j)[\alpha] & \textit{then} & T \models \neg\phi(s^{n_1}(0), \ldots, s^{n_j}(0)) \end{array}$$

for any assignment $\alpha = \{^{n_1}/_{v_1}, \ldots, ^{n_j}/_{v_j}\}$*, with* $n_1, \ldots, n_j \in \mathbb{N}$.

iii. A number-theoretic function $f^{\mathfrak{N}'}$ *(of arity* j*) is* representable *in* T *iff there is a formula* $\phi(v_1, \ldots, v_{j+1})$ *in* $\mathcal{L}_{PA}$ *such that, if* $\mathfrak{N}' \models f(v_1, \ldots, v_j)[\alpha] = v_{j+1}[\alpha]$*, then*

$$T \models \phi(s^{n_1}(0), \ldots, s^{n_j}(0), v_{j+1}) \equiv (v_{j+1} = s^{n_{j+1}}(0))$$

for any assignment $\alpha = \{^{n_1}/_{v_1}, \ldots, ^{n_{j+1}}/_{v_{j+1}}\}$*, with* $n_1, \ldots, n_{j+1} \in \mathbb{N}$.

iv. A number-theoretic function $f^{\mathfrak{N}'}$ *(of arity* j*) is* strongly representable *in* T *iff there is a formula* $\phi(v_1, \ldots, v_{j+1})$ *in* $\mathcal{L}_{PA}$ *such that, if* $\mathfrak{N}' \models f(v_1, \ldots, v_j)[\alpha] = v_{j+1}[\alpha]$*, then*

$$T \models \phi(s^{n_1}(0), \ldots, s^{n_{j+1}}(0))$$

and $T \models (\exists | v_{j+1})\phi(v_1, \ldots, v_j, v_{j+1})$*, for any assignment* $\alpha = \{^{n_1}/_{v_1}, \ldots, ^{n_{j+1}}/_{v_{j+1}}\}$*, with* $n_1, \ldots, n_{j+1} \in \mathbb{N}$.

If $p^{\mathfrak{N}'}$ is representable in *PA* by a formula $\phi(v_1, \ldots, v_j)$, then, according to definition B.2.4, ϕ is a *possible definition for* p *in PA*. A definitional extension of *PA* is obtained after the introduction of

$$p(v_1, \ldots, v_j) \equiv \phi(v_1, \ldots, v_j)$$

as a new axiom. Let $f^{\mathfrak{N}'}$, on the other hand, be strongly representable by a formula $\psi(v_1, \ldots, v_{j+1})$. Given that ψ must satisfy the condition

$$\psi(v_1, \ldots, v_{j+1}) \wedge \psi(v_1, \ldots, v_j, {}^{v_{j+2}}/_{v_{j+1}}) \supset v_{j+1} = v_{j+2}$$

which ensures the unicity of the represented function for a given input $v_1, \ldots, v_j$, then, also according to definition ii, ψ is a *possible definition for* f *in PA*. A definitional extension of *PA* follows from the introduction of a new axiom,

$$f(v_1, \ldots, v_j) = v_{j+1} \quad \equiv \quad \phi(v_1, \ldots, v_{j+1})$$

Since definitional extensions are conservative, all the decidable subclasses for PA, according to theorem C.5.2, preserve their decidability and type when representable functions and relations are introduced. Moreover, any representable function or relation admits a complete set of rules for their removal in terms of $\mathcal{L}_{PA}$.

Lemma 9.2.1 *Let T be a theory in $\mathcal{L}_{PA}$. If $f^{\mathfrak{N}'}$ is strongly representable in T, and $p^{\mathfrak{N}'}$ is representable in T, then there are T-valid complete sets of* remove *rules for f and p w.r.t. $\mathcal{L}_{PA}$.*

PROOF. Since $p^{\mathfrak{N}'}$ is representable in T, its defining axiom is the source of a total *remove* rule, $p(v_1, \ldots, v_j) \Rightarrow \phi(v_1, \ldots, v_j)$, where $\phi(v_1, \ldots, v_j)$ belongs to $\mathcal{L}_{PA}$. In the case of a function $f^{\mathfrak{N}'}$, given a possible definition

$$f(v_1, \ldots, v_j) = v_{j+1} \quad \equiv \quad \psi(v_1, \ldots, v_{j+1})$$

where ψ is also a formula of $\mathcal{L}_{PA}$, a complete *remove* rule scheme for f has the form

$$\rho[\![f(v_1, \ldots, v_j)]\!] \quad \Rightarrow \quad (\exists v)(\psi(v_1, \ldots, v_j, v) \wedge \rho[\![^v/_{f(v_1, \ldots, v_j)}]\!])$$

where v does not occur in ρ. ∎

For function symbols, whenever a possible definition $\phi(v_1, \ldots, v_{j+1})$ is an equation of the form $v_{j+1} = t$, where t is a term of $\mathcal{L}_{PA}$, then

$$f(v_1, \ldots, v_j) \quad \Rightarrow \quad t$$

corresponds to a total *remove* rule for f. Given that

(i) every recursive relation is representable in PA, and

(ii) every recursive function is strongly representable in PA,

it follows from the above lemma that the removal of symbols that stand for new recursive functions and relations can be, at least in principle, entirely dealt with by complete sets of rules[6].

9.2.3 Recursive Functions and Relations

Subtraction, exponentiation and division are some of the recursive functions that frequently occur in verification conditions. A similar remark applies to recursive predicates such as order relations. The complete list of new symbols added to $\mathcal{L}_{PA}$, as well as their intended meaning, is given in tables 9.2 and 9.3. The resulting expanded language, $\mathcal{L}_{PA^*}$, has two individual constant symbols, 0 and 1, fifteen function symbols,

$$\mathrm{s},\ +,\ \times,\ -,\ \mathrm{pr},\ \mathrm{exp},\ /,\ \mathrm{double},\ \mathrm{Sum}_n,\ \mathrm{half},\ \mathrm{gcd},\ \mathrm{gfc},\ \mathrm{rmdr},\ \mathrm{min}_n,\ \mathrm{max}_n$$

and nine predicate symbols,

$$<,\ \leq,\ >,\ \geq,\ |,\ \mathrm{even},\ \mathrm{prime},\ \mathrm{prime}_1,\ \equiv_n$$

PA^* is obtained as the definitional extension of PA in which the above symbols are defined by the $\mathcal{L}_{PA}$-formulae that represent in PA the corresponding functions and relations. Order relations, for instance, require quantified formulae,

$$\begin{aligned} v_1 < v_2 &\equiv (\exists v_3)(v_3 \neq 0 \wedge v_1 + v_3 = v_2) \\ v_1 \leq v_2 &\equiv (\exists v_3)(v_1 + v_3 = v_2) \\ v_1 > v_2 &\equiv (\exists v_3)(v_3 \neq 0 \wedge v_2 + v_3 = v_1) \\ v_1 \geq v_2 &\equiv (\exists v_3)(v_2 + v_3 = v_1) \end{aligned}$$

and the same applies to the other predicate symbols[7],

---

[6] Recursive functions and relations are also representable in Richard arithmetic, which is weaker than PA. See for instance [Monk 76], p. 248. Also, a function is representable iff it is strongly representable, as mentioned in [Mendelson 87], p. 130.

[7] To simplify the presentation of these definitions, some of the newly introduced symbols are adopted as abbreviations for $\mathcal{L}_{PA}$ expressions, as for instance the symbol for divisibility in the definition of prime_1, a relation defined in [Boyer & Moore 79]. Also, $(v_1 \not< v_2)$ is an abbreviation for $\neg(v_1 < v_2)$.

$$
\begin{aligned}
v_1|v_2 &\equiv v_1 \neq 0 \wedge (\exists v_3)(v_1 \times v_3 = v_2) \\
v_1 \equiv_n v_2 &\equiv (\exists v_3)(\exists v_4)(\exists v_5)(v_1 = n \times v_3 + v_5 \wedge v_2 = n \times v_4 + v_5), \quad \text{for all } n \in \mathbb{N} \\
\text{even}(v_1) &\equiv (\exists v_2)(v_1 = v_2 + v_2) \\
\text{prime}(v_1) &\equiv v_1 \neq 0 \wedge (v_2)(v_3)(v_1 = v_2 \times v_3 \supset ((v_2 = 1 \wedge v_3 = v_1) \vee (v_2 = v_1 \wedge v_3 = 1))) \\
\text{prime}_1(v_1, v_2) &\equiv v_2 \neq 0 \wedge (v_3)((v_3|v_1 \wedge 1 < v_3) \supset (v_2 < v_3))
\end{aligned}
$$

Some of the new functions can be represented by means of equations,

$$
\begin{aligned}
\text{double}(v_1) = v_2 &\equiv v_1 + v_1 = v_2 \\
\text{Sum}_n(v_1, \ldots, v_n) = v_{n+1} &\equiv v_{n+1} = v_1 + \cdots + v_n, \quad \text{for all } n \in \mathbb{N}
\end{aligned}
$$

whereas others require composite formulae,

$$
\begin{aligned}
v_1 - v_2 = v_3 &\equiv (v_2 < v_1 \wedge v_1 = v_2 + v_3) \vee (v_2 \not< v_1 \wedge v_3 = 0) \\
\text{pr}(v_1) = v_2 &\equiv (1 < v_1 \wedge v_1 = s(v_2)) \vee (1 \not< v_1 \wedge v_2 = 0) \\
\text{half}(v_1) = v_2 &\equiv v_1 = v_2 + v_2 \vee v_1 = s(v_2 + v_2) \\
\min_n(v_1, \ldots, v_n) = v_{n+1} &\equiv \textstyle\bigwedge_{i=1}^{n}(v_i \not< v_{n+1}) \wedge \bigvee_{j=1}^{n}(v_{n+1} = v_j), \quad \text{for all } n \in \mathbb{N} \\
\max_n(v_1, \ldots, v_n) = v_{n+1} &\equiv \textstyle\bigwedge_{i=1}^{n}(v_i \not< v_{n+1}) \wedge \bigvee_{j=1}^{n}(v_{n+1} = v_j), \quad \text{for all } n \in \mathbb{N}
\end{aligned}
$$

For some of the functions, the representation in $\mathcal{L}_{PA}$ involves a considerable enlargement of the original formula, even when some of the new symbols are adopted as abbreviations for $\mathcal{L}_{PA}$ expressions.

$$
\begin{aligned}
\gcd(v_1, v_2) = v_3 &\equiv v_3|v_1 \wedge v_3|v_2 \wedge (v_4)((v_4|v_1 \wedge v_4|v_2) \supset v_4|v_3) \\
v_1/v_2 = v_3 &\equiv ((v_2 = 0 \vee v_1 < v_2) \wedge v_3 = 0) \\
&\quad \vee (v_2 \neq 0 \wedge v_1 \not< v_2 \wedge (\exists v_4)(v_4 < v_2 \wedge v_3 \times v_2 + v_4 = v_1)) \\
\text{gfc}(v_1, v_2) = v_3 &\equiv ((v_1 \leq 1 \vee \text{prime}(v_1)) \wedge v_3 = v_1) \\
&\quad \vee (v_3|v_1 \wedge (v_4)((v_4|v_1 \wedge v_4 \leq v_2) \supset v_4 \leq v_3)) \\
\text{rmdr}(v_1, v_2) = v_3 &\equiv (v_2 = 0 \wedge v_3 = v_1) \vee (v_2 \neq 0 \wedge v_3 < v_2 \wedge (\exists v_4)(v_1 = v_4 \times v_2 + v_3))
\end{aligned}
$$

The only function symbol of $\mathcal{L}_{PA^*}$ still missing in the above list, exp, requires the equation

$$
v_1^{v_2} = v_3 \equiv v_3 = \delta(e(v_1, v_2), v_2)
$$

where

$$
\begin{aligned}
e(v_1, v_2) &= \mu v_3[\delta(v_3, 0) = 1 \wedge (\forall v_4)(v_4 < v_2 \supset (\delta(v_3, v_4 + 1) = \delta(v_3, v_4) \times v_1))] \\
\delta(v_1, v_2) &= \beta(K(v_1), L(v_1), v_2) \\
K(v_1) &= \mu v_2[(\exists v_3)(v_3 \leq v_1 \wedge J(v_2, v_3) = v_1)] \\
L(v_1) &= \mu v_2[(\exists v_3)(v_3 \leq v_1 \wedge J(v_3, v_2) = v_1)] \\
J(v_1, v_2) &= H((v_1 + v_2) \times (v_1 + v_2 + 1)) + v_1 \\
H(v_1) &= \mu v_2[v_1 \leq v_2 + v_2]
\end{aligned}
$$

and μ is the minimalisation operator. $\beta(v_1, v_2, v_3)$ denotes the Gödel β-function, which is strongly representable in PA by the formula

$$(\exists v_5)(v_1 = (1 + (v_3 + 1) \times v_2) \times v_5 + v_4 \wedge v_4 < 1 + (v_3 + 1) \times v_2)$$

Once formulae that strongly represent the functions K, L, H and μ are introduced in the equation that defines exponentiation, the resulting $\mathcal{L}_{PA}$ expression is highly complex from the syntactic viewpoint (i.e. length)[8].

9.3 Arithmetical *remove* rules

When efficiency is ignored, given that there are complete sets of *remove* rules for every new predicate and function symbol of $\mathcal{L}_{PA^*}$ in terms of $\mathcal{L}_{PA}$, the reduction problem for both $\mathcal{L}_{PA}$ and its expansion are essentially identical. Provided that the set of total and complete rules is normalised, all the new symbols can be eliminated in a first rewriting stage, where no search control is required. As control matters only after a formula of $\mathcal{L}_{PA}$ is obtained, there would be no reason to examine conjectures outside this language. In the set of formulae of $\mathcal{L}_{PA}$, on the other hand, partial rules have an essential role from the deductive point of view, since the undecidability of PA means that deviant symbols cannot be removed from all contexts.

If the complexity of the rewriting mechanism is taken into account, there are cases where, to prevent the introduction of quantifiers, additional deviant symbols or heavily

[8] The representation of exponentiation in PA is examined in [Enderton 72], p. 245 - 249. As discussed in chapter 11, future extensions of the rule base may include implication rules as well, as for instance

$$x^y = z \quad \Rightarrow \quad (y = 0 \wedge z = 1) \vee (y = 1 \wedge x = z) \vee 1 < y$$

Symbol	Well-formed term	Intended meaning
exp	u^v	iterated product of u by itself (v iterations)
double	double(v)	double of v
half	half(v)	arithmetical half of v
/	v/u	arithmetical quotient of v and u
rmdr	rmdr(v, u)	remainder of the quotient of v and u
gcd	gcd(v, u)	greatest common divider of v and u
$-$	$v - u$	arithmetical difference between v and u
pr	pr(v)	predecessor of v
gfc	gfc(u, v)	greatest factor of u less than or equal to v
$\min_n$	$\min_n(v_1, \ldots, v_n)$	minimum element of the n-tuple
$\max_n$	$\max_n(v_1, \ldots, v_n)$	maximum element of the n-tuple
Sum_n	$\mathrm{Sum}_n(v_1, \ldots, v_n)$	sum of the n-tuple

Table 9.2: Recursive Functions

Symbol	Well-formed formula	Intended meaning
$<$	$u < v$	u is less than v
$\leq$	$u \leq v$	u is less than or equal to v
$>$	$u > v$	u is greater than v
$\geq$	$u \geq v$	u is greater than or equal to v
$\mid$	$u \mid v$	v is divisible by u
even	even(v)	v is a natural even number
prime	prime(v)	v is a natural prime number
prime_1	$\mathrm{prime}_1(u, v)$	u has no non-unitary divisor less than or equal to v
$\equiv_n$	$u \equiv_n v$	u and v are equivalent modulo n

Table 9.3: Recursive Relations

expanded rewritten formulae, partial rules for completely removable symbols have to be given preference. One of the most typical examples concerns exponentiation, which, besides the total rule derived from its explicit definition, also admits partial rules derived e.g. from the base equation of its recursive definition,

$$\begin{array}{rcl} v^0 & = & 1 \\ v^{s(u)} & = & v \times v^u \end{array}$$

As a result, the transformation of formulae of $\mathcal{L}_{PA^*}$ cannot be restricted to the study of $\mathcal{L}_{PA}$.

9.3.1 Totally Removable Symbols

Proof plans can be used for the extension of decidable subclasses only after a set of *remove* rules has been selected. The definitions for the new symbols of $\mathcal{L}_{PA^*}$ in terms of $\mathcal{L}_{PA}$, described in section 9.2.3, are the primary source for such rules. The symbols for which a total *remove* rule is available are

$$<,\ >,\ \leq,\ \geq,\ \text{even},\ \text{prime},\ \text{prime}_1,\ |,\ \text{s},\ \text{double},\ \text{pr}$$

Given the current rewriting strategy, according to which the removal of symbols is divided into two blocks, made up of totally and partially removable symbols, the application of total rules has to come first, since at least some of them introduce partially removable symbols that are deviant w.r.t. one of the decidable sublanguages. For instance, the rule for | introduces $\times$, which is deviant w.r.t. $\mathcal{L}_{PrA^*}$, and the rule for *even* introduces $+$, which is deviant w.r.t. $\mathcal{L}_{SMA}$.

'$<$' is the only totally removable symbol for which additional partial rules are available in the rule base. Given that it belongs to one of the decidable sublanguages, the rules for the other order relation symbols have not been derived from their definitions in *PA*, but rather from

$$\begin{array}{rcl} v_1 \leq v_2 & \equiv & (v_1 = v_2) \vee (v_1 < v_2) \\ v_1 \geq v_2 & \equiv & (v_1 = v_2) \vee (v_2 < v_1) \\ v_1 > v_2 & \equiv & v_2 < v_1 \end{array}$$

which have the advantage of being quantifier-free. Once all total rules are normalised, $\mathcal{L}_{PA^*}$ is replaced with

$$\mathcal{L}_{PA^{\bullet\bullet}} = \{0, 1, +, \times, \exp, -, \text{half}, /, \text{gfc}, \text{rmdr}, \gcd, \min_n, \max_n, <\}$$

where $<$ is the only predicate symbol left. The new sets of deviant symbols are $\{\times, \exp, -, \text{half}, /, \text{gfc}, \text{rmdr}, \gcd, \min_n, \max_n\}$ (with respect to $\mathcal{L}_{PrA^{\bullet}}$) and $\{+, \exp, -, \text{half}, /, \text{gfc}, \text{rmdr}, \gcd, \min_n, \max_n, <\}$ (with respect to $\mathcal{L}_{SMA}$). Apart from its total rule,

$$v_1 < v_2 \quad \Rightarrow \quad (\exists v_3)(v_3 \neq 0 \wedge v_1 + v_3 = v_2)$$

three additional quantifier-free partial rules for $<$,

$$\begin{array}{lcl} 0 < 0 & \Rightarrow & \bot \\ 0 < v & \Rightarrow & v \neq 0 \\ 1 < v & \Rightarrow & (v \neq 0) \wedge (v \neq 1) \end{array}$$

have been selected. Since $<$ is not deviant w.r.t. $\mathcal{L}_{PrA^{\bullet}}$, these rules are used only when $\mathcal{L}_{SMA}$ is the target decidable sublanguage. When the total rule for $<$ is applied in the first place, the introduction of occurrences of $+$ may prevent a formula from being reduced to *SMA*.

Example 9.3.1 *Given the conjecture* $x^2 \times y \neq z^3 \wedge 1 < x \times y$, *the application of the total* remove *rule for* $<$ *generates*

$$x^2 \times y \neq z^3 \ \wedge \ (\exists w)(w \neq 0 \wedge 1 + w = x \times y)$$

which cannot be reduced to either $\mathcal{L}_{PrA^{\bullet}}$ *or* $\mathcal{L}_{SMA}$ *by the current set of* remove *rules. If a partial rule that does not introduce new occurrences of deviant symbols or quantifiers,*

$$1 < v \quad \Rightarrow \quad (v \neq 0 \wedge v \neq 1)$$

is chosen instead, the resulting rewritten formula,

$$x^2 \times y \neq z^3 \ \wedge \ x \times y \neq 0 \ \wedge \ x \times y \neq 1$$

now belongs to $\mathcal{L}_{SMA}$. □

Symbol	Rule		
$-$	$v_1 - v_2 = v_3$	$\Rightarrow$	$(v_2 < v_1 \wedge v_1 = v_2 + v_3) \vee (v_2 \not< v_1 \wedge v_3 = 0)$
	$v_1 - v_2 < v_3$	$\Rightarrow$	$(v_2 < v_1 \wedge v_1 < v_2 + v_3) \vee (v_2 \not< v_1 \wedge v_3 \neq 0)$
	$v_1 < v_2 - v_3$	$\Rightarrow$	$v_3 < v_2 \wedge v_1 + v_3 < v_2$
	$v_1 + (v_2 - v_3) = v_4$	$\Rightarrow$	$(v_3 < v_2 \wedge v_1 + v_2 = v_3 + v_4) \vee (v_3 \not< v_2 \wedge v_1 = v_4)$
	$v_1 = v_2 + (v_3 - v_4)$	$\Rightarrow$	$(v_4 < v_3 \wedge v_1 + v_4 = v_2 + v_3) \vee (v_4 \not< v_3 \wedge v_1 = v_2)$
	$v_1 = (v_2 - v_3) + v_4$	$\Rightarrow$	$(v_3 < v_2 \wedge v_1 + v_3 = v_2 + v_4) \vee (v_3 \not< v_2 \wedge v_1 = v_4)$
	$v_1 + (v_2 - v_3) < v_4$	$\Rightarrow$	$(v_3 < v_2 \wedge v_1 + v_2 < v_3 + v_4) \vee (v_3 \not< v_2 \wedge v_1 < v_4)$
	$v_1 < v_2 + (v_3 - v_4)$	$\Rightarrow$	$(v_4 < v_3 \wedge v_1 + v_4 < v_2 + v_3) \vee (v_4 \not< v_3 \wedge v_1 < v_2)$
exp	$v_1^{v_2} = 0$	$\Rightarrow$	$v_1 = 0 \wedge v_2 \neq 0$
	$v_1^{v_2} = 1$	$\Rightarrow$	$v_1 = 1 \vee v_2 = 0$
	$v_1^{v_2} = v_1$	$\Rightarrow$	$v_1 = 1 \vee v_2 = 1 \vee (v_1 = 0 \wedge v_2 \neq 0)$
	$v_1^{v_2} = v_2$	$\Rightarrow$	$v_1 = 1 \wedge v_2 = 1$
	$v_1^{v_2} < v_1$	$\Rightarrow$	$v_2 = 0 \wedge v_1 \neq 0 \wedge v_1 \neq 1$
	$v_1^{v_2} < v_2$	$\Rightarrow$	$(v_1 = 0 \wedge v_2 \neq 0) \vee (v_1 = 1 \wedge v_2 \neq 0 \wedge v_2 \neq 1)$
	$v_1 < v_1^{v_2}$	$\Rightarrow$	$(v_1 = 0 \wedge v_2 = 0) \vee (v_1 \neq 0 \wedge v_1 \neq 1 \wedge v_2 \neq 0 \wedge v_2 \neq 1)$
	$v_2 < v_1^{v_2}$	$\Rightarrow$	$v_2 = 0 \vee (v_1 \neq 0 \wedge v_1 \neq 1)$

Table 9.4: Partial rules for Subtraction and Exponentiation

For this reason, in the particular case of $<$, total and partial rules coexist in the rule base. For subtraction, to avoid the existential quantifier of the *remove* rule-scheme,

$$\phi[\![v_1 - v_2]\!] \Rightarrow (\exists v_3)(((v_2 < v_1 \wedge v_1 = v_3 + v_2) \vee (v_2 \not< v_1 \wedge v_3 = 0)) \wedge \phi[\![^{v_3}/(v_1 - v_2)]\!])$$

an incomplete set of quantifier-free partial rules has been chosen instead, the first of which is the definition of $-$ in $\mathcal{L}_{PA^*}$. Eight distinct contexts are covered. For exponentiation, the selected partial rules avoid the main disadvantage of its total rule, i.e. the size of the expanded formula. Both groups of rules are listed in table 9.4. Quantified rules have been allowed for the remaining function symbols, partly due to the fact that their possible definitions in $\mathcal{L}_{PA}$ already contain quantifiers. They are actually instances of the corresponding *remove* rule schemes. gcd, for instance, includes

$$\begin{aligned} \gcd(v_1, v_2) = v_3 &\Rightarrow v_3 | v_1 \wedge v_3 | v_2 \wedge (v_4)((v_4 | v_1 \wedge v_4 | v_2) \supset v_4 | v_3) \\ \gcd(v_1, v_2) < v_3 &\Rightarrow (\exists v_4)(v_4 < v_3 \wedge v_4 | v_1 \wedge v_4 | v_2 \wedge (v_5)((v_5 | v_1 \wedge v_5 | v_2) \supset v_5 | v_4)) \end{aligned}$$

The complete arithmetical rule set, $\mathcal{R}_{PA^*}$, can be found in appendix H.

9.3.2 Partially Removable Symbols

Both + and × have only partial rules in the rule base $\mathcal{R}_{PA^*}$. All of the chosen rules are quantifier-free. Most of them do not introduce deviant symbols, and only a few contain < in their rhs expressions. They can be classified in three main groups. First, there are those that *simplify* expressions, through their reduction either to proper subexpressions,

$$\begin{aligned} v + 0 &\Rightarrow v \\ v \times 1 &\Rightarrow v \end{aligned}$$

or to shorter expressions, unrelated to the original ones,

$$\begin{aligned} \mathrm{s}(0) &\Rightarrow 1 \\ v^0 &\Rightarrow 1 \end{aligned}$$

There are also those which *decompose* atomic formulae into boolean combinations of simpler atoms, where the arguments in the new subformulae are subterms of the arguments of the original atom, or individual constants of the underlying language, e.g.

$$\begin{aligned} v_1 + v_2 = 0 &\Rightarrow v_1 = 0 \wedge v_2 = 0 \\ v_1 < 1 + v_2 &\Rightarrow v_1 = v_2 \vee v_1 < v_2 \\ v_1 \times v_2 = v_1 \times v_3 &\Rightarrow v_1 = 0 \vee v_2 = v_3 \end{aligned}$$

A final group has rules which *replace* formulae by propositional constants, such as

$$v_1 + v_2 < 0 \Rightarrow \bot$$

In spite of deductive limitations, partial rules play a major role when it comes to efficiency, due to the absence of explicit occurrences of any predicate or function symbols (other than = or <) in their rhs expressions, and the limited expansion of the rewritten formula, when compared to some of the total rules. The set of partial rules is also listed in appendix H.

9.3.3 The Equality Base

The application of *remove* rules becomes blocked in the presence of fatal disagreements. To eliminate them, difference reduction procedures such as RGM (the *rule generation*

mechanism) employ equations in the construction of equality graphs, where neighbouring expressions are unifiable or matchable. Since *decide1*/1, *decide2*/1, *simplify*/1 and *weak_simplify*/1 are interfaced to RGM, a set of equations must then be available to these plans[9]. The algebraic properties of some of the symbols, specially their commutativity, associativity and distributivity w.r.t. other symbols, are the primary source of equations. The set of *remove* rules itself is an additional source, since they are derived from either equalities or equivalences. Besides the absence of certain symbols in their rhs expressions, they can also raise disagreements, as for instance one of the rules for $+$,

$$v_1 + v_2 = v_1 \quad \Rightarrow \quad v_2 = 0$$

whose instance

$$v_1 + S(t_1, \ldots, t_n) = v_1 \quad \Rightarrow \quad S(t_1, \ldots, t_n) = 0$$

is an elimination equation for the pair $\langle +, S \rangle$, where S is any function symbol.

A substantial reduction in the search space is obtained when rules whose lhs expressions are terms are excluded from the generation of new rules, since terms are potentially matchable to any subterm of the conjecture that contains the relevant deviant symbol. Also, for equality bases such as $\mathcal{E}_{PA^*}$, where the sides of each equation are either both terms or both atoms, the selection of disagreement sets between *remove* rules and conjectures can be limited to atomic subformulae of the conjecture, given that, under these circumstances, semantic matching takes place only between atoms or terms. The effects of both restrictions are examined in the next example.

Example 9.3.2 *Suppose the conjecture*

$$y \times z^2 < x \ \wedge \ x \times (y + z) = y \times x$$

has to be reduced to $\mathcal{L}_{SMA}$, *and that the available* remove *rules for* $+$ *are*

[9] RGM is described in chapter 8. The complete set of arithmetical equations $\mathcal{E}_{PA^*}$ can be found in appendix H.

$$
\begin{array}{lrcl}
R_1 & v + 0 & \Rightarrow & v \\
R_2 & 0 + v & \Rightarrow & v \\
R_3 & v_1 + v_2 = 0 & \Rightarrow & v_1 = 0 \wedge v_2 = 0 \\
R_4 & v_1 + v_2 = v_1 & \Rightarrow & v_2 = 0
\end{array}
$$

R_1 and R_2 are matchable to terms, whereas R_3 and R_4 are matchable to atoms. Since none of them is applicable to the conjecture, elimination equations have to be selected to try to raise disagreements. Each rule has to be examined in turn. R_1 and R_2 are each linked to a pair of disagreement sets, all of which are made up of terms,

$$
\begin{array}{rl}
\langle x \times (y + z),\ v + 0 \rangle & (R_1) \\
\langle y + z,\ v + 0 \rangle & (R_1) \\
\langle x \times (y + z),\ 0 + v \rangle & (R_2) \\
\langle y + z,\ 0 + v \rangle & (R_2)
\end{array}
$$

where the solution of any of them leads to a solution for the original problem. As heuristic criteria discourage the generation of new rules applicable to terms, the four pairs above are ignored.

For each of the other two remove *rules, whose lhs expressions are formulae, there are also two disagreement sets. R_4, for instance, involves*

$$
\begin{array}{r}
\langle y \times z^2 < x \wedge x \times (y + z) = y \times x,\ v_1 + v_2 = v_1 \rangle \\
\langle x \times (y + z) = y \times x,\ v_1 + v_2 = v_1 \rangle
\end{array}
$$

However, since all equations of $\mathcal{E}_{PA^*}$, listed in appendix H, consist of formulae whose rhs and lhs expressions are both atoms or both terms, the first of the above pairs is unsolvable, for no equality chain can link a composite to an atomic formula if there is no equation where one of the sides is an atom and the other one is a composite formula. Hence, from eight disagreement sets, six have been disregarded: four of them were composed of terms, and two contained composite formulae. Only two then are left,

$$
\begin{array}{rl}
\langle x \times (y + z) = y \times x,\ v_1 + v_2 = 0 \rangle & (R_3) \\
\langle x \times (y + z) = y \times x,\ v_1 + v_2 = v_1 \rangle & (R_4)
\end{array}
$$

A new applicable remove *rule is eventually obtained from R_4.* □

9.4 Termination

The effective enlargement of a subclass requires that the process of removal of deviant symbols halts. Termination has to be ensured both locally, w.r.t. the application of rules for each symbol, and globally, for the elimination of all deviant symbols.

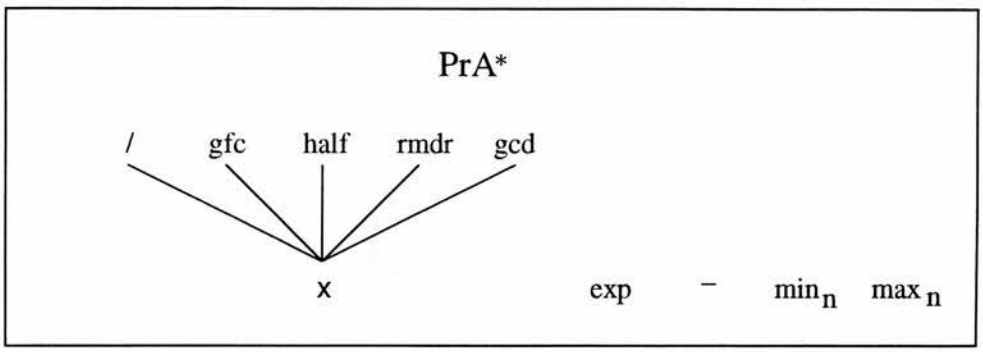
The existence of a well-founded relation for deviant symbols, as described in section 6.2.2, satisfies the requirements from the global point of view. Figure 9.2 presents a well-founded order for deviant symbols with respect to $\mathcal{L}_{PrA^{\bullet}}$ and $\mathcal{L}_{SMA}$. Total orders can be generated once certain restrictions are observed. Unconnected symbols can be removed at any stage of rewriting: this is the case, for instance, of exp and $-$ with respect to $\mathcal{L}_{PrA^{\bullet}}$. Bottom elements of chains, on the other hand, must be placed towards the end of the total order. In the case of $\mathcal{L}_{SMA}$, $<$ can be succeeded only by exp, which is also minimal: for all the other symbols, since each of them has at least one rule containing $<$ in its rhs expression, their removal must antecede $<$. Global termination can then be guaranteed on the following grounds: given a total ordering $\langle S_1, \ldots, S_n \rangle$ for a finite set of deviant symbols,

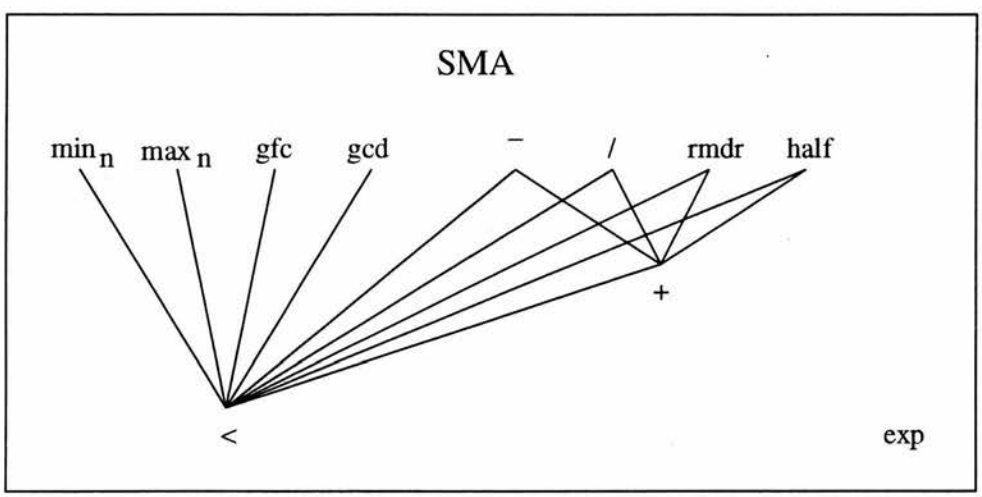
i. If S_i is the first symbol of this ordering that occurs in a conjecture ϕ, *remove* rules for S_i are exhaustively applied to a conjecture ϕ.

ii. If S_i is completely removed, the next symbol in the ordering is selected and the process is repeated from start. When no symbol is left, the process is complete.

iii. If it is not possible to remove all occurrences of S_i from ϕ, the process fails.

Given that no rule for a symbol in this ordering, due to its construction, introduces any previously removed symbol, no deviant symbol is left in the rewritten conjecture after the removal of S_n, in which case an element of the chosen decidable subclass must have been obtained. Local termination, on the other hand, is ensured only after the set of *remove* rules for each deviant symbol is examined.





Intermediate language	$\mathcal{L}_{PA^{\bullet\bullet}}$
Decidable sublanguages	$\mathcal{L}_{PrA^{\bullet}}$ and $\mathcal{L}_{SMA}$
Deviant symbols	$\times$, exp, $-$, half, /, gfc, rmdr, gcd, $\min_n, \max_n$ (w.r.t. $\mathcal{L}_{PrA^{\bullet}}$)
	$+$, exp, $-$, half, /, gfc, rmdr, gcd, $\min_n, \max_n, <$ (w.r.t. $\mathcal{L}_{SMA}$)

Figure 9.2: Well-founded Orders for Deviant Symbols

9.4.1 Chain-reducing Rules

The application of partial dominant rules always halts, given that the measure for total rules of lemma 6.2.1 is also valid in this case: any rule of the form

$$S_i(t_1, \ldots, t_n) \;\Rightarrow\; S_j(u_1, \ldots, u_m)$$

where $S_i \stackrel{s}{\neq} S_j$, reduces the number of occurrences of S_i in a layer of depth d in a conjecture ϕ, and possibly increases the number of such occurrences at lower layers. Since the expansion of the rewritten formula in the course of rewriting is finite, a non-negative decreasing measure can be defined as a result.

This measure, however, does not necessarily decrease for partial non-dominant rules. For instance, when

$$v_1 + v_2 = 1 \times (v_1 + v_3) \;\Rightarrow\; v_2 = v_3 \quad (*)$$

is applied to $x + (y + z) = 1 \times (x + (w + 3))$, which has one occurrence of $+$ at depth 2, the resulting formula, $y + z = w + 3$, has two occurrences of $+$ at this depth. Each group of non-dominant *remove* rules may require therefore a distinct measure. The simplest case involves a reduction in the total number of occurrences of the deviant symbol in the conjecture. For function symbols, this happens only when the number of occurrences of each variable in the rhs expression of a rule is less than or equal to the number of occurrences of the same variable in the lhs expression, as in

$$\begin{aligned} v_1^{v_2} = 0 &\;\Rightarrow\; v_1 = 0 \wedge v_2 \neq 0 \\ v_1 \times v_2 = v_3 \times v_2 &\;\Rightarrow\; v_2 = 0 \vee v_1 = v_3 \end{aligned}$$

or even in $(*)$. For predicate symbols, the number of occurrences always diminishes, provided that the lhs expression of the rule is an atom.

Lemma 9.4.1 *Any finite and normalised set of atomic* remove *rules*[10] *for predicate symbols is noetherian.*

PROOF. Let $\mathcal{R}$ be a normalised set of atomic *remove* rules for predicate symbols $\mathcal{S} = \{p_1, \ldots, p_n\}$. Let $msr(\epsilon)$ be defined as the number of occurrences of $p_1, \ldots, p_n$

[10] A rule is *atomic* iff its lhs expression is an atom.

in an expression ϵ. The application of any rule R of $\mathcal{R}$ to a formula ϕ then reduces $msr(\phi)$, given that R must have the form

$$p_i(t_1, \ldots, t_{l_i}) \quad \Rightarrow \quad p(u_1, \ldots, u_m)$$

such that $p \notin \mathcal{S}$. Hence, since any instance of the rhs expression of a rule of $\mathcal{R}$ has fewer occurrences of symbols of $\mathcal{S}$ than its lhs expression, *msr* is a termination measure for this rewrite set[11]. ∎

The above lemma applies to the set of selected partial rules for $<$, since they are all atomic. For the remaining symbols, however, other results must be sought. Two distinct types of partial rules are examined after the introduction of the notion of *chain reduction.*

Definition 9.4.1 (S-chains)

Let ϵ be an expression of $\mathcal{L}$, and $R.\ \delta_1 \Rightarrow \delta_2$ be a partial remove *rule for a symbol S of $\mathcal{L}$.*

i. A symbol chain $\boldsymbol{S}$ *of ϵ is any finite non-empty sequence of symbols that satisfies one of the conditions.*

(a) $\boldsymbol{S} = \langle S' \rangle$ and S' occurs in ϵ, or

(b) $\boldsymbol{S} = \langle S_1, \ldots, S_n \rangle$, ϵ has a subexpression of the form $S_1(\epsilon_1, \ldots, \epsilon_m)$, and $\langle S_2, \ldots S_n \rangle$ is a symbol chain for ϵ_i, for some $i, 1 \leq i \leq m$.

ii. A S-chain *$\langle S_1, \ldots, S_n \rangle$ for ϵ is a symbol chain for ϵ such that $S_1 \stackrel{s}{=} S_n \stackrel{s}{=} S$. The* length *of a S-chain is the number of occurrences of S in the chain.*

iii. R is chain-reducing *w.r.t. S iff, for every substitution σ for the free variables of δ_1, the longest S-chain of $\sigma\delta_2$ has fewer occurrences of S than the longest S-chain of $\sigma\delta_1$.*

---

[11] For non-atomic rules, occurrences of the deviant predicate symbol may actually increase, as in the case of the (rather artificial) *remove* rule for $\leq$,

$$(v_1 \leq v_2) \supset \phi \quad \Rightarrow \quad (v_1 = v_2 \wedge \phi) \vee (v_1 < v_2 \wedge \phi) \vee v_2 < v_1$$

iv. *R is* chain-preserving *w.r.t. S iff there is a substitution σ for the free variables of δ_1 such that the longest S-chain of $\sigma\delta_2$ has the same number of occurrences of S as the longest S-chain of $\sigma\delta_1$.*

Example 9.4.1

i. *Figure 9.3 shows three examples of S-chains, one with length 2 and two with length 1.*

ii. *The rule*

$$v_1 + v_2 = v_1 + v_3 \quad \Rightarrow \quad v_2 = v_3$$

is chain-reducing w.r.t. + , since, given any substitution $\sigma = \{^{t_1}/_{v_1}, ^{t_2}/_{v_2}, ^{t_3}/_{v_3}\}$, if the longest +-chains of t_1, t_2 and t_3 have respectively lengths n_1, n_2 and n_3, then the longest +-chain of $(t_1 + t_2 = t_1 + t_3)$ has length $max(n_1 + 1, n_2 + 1, n_3 + 1)$, whereas the longest +-chain of $(t_2 = t_3)$ has length $max(n_2, n_3)$.

iii. *The rule*

$$v_1 < 1 + v_2 \quad \Rightarrow \quad (v_1 = v_2 \vee v_1 < v_2)$$

is chain-preserving w.r.t. +, for, given the substitution $\sigma = \{^{1+(x+y)}/_{v_1}, ^{0}/_{v_2}\}$, the longest +-chain of both rhs and lhs expressions of the rule, after the application of σ, has length 2.

□

Chain-reducing rules may be redefined in terms of occurrences of variables in their rhs and lhs expressions.

Lemma 9.4.2 *Let R. $\delta_1(v_1, \ldots, v_n) \Rightarrow \delta_2\{v_1, \ldots, v_n\}$ be a partial* remove *rule for S such that $v_1, \ldots, v_j$, $j \leq n$, are the variables shared by δ_1 and δ_2. R is chain-reducing w.r.t. S iff all the occurrences of $v_1, \ldots, v_j$ in δ_1 lie in the scope of occurrences of S.*

PROOF. If each occurrence of v_i, $1 \leq i \leq j$, in δ_1 takes place in the scope of occurrences of S, then, for any substitution $\sigma = \{^{t_1}/_{v_1}, \ldots, ^{t_n}/_{v_n}\}$, the longest S-chain of

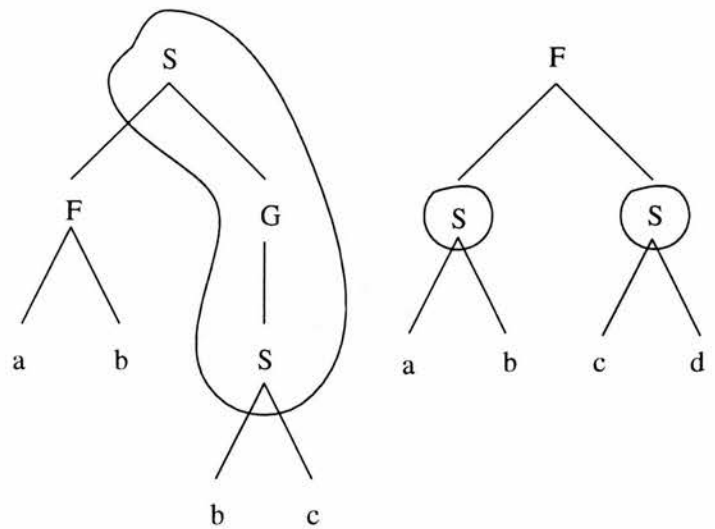


Figure 9.3: Examples of S-chains

$\sigma\delta_1$ has at least length $l+1$, where l is the length of the longest S-chain of $t_1, \ldots, t_j$. Since S does not occur in δ_2, the longest S-chain of $\sigma\delta_2$ has length l.

If there is an occurrence of a variable v_i, $1 \leq i \leq j$, in δ_1 that takes place outwith the scope of an occurrence of S, let $\sigma = \{^{t_1}/_{v_1}, \ldots, ^{t_n}/_{v_n}\}$ be a substitution such that t_i has a S-chain of length l, which is also the length of the longest S-chain in $\sigma\delta_1$. Since by hypothesis v_i also occurs in δ_2, the longest S-chain of both $\sigma\delta_1$ and $\sigma\delta_2$ has length l. Thus R is not chain-reducing. ∎

Clearly, every total and partial dominant *remove* rule is chain-reducing. Termination can be guaranteed for any set of chain-reducing rules for a symbol S.

Lemma 9.4.3 *If $\mathcal{R}$ is a finite set of chain-reducing partial* remove *rules for S, then $\mathcal{R}$ is noetherian.*

PROOF. (By transfinite induction on the length & number of occurrences of S-chains)

$\mathcal{R}$ is noetherian iff, for every formula ϕ of the underlying language, any $\mathcal{R}$-rewriting

sequence for ϕ is finite. Let then the set of formulae of the underlying language be partitioned in subclasses defined by a pair of parameters, $\langle n, m\rangle$, such that

i. n is the length of the longest S-chain of every element of the subclass, and

ii. m is the number of occurrences of S-chains of length n (i.e. the longest S-chain) in every element of the subclass.

Clearly, every formula of the underlying language belongs to one and only one element of the above partition. As a result, $\mathcal{R}$ is noetherian iff, for every $\langle n, m\rangle \in \mathbb{N}^2$, rewriting halts for each formula of the subclass defined by $\langle n, m\rangle$. Since $\langle \mathbb{N}^2, \sqsubseteq\rangle$ is well-ordered, where $\sqsubseteq$ is the lexicographic order for $\mathbb{N}^2$, transfinite induction is applicable.

For any well-ordered set $\langle X, \leq\rangle$, the *principle of transfinite induction* asserts that

$$(u)(u \in X \supset ((v)((v \in X \wedge v < u) \supset \psi(v)) \supset \psi(u)))$$

$$\supset$$

$$(u)(u \in X \supset \psi(u))$$

where ψ is a wf formula of the underlying language[12]. Taking $\langle \mathbb{N}^2, \sqsubseteq\rangle$ as the well-ordered set for the present lemma, the above principle can be instantiated to

$$(\forall\langle n, m\rangle)(\langle n, m\rangle \in \mathbb{N}^2 \supset ((\forall\langle n', m'\rangle)((\langle n', m'\rangle \in \mathbb{N}^2 \wedge \langle n', m'\rangle \sqsubset \langle n, m\rangle) \supset \psi(\langle n', m'\rangle)) \supset \psi(\langle n, m\rangle)))$$

$$\supset$$

$$(\forall\, \langle n, m\rangle)(\langle n, m\rangle \in \mathbb{N}^2 \supset \psi(\langle n, m\rangle))$$

After a few manipulations, the induction hypothesis unfolds into a pair of hypotheses, as schematically indicated below,

$$\begin{array}{rcl} \langle n', m'\rangle \sqsubset \langle n, m\rangle & \supset & \psi(\langle n', m'\rangle) \\ (n' < n \vee (n' = n \wedge m' < m)) & \supset & \psi(\langle n', m'\rangle) \\ (n' < n \supset \psi(\langle n', m'\rangle)) & \wedge & ((n' = n \wedge m' < m) \supset \psi(\langle n', m'\rangle)) \end{array}$$

whereas $\psi(\langle n, m\rangle)$ remains as induction conclusion.

Let the induction hypotheses and conclusion be represented as

[12] See for instance [Mendelson 87], p. 8.

(IH_1) Rewriting halts for every formula where S-chains have length less than n

(IH_2) Rewriting halts for every formula whose longest S-chain has length n and which contains fewer than m occurrences of S-chains of this length

(IC) Rewriting halts for every formula where the longest S-chain has length n and which has m occurrences of S-chains of this length

and let ϕ_0 be an element of the subclass defined by the pair $\langle n, m \rangle$.

Case (1) If no rule in $\mathcal{R}$ is applicable to ϕ_0, then the only possible rewriting sequence for ϕ_0 is empty.

Case (2) If there is a rule in $\mathcal{R}$ which is applicable to ϕ_0, then let

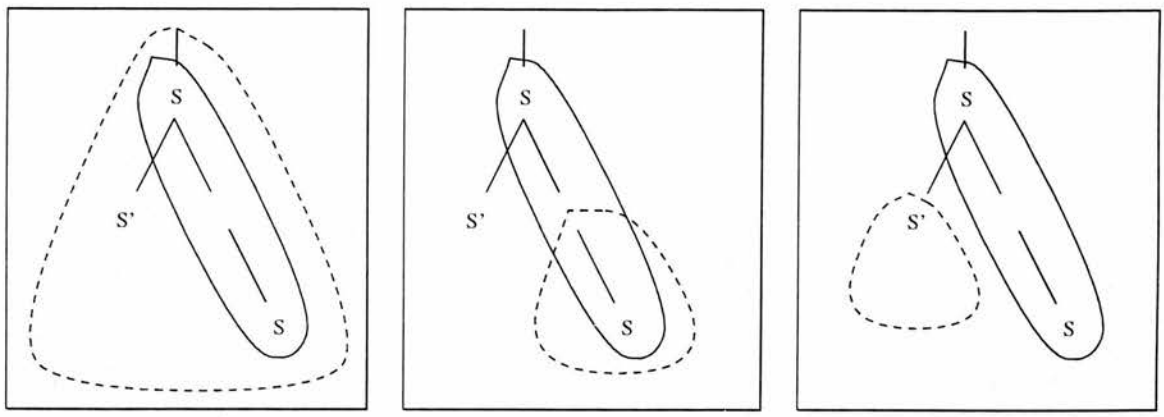
$$\phi_0 \overset{R_1}{\Rightarrow} \phi_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_p}{\Rightarrow} \phi_p \overset{R_{p+1}}{\Rightarrow} \ldots \quad (*)$$

be a rewriting sequence for ϕ_0. As indicated in figure 9.4, three cases must be considered.

i. If there is a rule $R.\ \delta_1 \Rightarrow \delta_2$ which is applicable to a subexpression ϵ of ϕ_0 that has a S-chain of length n, let σ be the substitution such that $\epsilon \overset{s}{=} \sigma\delta_1$. Since each rule of $\mathcal{R}$ is chain-reducing w.r.t. S, the longest S-chain of ϵ is shorter than the longest S-chain of $\sigma\delta_2$. Since ϵ by hypothesis has a S-chain of length n, it must be the longest S-chain of the expression (otherwise n would not be the length of the longest S-chain of ϕ_0). As a result, the longest S-chain of $\sigma\delta_2$ has length less than n and $\phi_0[\![^{\sigma\delta_2}/_{\epsilon}]\!]$ has fewer than m S-chains of length n. Then, according to IH_2, rewriting halts.

ii. If there is a rule $R.\ \delta_1 \Rightarrow \delta_2$ that is applicable to a subexpression ϵ of ϕ_0, such that ϵ contains a proper fragment of a S-chain of length n, then the longest S-chain that occurs in ϵ must have length $n' < n$ (for otherwise ϕ would have a S-chain of length greater than n, as it can be inferred from figure 9.4, case 2 ii). Since R is chain-reducing, and since there is a substitution σ such that $\sigma\delta_1 \overset{s}{=} \epsilon$, then $\sigma\delta_2$ must have S-chains of length less than n'. Hence, $\phi_0[\![^{\sigma\delta_2}/_{\epsilon}]\!]$ has at most $(m-1)$ S-chains of length n. Again, according to IH_2, rewriting halts.

iii. If no rule in the sequence is applied to a subexpression of ϕ_0 containing a fragment (proper or not) of a S-chain of length n, then the sequence must be finite.



Case 2 i Case 2 ii Case 2 iii

$\boxed{S \cdots S}$ S-chain of length n
$- - -$ subexpression ϵ

Figure 9.4: Fragments of S-chains

Otherwise, for every $p \in \mathbb{N}$, there would be a rewriting sequence of the form

$$\phi'_0 \overset{R_1}{\Rightarrow} \phi'_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_p}{\Rightarrow} \phi'_p(**)$$

where ϕ'_i is obtained from ϕ_i through the replacement of the terminal subexpression $S(u_1, \ldots, u_p)$ of every S-chain of length n with an expression ϵ in which S does not occur[13]. However, since by construction ϕ'_0 has only S-chains of length less than n, it follows from (IH_1) that ϕ'_0 has a normal form ϕ'_q, for some $q \in \mathbb{N}$, in contradiction with the assumption that $(**)$ could be arbitrarily enlarged.

Since rewriting halts for every element of the partition, $\mathcal{R}$ is noetherian. ∎

The group of chain-reducing rules is not contained in any of the two classes for which halting measures (the reduction in the number of occurrences of a deviant symbol S, and the function m_t defined in lemma 6.2.1) have been exhibited, i.e.

(i) there is a chain-reducing rule that does not reduce the number of occurrences of

[13] See lemma E.5.1 in appendix E.

the corresponding deviant symbol, and

(ii) there is a chain-reducing rule that does not reduce the number of occurrences of the deviant symbol in the highest layers of the rewritten expression.

An example is given next.

Example 9.4.2 *The partial* remove *rule for* f,

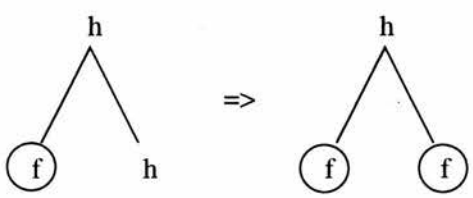
$$R \quad h(f(v_1, v_2), h(v_3, f(v_1, v_3))) \quad \Rightarrow \quad h(v_2, v_2)$$

does not *necessarily reduce the total number of occurrences of* f *in an expression, since, in the sequence*

$$\underline{h(f(a, f(f(a, c), b)), h(e, f(a, e)))} \overset{R}{\Rightarrow} h(f(f(a, c), b), f(f(a, c), b))$$

the initial and final number of occurrences of f *is 4. The measure* m_t *is not necessarily decreased either, considering that, in the sequence*

$$\underline{h(f(a, f(b, b)), h(c, f(a, c)))} \overset{R}{\Rightarrow} h(f(b, b), f(b, b))$$



the number of occurrences of f *in the second layer increases. Nonetheless, according to lemma 9.4.2,* R *is chain reducing and, as a result, its application always halts.* □

9.4.2 Chain-preserving Rules

Lemma 9.4.3 cannot be extended to chain-preserving partial rules, given that some of them generate infinite rewriting sequences.

Example 9.4.3

i. Let $\mathcal{R}$ *be a set of* remove *rules for* g *that contains the rule* $R.\ f(g(v_1), v_2) \Rightarrow f(v_2, v_2)$. $\mathcal{R}$ *then is not noetherian, since*

$$\underline{f(g(a), g(a))} \overset{R}{\Rightarrow} \underline{f(g(a), g(a))} \overset{R}{\Rightarrow} \ldots$$

is infinite.

ii. Given the remove *rules for* f,

$$\begin{array}{lrcl} R_1 & g(f(v_1, v_2)) & \Rightarrow & g(e) \\ R_2 & h(f(v_1, v_2), g(e), v_4) & \Rightarrow & h(v_4, g(v_4), v_4) \end{array}$$

where R_1 *is chain-reducing and* R_2 *is chain-preserving, the set* $\{R_1, R_2\}$ *is not noetherian, given that the sequence*

$$h(f(a,b), \underline{g(f(a,b))}, f(a,b)) \overset{R_1}{\Rightarrow} \underline{h(f(a,b), g(e), f(a,b))} \overset{R_2}{\Rightarrow}$$

$$\overset{R_2}{\Rightarrow} h(f(a,b), \underline{g(f(a,b))}, f(a,b)) \overset{R_1}{\Rightarrow} \ldots$$

is cyclical. □

Under certain circumstances, however, it is possible to guarantee termination for a set $\mathcal{R}$ composed of both chain-reducing and chain-preserving rules. For any fragment of a rewriting sequence of the form

$$\ldots \phi \overset{R}{\Rightarrow} \phi_1 \overset{\widehat{R}_1}{\Rightarrow} \ldots \overset{\widehat{R}_n}{\Rightarrow} \phi_{n+1} \overset{R'}{\Rightarrow} \psi \ldots$$

where R and R' are chain-preserving and $\widehat{R}_1, \ldots, \widehat{R}_n$ are chain-reducing, the number of rewriting steps involving chain-reducing rules must be finite, since any finite (sub)set of such rules is noetherian. As a result, the subsequence can be collapsed into a series of composite rules of the form

$$\ldots \phi \overset{R}{\Rightarrow} \phi_1 \overset{R''}{\Rightarrow} \psi \ldots$$

where R'' is derived from the composition of $\widehat{R}_1, \ldots, \widehat{R}_n$ and R'. Since the resulting composite rule can be either chain-reducing or chain-preserving, the same procedure

can be repeatedly applied until either all chain-preserving or chain-reducing rules are eliminated from the sequence. $\mathcal{R}$ is non-halting only when there is a formula that admits an infinite rewriting sequence entirely composed of (primitive or derived) chain-preserving rules. Therefore, given a set $\mathcal{R}$ containing chain-preserving and chain-reducing *remove* rules, if it is possible to guarantee that

(i) there is no infinite rewriting sequence generated by chain-preserving rules of $\mathcal{R}$, and
(ii) all rules obtained from $\mathcal{R}$ by the composition of chain-reducing and chain-preserving elements are chain-reducing,

then $\mathcal{R}$ is noetherian.

Concerning the rule generation mechanism, provided that the original set $\mathcal{R}$ is noetherian, the same applies to the derived rules. This can be informally justified on the grounds that the transformation that leads to the new rule can be reversed and applied to the conjecture, since disagreements may be dealt with both at the rule and the conjecture levels, and there is a one-to-one correspondence between both processes. Given that rewriting halts at the conjecture level, because the removal of disagreements by RGM involves a finite number of steps and $\mathcal{R}$ is noetherian, the transformation at the rule level must also terminate.

9.5 Limitative Results

Even though the expansion of the rule base $\mathcal{R}_{PA^\bullet}$ is always possible, the undecidability of *PA* sets several constraints for such expansions and virtually excludes certain classes of candidate rules. Similar remarks apply to possible extensions of any arithmetical decidable subclass.

9.5.1 Expansions of $\mathcal{R}_{PA^\bullet}$

Several candidate *remove* rules can be discarded as invalid as a direct consequence of the essential undecidability of *PA*. All three types of rules (total, partial dominant and partial non-dominant) are affected.

Lemma 9.5.1 (Total rules)

There is no valid total remove *rule for* $+, s$ *or* $<$ *w.r.t.* $\mathcal{L}_{SMA}$.

PROOF. Once occurrences of successor are eliminated in favour of $+$, a total *remove* rule for this symbol would eliminate all its occurrences from any formula of $\mathcal{L}_{PA}$, effectively reducing it to an equivalent formula of $\mathcal{L}_{SMA}$, which is decidable, thus contradicting the undecidability of arithmetic. The non-existence of total rules for either s or $<$ follows from results reported in [Robinson 49]. ∎

Lemma 9.5.2 (Partial dominant rules)

i. No valid partial dominant remove *rule for* $+$ *has any of the forms*

$$\begin{array}{lrcl} R_1 & v+v & \Rightarrow & \delta_1\{v\} \\ R_2 & v+1 & \Rightarrow & \delta_2\{v\} \\ R_3 & v_1+(v_2+v_3) & \Rightarrow & \delta_3\{v_1,v_2,v_3\} \\ R_4 & v_1+(v_2\times v_3) & \Rightarrow & \delta_4\{v_1,v_2,v_3\} \end{array}$$

where δ_i *is a term of* $\mathcal{L}_{SMA}$.

ii. No valid partial dominant remove *rule for* $\times$ *has any of the forms*

$$\begin{array}{lrcl} R_5 & v\times v & \Rightarrow & \delta_1\{v\} \\ R_6 & v\times 1 & \Rightarrow & \delta_2\{v\} \\ R_7 & v_1\times(v_2\times v_3) & \Rightarrow & \delta_3\{v_1,v_2,v_3\} \\ R_8 & v_1\times(v_2+v_3) & \Rightarrow & \delta_4\{v_1,v_2,v_3\} \end{array}$$

where δ_i *is a term of* $\mathcal{L}_{PrA}$.

PROOF.

i. The first two rules, if valid, would have to have originated from an equation of one of the forms

$$\begin{array}{rcl} v+v & = & t_1 \\ v+1 & = & t_2 \end{array}$$

where t_1 and t_2 are terms of $\mathcal{L}_{SMA}$. Since terms of $\mathcal{L}_{SMA}$ can be effectively put in one of the following normal forms

$$0,\ 1,\ v^m \times v_1^{m_1} \times \cdots \times v_n^{m_n}$$

the above equations are reducible to

$$\begin{array}{lcllcl} v+v & = & 0 & \qquad v+1 & = & 0 \\ v+v & = & 1 & \qquad v+1 & = & 1 \\ v+v & = & v^m \times v_1^{m_1} \times \cdots \times v_n^{m_n} & \qquad v+1 & = & v^m \times v_1^{m_1} \times \cdots \times v_n^{m_n} \end{array}$$

which are PA-invalid, for all $n, m, m_1, \ldots, m_n \in \mathbb{N}$.

Concerning R_3 and R_4, it has to be taken into account that any term of $\mathcal{L}_{PA}$ in which $+$ occurs can be effectively converted into expressions of the forms $t_1 + (t_2 + t_3)$ and $t_1 + (t_2 \times t_3)$. A procedure for the first pattern has five steps,

(a) Occurrences of successor are totally eliminated by the rule $\mathrm{s}(v) \Rightarrow v + 1$.

(b) Stratification rules,

$$\begin{array}{rcl} v_1 \times (v_2 + v_3) & \Rightarrow & v_1 \times v_2 + v_1 \times v_3 \\ (v_1 + v_2) \times v_3 & \Rightarrow & v_1 \times v_3 + v_2 \times v_3 \end{array}$$

transform the term until it assumes the form $t_1 + t_2$.

(c) If $+$ occurs in t_2, the same stratification rules are applied to it, and an expression of the desired form, $t_1 + (t_3 + t_4)$, results.

(d) If $+$ occurs only in t_1, it is first reduced to $(t_3 + t_4)$, then the original term is put into the desired form by the application of $(v_1 + v_2) + v_3 \Rightarrow v_1 + (v_2 + v_3)$.

(e) If $+$ occurs neither in t_1 nor in t_2, t_2 is replaced with $t_2 + 0$.

A procedure for the second pattern adopts steps (a) and (b), plus an additional one,

(c') If $\times$ is not the dominant symbol of t_2, it is replaced with $t_2 \times 1$.

Hence any term could be effectively stripped of all the occurrences of $+$ by the exhaustive application of either R_3 or R_4, interleaved with calls to the above procedures. Since both rules are chain-reducing and the procedures always terminate, any formula of $\mathcal{L}_{PA}$ would be reducible to $\mathcal{L}_{SMA}$.

ii. Similar to the above proof. ∎

Lemma 9.5.3 (Partial non-dominant rules)

i. No valid remove *rule for* $+$ *has any of the forms*

$$\begin{array}{rcl} v_1 + v_2 = v_3 & \Rightarrow & \phi_1\{v_1, v_2, v_3\} \\ v_1 + v_2 = v_3 + v_4 & \Rightarrow & \phi_2\{v_1, v_2, v_3, v_4\} \\ v_1 + v_2 = v_3 \times v_4 & \Rightarrow & \phi_3\{v_1, v_2, v_3, v_4\} \\ v_1 \times v_2 + v_3 = v_4 & \Rightarrow & \phi_4\{v_1, v_2, v_3, v_4\} \end{array}$$

where each ϕ_i *is a formula of* $\mathcal{L}_{SMA}$.

ii. No valid remove *rule for* $\times$ *has any of the forms*

$$\begin{array}{rcl} v_1 \times v_2 = v_3 & \Rightarrow & \phi_5\{v_1, v_2, v_3\} \\ v_1 \times v_2 = v_3 \times v_4 & \Rightarrow & \phi_6\{v_1, v_2, v_3, v_4\} \\ v_1 \times v_2 = v_3 + v_4 & \Rightarrow & \phi_7\{v_1, v_2, v_3, v_4\} \\ v_1 \times v_2 + v_3 = v_4 & \Rightarrow & \phi_8\{v_1, v_2, v_3, v_4\} \end{array}$$

where each ϕ_i *is a formula of* $\mathcal{L}_{PrA}$.

iii. There is no remove *rule for* s *of the form*

$$v_1 \times v_2 = s(v_3) \quad \Rightarrow \quad \phi\{v_1, v_2, v_3\}$$

where ϕ *is a formula of* $\mathcal{L}_{SMA}$. *Also, there is no* remove *rule for* $\times$ *of the above form, where* ϕ *belongs to* $\mathcal{L}_{PrA}$.

PROOF.

i. The validity in *PA* of $(v_1 + v_2 = v_3) \equiv \phi_1\{v_1, v_2, v_3\}$ would mean that a *remove* rule scheme for $+$,

$$\psi[\![v_1 + v_2]\!] \quad \Rightarrow \quad (\exists v_3)(\psi[\![^{v_3}/(v_1 + v_2)]\!] \wedge \phi_1\{v_1, v_2, v_3\})$$

could be defined, thus contradicting the undecidability of *PA*. A similar argument applies to the other three cases, considering that

$$\begin{array}{rcccl} v_1 + v_2 = v_3 & \equiv & v_1 + v_2 = v_3 + 0 & \equiv & \phi_2\{v_1, v_2, v_3, {}^0/_{v_4}\} \\ v_1 + v_2 = v_3 & \equiv & v_1 + v_2 = v_3 \times 1 & \equiv & \phi_3\{v_1, v_2, v_3, {}^1/_{v_4}\} \\ v_1 + v_3 = v_4 & \equiv & (v_1 \times 1) + v_3 = v_4 & \equiv & \phi_4\{v_1, {}^1/_{v_2}, v_3, v_4\} \end{array}$$

and, again, a complete set of *remove* rules for $+$ in terms of $\mathcal{L}_{SMA}$ could be obtained.

ii. If $(v_1 \times v_2 = v_3) \equiv \phi_5\{v_1, v_2, v_3\}$ is valid in PA, then

$$\psi[\![v_1 \times v_2]\!] \;\Rightarrow\; (\exists v_3)(\psi[\![^{v_3}/_{(v_1 \times v_2)}]\!] \wedge \phi_5\{v_1, v_2, v_3\})$$

represents a *remove* rule scheme for $\times$ in terms of $\mathcal{L}_{PrA}$, also contradicting the undecidability of PA. A similar argument applies to the other three cases, considering that

$$\begin{array}{lcll} v_1 \times v_2 = v_3 & \equiv & v_1 \times v_2 = v_3 \times 1 & \equiv \quad \phi_6\{v_1, v_2, v_3, {}^1/_{v_4}\} \\ v_1 \times v_2 = v_3 & \equiv & v_1 \times v_2 = v_3 + 0 & \equiv \quad \phi_7\{v_1, v_2, v_3, {}^0/_{v_4}\} \\ v_1 \times v_2 = v_4 & \equiv & (v_1 \times v_2) + 0 = v_4 & \equiv \quad \phi_8\{v_1, v_2, {}^0/_{v_3}, v_4\} \end{array}$$

iii. Assume

$$v_1 \times v_2 = \mathrm{s}(v_3) \quad \equiv \quad \phi\{v_1, v_2, v_3\} \quad (*)$$

is valid in PA. If $\phi\{v_1, v_2, v_3\}$ is a formula of $\mathcal{L}_{SMA}$, considering that $v \times 1 = v$, then, from a rewritten instance of $(*)$,

$$v_1 = \mathrm{s}(v_3) \quad \equiv \quad \phi\{v_1, {}^1/_{v_2}, v_3\}$$

a complete set of *remove* rules for s w.r.t. $\mathcal{L}_{SMA}$ could be obtained.

If $\phi\{v_1, v_2, v_3\}$ belonged to $\mathcal{L}_{PrA}$ instead, since

$$\begin{array}{rcl} v_1 \times v_2 = v_4 & \equiv & (v_4 = 0 \vee v_4 \neq 0) \wedge (v_1 \times v_2 = v_4) \\ & \equiv & (v_4 = 0 \wedge v_1 \times v_2 = 0) \vee (\exists v_3)(v_4 = \mathrm{s}(v_3) \wedge v_1 \times v_2 = \mathrm{s}(v_3)) \\ & \equiv & (v_4 = 0 \wedge (v_1 = 0 \vee v_2 = 0)) \vee (\exists v_3)(v_4 = \mathrm{s}(v_3) \wedge \phi\{v_1, v_2, v_3\}) \end{array}$$

there would then be a $\mathcal{L}_{PrA}$-formula equivalent to $v_1 \times v_2 = v_4$, from which a complete set of *remove* rules for $\times$ in terms of $\mathcal{L}_{PrA}$ could be derived. ∎

None of these negative results, however, excludes the existence of other partial non-dominant rules for either sum or multiplication.

9.5.2 Extended Classes

The recursive axiomatisability of PA ensures that any of its decidable subclasses is expandable. The undecidability of PA, on the other hand, prevents any recursive

extension of such classes from embracing $\mathcal{L}_{PA}$. Another limitation is the absence of undecidable formulae in extended classes of type II.[14]

Since both PrA^* and SMA are negation complete, and since the generation of expansions of $\mathcal{L}_{PrA^\bullet}$ and $\mathcal{L}_{SMA}$ by means of proof plans is currently based on rules originated from equations and equivalences, no arithmetical undecidable formula can be incorporated in the enlarged subclasses.

9.6 Recognising the Elements of the Extended Class

Theorem proving for recursively axiomatisable and undecidable theories has to resort to heuristically guided semidecision procedures whenever a conjecture lies outside the identified decidable domain. For this reason, the complexity of the process of computing the membership relation for decidable subclasses affects the overall efficiency of hybrid systems. For the extended class $\Sigma_{PA^\bullet}$ generated by $\mathcal{R}_{PA^\bullet}$, there is no mechanism that establishes whether a formula ϕ belongs to it other than the application of rewrite rules controlled by proof plans (or any equivalent procedure), until an element of a decidable sublanguage $\mathcal{L}'$ is obtained, i.e.

$$\phi \in \Sigma_{PA^\bullet} \quad \text{iff} \quad (\exists \psi)(\phi \overset{\mathcal{R}_{PA^\bullet}}{\Longrightarrow} \psi \;\&\; \psi \in Fml_{\mathcal{L}'})$$

In the event the reduction process fails, the conjecture has then to be supplied to a semidecision procedure. In such cases, if a proof is found, the total time for its computation has to include the failed attempt involving the decision procedure. Hence, since rewriting is in some cases very time consuming, particularly in the presence of difference reduction procedures, the time performance of the integrated system suffers as a result.

Ideally, given any formula of the underlying language, it should be possible to determine whether it belongs to a decidable domain without any rewritten formula being

[14] A proof for these two limitations can be found in appendix E. It is worth noting that they are not equivalent. On one hand, a theory is undecidable iff none of its decidable classes can be extended to the full set of formulae of the underlying language. On the other hand, the impossibility of introducing undecidable formulae in a negation-complete decidable subclass as it is expanded by the inclusion of (T)-equivalent formulae is not restricted to undecidable theories. This is due to the fact that there are decidable consistent theories for which the set of undecidable sentences is not empty (i.e. the theory is not negation complete). For such theories, lemma E.5.2 applies as well.

generated as a side effect, in which case rewriting could be confined to the decision process. Sublanguages, for instance, are classes where the computation of the membership relation involves simply the search for deviant symbols in a conjecture ϕ: if $dev\_sym(\epsilon, \mathcal{L}')$ denotes the set of deviant symbols of an expression ϵ with respect to a decidable sublanguage $\mathcal{L}'$, then

$$\phi \in Fml_{\mathcal{L}'} \quad \text{iff} \quad \phi \in Fml_{\mathcal{L}} \;\&\; dev\_sym(\phi, \mathcal{L}') = \emptyset$$

For other context-free generated classes, elements may be also identified based on the strict inspection of syntactic properties of formulae or terms. For decidable classes extended by the application of rewrite rules, rewriting may be avoided once certain properties of the rule base are taken into account.

Example 9.6.1 *Let Σ be the class of terms of $\mathcal{L}_{PA^*}$ defined by the productions*

$$\begin{array}{rcl} tm & := & attm \mid double(tm) \\ attm & := & 0 \mid 1 \mid var \\ var & := & x \mid y \mid \ldots \end{array}$$

and let Σ' be the extended class generated from Σ by the application of the rule

$$R. \quad v + v \;\Rightarrow\; double(v)$$

which can be represented as

$$\Sigma' \;=\; \left\{ t \in Trm_{\mathcal{L}_{PA^*}} \mid t \overset{\{R\}}{\Rightarrow} t' \;\&\; t' \in \Sigma \right\}$$

Given that Σ' is recursively definable as

i. $\Sigma \subset \Sigma'$

ii. If $t \in \Sigma'$, then $double(t) \in \Sigma'$ and $(t + t) \in \Sigma'$

it is then possible to determine whether a term t of $\mathcal{L}_{PA^*}$ belongs to Σ' by means of a syntactic test that checks whether the symbols that occur in t are only variables, $0, 1, +$ and double, and that, for each occurrence of $+$, both left and right subterms are syntactically identical. □

Another solution is the identification of proper subsets of the extended class for which a definition based on syntactic properties of the formulae can be given. There are some simple examples involving $\mathcal{R}_{PA^*}$.

Example 9.6.2 *Let*

$$\Sigma = \left\{ \phi \in Atm_{\mathcal{L}_{PA}} \mid (\exists \psi_1)(\exists \psi_2) \left(\phi \overset{R_1}{\Rightarrow} \psi_1 \overset{\{R_2,R_3\}}{\Longrightarrow} \psi_2 \;\&\; \psi_2 \in Fml_{\mathcal{L}_{SMA}} \right) \right\}$$

be a subset of the decidable class generated by $\mathcal{R}_{PA^*}$, such that

$$\begin{array}{lll} R_1 & v_1 \times v_2 = v_1 & \Rightarrow \quad v_1 = 0 \vee v_2 = 1 \\ R_2 & v_1 + v_2 = 0 & \Rightarrow \quad v_1 = 0 \wedge v_2 = 0 \\ R_3 & v_1 + v_2 = 1 & \Rightarrow \quad (v_1 = 1 \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = 1) \end{array}$$

Since R_1 is applicable to every atom ϕ of Σ, ϕ must have the form

$$t_1 \times t_2 = t_1$$

where t_1 and t_2 are terms of $\mathcal{L}_{PA}$. After the application of R_1, ϕ is transformed into

$$t_1 = 0 \vee t_2 = 1$$

By induction on the length of the rewriting sequence, it can be proved that t_1 and t_2 are elements of the class Θ, defined by the productions[15]

$$\begin{array}{rcl} pol & := & sum | pol + pol \\ sum & := & 0 | 1 | var | sum \times sum \\ var & := & x_1 | y_1 | z_1 | x_2 | \ldots \end{array}$$

Therefore, every element of Σ has the form

$$P_1(v_1, \ldots, v_{n_1}) \times P_2(v_1, \ldots, v_{n_2}) = P_1(v_1, \ldots, v_{n_1})$$

where $P_i(v_1, \ldots, v_{n_i})$ is a polynomial of Θ. No rewritten formula is derived in this process[16]. □

[15] See appendix E.

[16] A similar approach has been adopted by Armando and Giunchiglia in their study of extensions of the UE-form subclass. As described in [Armando & Giunchiglia 93], p. 498-9, certain syntactically defined subsets of the extended class have been revealed after the formal analysis of the set of rules.

The direct computation of the membership relation may be avoided once equivalent but possibly simpler problems are identified. There are cases, for instance, in which a formula ϕ belongs to an extended class if and only if canonical versions of ϕ share the same property. For $\mathcal{R}_{PA^*}$, since every (primitive or derived) *remove* rule available to the system is applicable either to a term or an atomic formula, then

$$\phi \in \Sigma_{PA^*} \quad \text{iff} \quad \phi^D \in \Sigma_{PA^*} \quad \text{iff} \quad \phi^D \in \Sigma^D_{PA^*} \quad (\dagger)$$

where ϕ^D is the prenex disjunctive normal form (PDNF) of ϕ, and $\Sigma^D_{PA^*}$ is the intersection of Σ_{PA^*} with the PDNF class of $\mathcal{L}_{PA^*}$. The domain of the original relation can therefore be limited to PDNF formulae[17]. No similar result applies to the atom level though. Due to the incompleteness of the rule generation mechanism and the limited deductive power of $\mathcal{R}_{PA^*}$, an atom may not be reducible into a decidable sublanguage even when its canonical form (or other *PA*-equivalent atoms) is. For instance, the conjecture

$$x^2 \times x^2 \;+\; (x^2 \times y + x^2 \times y) \;=\; x^2$$

is equivalent (in *PA*) to other atoms, e.g.

$$\begin{array}{rcrcrcll}
x^2 \times x^2 & + & (y \times x^2 & + & y \times x^2) & = & x^2 & \text{(i)} \\
x^4 & + & (y \times x^2 & + & y \times x^2) & = & x^2 & \text{(ii)} \\
x \times (x^2 \times x) & + & (x^2 \times y & + & x^2 \times y) & = & x^2 & \text{(iii)} \\
(x^2 \times x) \times x & + & (y \times x^2 & + & y \times x^2) & = & x^2 & \text{(iv)} \\
x^2 \times x^2 & + & ((x \times y) \times x & + & x^2 \times y) & = & x^2 & \text{(v)}
\end{array}$$

Nevertheless, only the original atom and formulae (i) - (iii) are reducible to $\mathcal{L}_{PrA}$ by the controlled application of $\mathcal{R}_{PA^*}$, on the assumption that $\mathcal{E}_{PA^*}$ is the set of equations available for disagreement elimination. The assessment of the boundaries

[17] See lemma E.5.4 in appendix E. When the rules are not conditional, each atom of ϕ can be dealt with as an independent expression, since

$$\phi \in \Sigma \quad \text{iff} \quad \bigwedge_{i=1}^{n} (\phi_i \in \Sigma)$$

where $\phi_1, \ldots, \phi_n$ are all the distinct atomic formulae that occur in ϕ.

of the enlarged decidable sublanguage cannot therefore be limited to the study of the class of formulae containing atoms in e.g. polynomial form[18].

Finally, once a partition $\wp = \{\boldsymbol{P}_1, \ldots, \boldsymbol{P}_n\}$ for the universe set is given, the recognition of members of an extended class Σ can be decomposed into a series of subtasks,

$$\phi \in \Sigma \quad \text{iff} \quad (\phi \in \Sigma \cap \boldsymbol{P}_1) \vee \ldots \vee (\phi \in \Sigma \cap \boldsymbol{P}_n)$$

A possible partition for the PDNF class of $\mathcal{L}_{PA}$ has two elements, Ω_1, made up of formulae whose disjuncts contain just a single literal,

$$(Q_1 v_1) \ldots (Q_n v_n)(\ell_1 \vee \cdots \vee \ell_m)$$

where ℓ_i is a literal, and the complementary class, Ω_2. Various strategies may be conceived for the approximate and empirical characterisation of each subclass $\Sigma_{PA^*} \cap \Omega_i$. The use of representative formulae of each element of the partition and of randomly generated formulae is described in the next chapter.

9.7 Conclusions

Two groups of general-purpose proof plans, made up respectively of deciders and simplifiers, have been implemented to control the application of rewrite rules in the extension of decidable sublanguages. The simplifiers mimic some of the features of other systems that contain decision procedures, such as the *Stanford Pascal Verifier* and *Nqthm*. Since these plans are applicable to any theory that admits decidable sublanguages, they are suitable for Peano arithmetic and its extensions.

Extensions of *PA* obtained by the introduction of recursive functions and relations are such that complete sets of *remove* rules can be provided for all the new symbols. Complexity considerations, however, discourage the use of such rules whenever they either introduce new occurrences of quantifiers and/or deviant symbols, or substantially increase the size of a conjecture. In such cases, partial rules become an additional asset for systems that convert formulae of the expanded language into a decidable domain.

[18] Moreover, even when an atom and its canonical (polynomial) normal form are both reducible to a decidable subclass by means of $\mathcal{R}_{PA^*}$, time performance may vary significantly in each case.

The effectiveness of rewriting requires $\mathcal{R}_{PA^\bullet}$ to be noetherian. Termination for this rule set is guaranteed by the existence of a total ordering for deviant symbols, based on a well-founded relation, and the noetherianity of certain subsets of partial *remove* rules. The essential undecidability of *PA* rules out the introduction of total rules for some symbols and affects the set of partial rules as well, particularly those matchable to terms or atoms. It also prevents any extension of a decidable sublanguage from embracing $\mathcal{L}_{PA}$. The boundaries of certain subsets of extended classes, on the other hand, can be assessed by mechanisms which do not require the direct application of rewrite rules.

An experiment involving arithmetical conjectures has been conducted to evaluate the role of $\mathcal{R}_{PA^\bullet}$ in extending both $\mathcal{L}_{PrA^\bullet}$ and $\mathcal{L}_{SMA}$, as well as the efficiency of plans such as *decide1*/1 and *simplify*/1. Results are reported in the next chapter.

Chapter 10

Empirical Results

Four groups of formulae have been supplied to the general-purpose plans described in chapter 9 to assess their complexity and effect on the extension of decidable sublanguages. The first group, examined in sections 10.1 and 10.2, shows that the control mechanisms embedded in these plans have a positive impact on efficiency, at least for a subdomain of $\mathcal{L}_{PA^*}$. The second group, described in section 10.3, is made up of arithmetical lemmas related to a set of representative verification conditions. A set of randomly generated formulae then provides a statistical mapping for the extended decidable classes, as described in section 10.4. The final group, composed of quantifier-free conjectures, has been checked by *Nqthm*'s simplifier and by the the plans *simplify*/1 and *weak_simplify*/1, as described in section 10.5.

10.1 Controlled Rewriters

The six rewriters defined in the previous chapter are ordered according to the scope of their control structures, which impose restrictions upon the choice of decidable classes, disagreement pairs or *remove* rules.

i. *rewrite1*/4 exhaustively applies all the *remove* rules available to the system, until the input expression is reduced to one of the decidable sublanguages, whenever possible.

ii. *rewrite2*/5 is an extension of *rewrite1*/4 that includes RGM (the *rule generation mechanism*) to raise disagreements.

iii. *rewrite3*/5 selects a decidable sublanguage before starting to apply rules, and, once it is selected, only *remove* rules for its deviant symbols are tested.

iv. *rewrite4*/5 additionally restricts the application of *remove* rules to deviant subformulae, i.e. those containing deviant symbols.

v. *rewrite5*/5 heuristically orders decidable sublanguages before rewriting starts.

vi. *rewrite6*/5 heuristically orders the symbols that are deviant w.r.t. the selected decidable sublanguage.

The plan *decide1*/1 is placed at the end of this order, since, besides all the control elements of its predecessors, it also heuristically organises rules. RGM is interfaced to all but the first system. Further reductions in the search space for disagreement elimination have been obtained after both non-matchable *remove* rules applicable to terms and disagreement pairs containing composite formulae have been discarded[1].

Even though the control structures would be intuitively expected to have a cumulative positive effect on the performance of the rewriters, that is not always the case, as shown by the experiments conducted with a set of arithmetical formulae[2]. One of them is derived from a verification condition proposed by Boyer and Moore, whose reduction to a decidable sublanguage by *decide1*/1 is detailed below.

Example 10.1.1 *Let ϕ be the formula*

$$x \neq 0 \supset x^2 \times x^2 + (x^2 \times y + x^2 \times y) \leq x^2 \qquad (*)$$

Once it is supplied to decide1/1, *a decidable sublanguage is first chosen. With respect to $\mathcal{L}_{PrA^*}$, ϕ has nine occurrences of deviant symbols (eight of $\times$ and one of $\leq$) and only three with respect to $\mathcal{L}_{SMA}$ (two occurrences of $+$ and one of $\leq$). Given the measures for these symbols,*

$$\begin{aligned} m_d(+, \mathcal{L}_{SMA}) &= 2.44 \\ m_d(\times, \mathcal{L}_{PrA^*}) &= 2.33 \\ m_d(\leq, \mathcal{L}_{SMA}) &= 3.00 \\ m_d(\leq, \mathcal{L}_{PrA^*}) &= 1.00 \end{aligned}$$

[1] The effect of these restrictions is discussed in section 9.3.3.

[2] The set of conjectures is listed in table G.1.

and the measures for the sublanguages,

$$\begin{aligned} m_c(\phi, \mathcal{L}_{PrA^\bullet}) &= 8 \times m_d(\times, \mathcal{L}_{PrA^\bullet}) + m_d(\leq, \mathcal{L}_{PrA^\bullet}) \\ m_c(\phi, \mathcal{L}_{PrA^\bullet}) &= 19.64 \end{aligned}$$

$$\begin{aligned} m_c(\phi, \mathcal{L}_{SMA}) &= 2 \times m_d(+, \mathcal{L}_{SMA}) + m_d(\leq, \mathcal{L}_{SMA}) \\ m_c(\phi, \mathcal{L}_{SMA}) &= 7.88 \end{aligned}$$

twenty rewriting steps are therefore expected to take place in the reduction of ϕ to $\mathcal{L}_{PrA^\bullet}$, if it is possible, whilst eight steps would suffice for $\mathcal{L}_{SMA}$, which is then chosen. Given that $\leq$ is totally removable and has only a single remove *rule in the rule base, $\langle \leq, + \rangle$ is the only total order for the set of deviant symbols that has to be considered. After the removal of $\leq$, given the rewritten formula, ϕ',*

$$x \neq 0 \supset (x^2 \times x^2 + (x^2 \times y + x^2 \times y) = x^2 \vee x^2 \times x^2 + (x^2 \times y + x^2 \times y) < x^2) \qquad (\dagger)$$

since none of the nine remove *rules for $+$ in $\mathcal{R}_{PA^\bullet}$ is applicable to ϕ', the rule generation mechanism is invoked. Only seven rules are left though, because* remove *rules for terms are not currently admitted by this mechanism. For the first disjunct of ϕ', the first choice is*

$$v_1 + v_2 = v_1 \Rightarrow v_2 = 0$$

given that the disagreement pair formed by an instance of its lhs expression[3]*, $x \times x + v_2 = x \times x$, and the first disjunct of $(\dagger)$ has the lowest disagreement depth,*

$\boxed{x}$	$\times$	$\boxed{x}$	$+$	v_2	$=$	$x \times x$
$(x \boxed{\times} x)$	$\times$	$(x \boxed{\times} x)$	$+$	$(x^2 \times y + x^2 \times y)$	$=$	$x \times x$

[3] A variable in a rule has to be instantiated in the course of disagreement elimination only when it has multiple occurrences, and each occurrence is supposed to match syntactically distinct subexpressions of the conjecture. For the above example, there are two candidates for the replacement of v_2, x^2 and $x^2 \times x^2$, the first of which has been chosen. This guideline has already been examined in chapter 8.

After failed attempts to erase the disagreements of this pair, the next choice involves two rules with equal disagreement depths,

$$\begin{array}{lcl} v_1 + v_2 = 1 & \Rightarrow & (v_1 = 1 \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = 1) \\ v_1 + v_2 = 0 & \Rightarrow & (v_1 = 0 \wedge v_2 = 0) \end{array}$$

For the first rule, the first disagreement, which occurs in the second layer,

v_1	$+$	v_2	$=$	$\boxed{1}$
$(x^2 \times x^2)$	$+$	$(x^2 \times y + x^2 \times y)$	$=$	$x\boxed{\times}x$

is eliminable, and a suitable applicable new rule is then generated, as described in section 8.1.1. The same procedure is repeated until all the occurrences of $+$ *are eliminated, and* (†) *is successfully rewritten to* ϕ''*,*

$$\begin{array}{l} x \neq 0 \supset ((x^2 = 1 \wedge y = 0 \wedge y = 0) \vee \\ \vee (x^2 = 0 \wedge ((y = 1 \wedge y = 0) \vee (y = 0 \wedge y = 1)) \vee (x^2 = 0 \wedge y = 0 \wedge y = 0))) \end{array} \qquad (\ddagger)$$

which belongs to the chosen decidable sublanguage. □

Two variants of ϕ,

$$x^4 + (x^2 \times y + x^2 \times y) = x^2$$

$$x \times (x^2 \times x) + (x^2 \times y + x^2 \times y) = x^2$$

have also been successfully reduced to $\mathcal{L}_{SMA}$ by *decide1*/1. Experimental results[4] involving these and other conjectures are listed in table G.2. Table G.3 and figure 10.1 show that *decide1*/1 required on the average less than one third of the time consumed by *rewrite5*/5, the second fastest of the systems interfaced to RGM. Moreover,

- The performance of *decide1/1* was superior to the performances of the rewriters for 57% of the conjectures that required the intervention of RGM. Even when the full sample is taken into account, *decide1*/1 is still the system that has the highest joint rate of best or second best performances (76%)[5].

[4] Experiments have been conducted on a SunOS 4.1.2 workstation, with the exception of those involving the verification conditions of section 10.3, which were run on a Solbourne 6/702 (100 mips/cpu at 33MHz). All the times indicated have been obtained as a result of a single run.

[5] The conjectures of table G.1 for which *decide1*/1 consumed more time than at least one of the rewriters are ϕ_{101}, ϕ_{104}, ϕ_{105}, ϕ_{107}, ϕ_{108}, ϕ_{110} to ϕ_{113}, ϕ_{115}, ϕ_{120}, ϕ_{121}, ϕ_{123}, ϕ_{125}, ϕ_{127}, ϕ_{202}, ϕ_{204}, ϕ_{302} and ϕ_{303}.

* The gradual introduction of new control features, starting from *rewrite2*/5 till the decider is reached, has in the end a positive effect on time performance for 50% to 80% of those conjectures whose reduction into a decidable sublanguage has been accomplished, as shown in table G.4. When the conjectures solved by *rewrite1*/5 (without the help of RGM, therefore) are excluded, these numbers raise to 64% and 100%. Significative improvements have been also observed in the transitions from *rewrite2*/5 to *rewrite5*/5, *rewrite2*/5 to *rewrite6*/5 and *rewrite3*/5 to *rewrite4*/5.

* *Rewrite4*/5 in particular outperformed *rewrite3*/5 in almost all examples, with the exception of ϕ_{106}, ϕ_{205} and ϕ_{301}. These cases are accounted for by the fact that *rewrite3*/5 allows semantic matching to take place between *remove* rules and subexpressions of the conjecture that do not contain the corresponding deviant symbols, whereas *rewrite4*/5 does not allow such attempts. Although they will usually lead to failure, thus increasing the time required for a successful reduction, there are exceptions where a shorter rewriting path is unveiled, as in the case of the above three conjectures[6].

* *Rewrite5*/5 and *rewrite6*/5 had similar performances, when it comes to the time consumed for the average conjecture, and can be jointly regarded as the most efficient systems after the decider. *Decide1*/5 outperformed both rewriters in all but one of the cases where *rewrite1*/4 failed (ϕ_{111}). Both *rewrite5*/5 and *rewrite6*/5 showed a better performance than *rewrite2*/5 on 70% of the sample. They also led to a cut of at least 78% of the average running time consumed by *rewrite3*/5 and *rewrite4*/5.

Since the control structures are mainly heuristic, their effect on efficiency is not uniform over the domain of application of the plans. *Decide1*/1 has a poor performance in some of the cases where the set of primitive rules alone can reduce the conjecture to a decidable subclass. The heuristic measures it has to determine before rewriting

[6] Previous experiments showed that *rewrite3*/5 fails to reduce ϕ_{203} and ϕ_{301} when the order in which the deviant symbols are stored by the system is changed. Unlike *rewrite4*/5, *rewrite3*/5 does not examine every permissible total order for deviant symbols and, for these particular conjectures, if $\leq$ is not removed in the first place, no other deviant symbol can be eliminated.

starts can consume more time than the exhaustive application of primitive rules, as is probably the case in conjectures ϕ_{108} and ϕ_{113}. Also,

* *Decide1*/1 performs even more poorly w.r.t. *rewrite1*/4 when the guideline that only *remove* rules for deviant symbols should be applied precludes the discovery of shorter rewriting paths. *Rewrite1*/4, on the other hand, exhaustively applies all *remove* rules of $\mathcal{R}_{PA^*}$, independently of the corresponding symbols being deviant or not, and may benefit from such lack of constraints. This is the case for conjectures ϕ_{101}, ϕ_{204} and ϕ_{302}.

* *Rewrite2*/5 exhibited a better performance than *rewrite3*/5 and *rewrite4*/5 in ten cases (27.8% of the formulae where the reduction was successful), which suggests that the non-heuristic selection of a decidable subclass without additional control strategies can cause a considerable deviation from the shortest rewriting path. In the worst case examined, ϕ_{203}, the quotient between the times consumed respectively by *rewrite3*/5 and *rewrite2*/5 was 73.5.

* Amongst those problems where *rewrite1*/4 failed but the reduction was possible, *rewrite4*/5 exhibited a better performance than *decide1*/1 in 35.7% of the cases. This can be explained by the fact that *decide1*/1 may choose an unreachable decidable subclass in the first place, thus delaying the conclusion of rewriting. Such examples are inevitable, since the choice of decidable sublanguages is heuristic.

* When considered individually, some heuristic control features do not have any significant effect on efficiency for certain contexts. For instance, when ordering mechanisms for both decidable classes and deviant symbols are added to *rewrite3*/5, the performance improves in 38.9% of cases, but worsens for the same percentage of the sample. The improvement therefore is entirely cancelled out if mean time is not taken into account.

Another evidence about the damaging effect that narrow control structures may have on performance is illustrated in the cases of *rewrite3*/5 and *rewrite4*/5. For both systems, rewriting is oriented towards a decidable class, but classes are not heuristically selected, i.e. they are always tested according to a pre-established order. As a result,

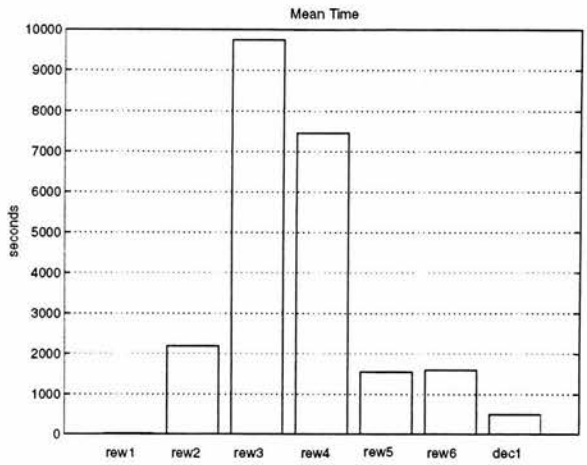
there were examples where only more elaborate rewriters have been able to identify beforehand an accessible decible subclass. For instance, in the cases of conjectures ϕ_{102}, ϕ_{109}, ϕ_{114} and ϕ_{203}, all of which are reducible to $\mathcal{L}_{SMA}$ by the current set of *remove* rules and elimination equations, *rewrite3*/5 and *rewrite4*/5 first tried to convert them to $\mathcal{L}_{PrA^*}$. As a result, for ϕ_{102}, ϕ_{109}, ϕ_{114}, both rewriters consumed about tenfold the time required by *rewrite5*/5, *rewrite6*/5 and *decide1*/1, whereas, for ϕ_{203}, this factor increased to 100. These results account for the fact that, although *rewrite4*/5 had the highest percentage of best time performances (due to several of the cases where a conjecture could be reduced to $\mathcal{L}_{PrA^*}$), it also exhibited the the second worst mean time (due to the conjectures that could not be reduced to this subclass).

Four main conclusions can be derived from this experiment. Firstly, there is a domain in which the mechanisms present in *decide1*/1 effectively reduce the rewriting search space generated in the presence of RGM. Secondly, the combination of the strategic elements of *decide1*/1 seems to prevent cases where limited search guidance locally damages the performance of the system. Also, RGM makes a significant contribution to the inference resources of all systems, since *rewrite1*/5, which is not interfaced to it, proved to be efficient but deductively weak: its success rate of 21.1% compares unfavourably with more than 90% for the other systems. Finally, for some of the cases where $\mathcal{R}_{PA^*}$ suffices for the reduction of formulae to a sublanguage, *decide1*/5 proved to be fairly inadequate, from the viewpoint of time performance.

10.2 Weak Simplification

The same set of formulae used to test the rewriters has been supplied to the plans *decide1*/1, *decide2*/1, *simplify*/1 and *weak_simplify*/1.

i. *decide2*/1 has the same search control as *decide1*/1 for the selection of decidable sublanguages, the hierarchisation of the deviant symbols and the choice of *remove* rules, but is interfaced to a stronger version of RGM that explores conjecture subexpressions other than the particular one directly involved in semantic matching.



Decide1/1 was the most efficient of all six systems that have access to RGM.

Figure 10.1: Rewriters and *decide1*/1 (I)

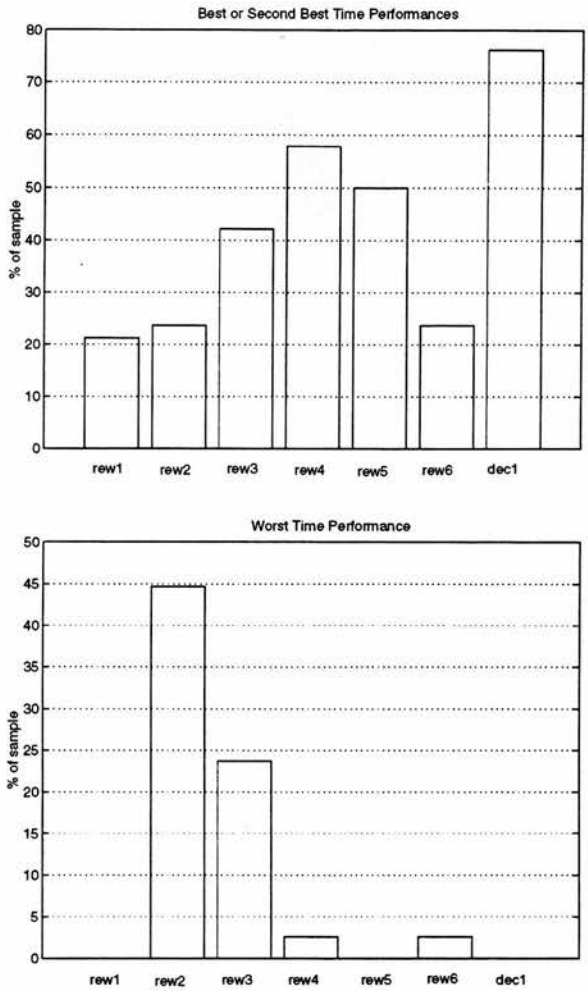
ii. *simplify*/1 explores the fact that, even if a formula is not reducible to a decidable sublanguage, the rewriting process simplifies it, in the sense that the number of occurrences of deviant symbols usually decreases. When the reduction fails, its output consists of the initial formula and one simplified formula for each sublanguage, corresponding to the failed reduction attempts.

iii. *weak_simplify*/1 imposes additional restrictions on the selection of equations for the elimination of disagreements[7].

In the particular case of $\mathcal{E}_{PA^*}$, *weak_simplify*/1 excludes two of its equations, $v \times 1 = v$ and $v^1 = v$. The results presented in table G.6 reveal that, on the average, the simplifiers performed better than the deciders. In particular,

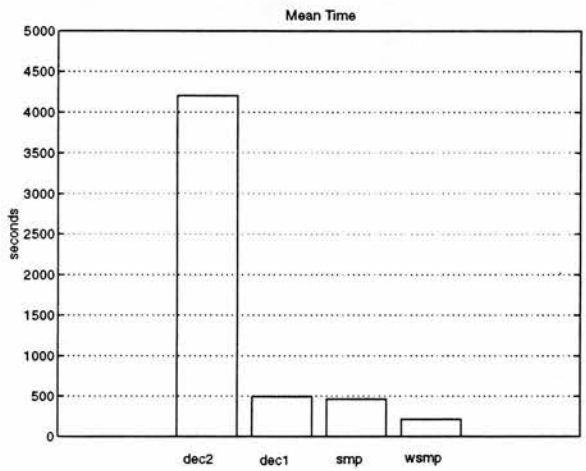
* The performance of *simplify*/1 remained close to *decide1*/1 in most cases, and improved considerably for some of the problems where the primitive rules suffice for normalisation. As a result, a reduction of 7% in the mean time consumed

[7] The set of conjectures is listed in table G.1. The above plans have already been described in chapter 9.



Decide1/1 was either the fastest or the second fastest system to complete a successful reduction into a decidable sublanguage in about $\frac{3}{4}$ of the sample (top graph). Moreover, it was not associated with any of the cases of worst time performance (bottom graph).

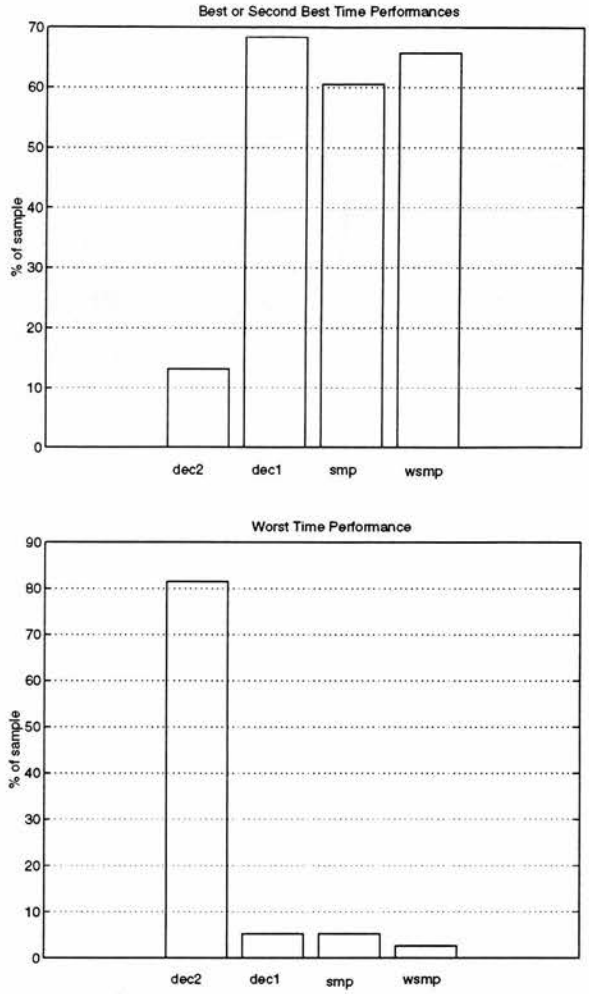
Figure 10.2: Rewriters and *decide1*/1 (II)



dec1	decide1/1
dec2	decide2/1
smp	simplify/1
wsmp	weak_simplify/1

The weak simplifier exhibited the lowest average running time of all four plans in this sample.

Figure 10.3: Deciders and Simplifiers (I)



dec1	decide1/1
dec2	decide2/1
smp	simplify/1
wsmp	weak_simplify/1

Both simplifiers and *decide1*/1 showed a similar performance from the viewpoint of running time. They all had a combined rate of best or second best performances of at least 60%, and worst performance of 5% or less. *Decide2*/1, on the other hand, was associated with most of the cases of worst performance.

Figure 10.4: Deciders and Simplifiers (II)

by *decide1*/5 has been observed for the simplifier, as indicated in table G.7 and figure 10.3.

* *Decide1*/1 had a better performance than *simplify*/1 in 55.6% of the 36 conjectures in which both plans succeeded, as shown in table G.8. Time increments brought about by the simplifier for these cases, however, have been relatively small.

* Amongst the cases where *simplify*/1 outperformed *decide1*/1 (44.4%), there are examples of substantial time reduction, e.g. 92.9% for ϕ_{204} and 94.3% for conjecture ϕ_{302}. The disadvantage of *decide1*/1 with respect to *rewrite1*/4 is overcome in both simplifiers, considering that their initial module, which is absent from the deciders, is limited to the application of primitive rules.

* The time consumed by the exhaustive application of the rules of $\mathcal{R}_{PA^*}$ proved to be confined to the order of seconds, and, as a result, even in the event of failure, the amount of time spent in the process has no significant effect on the time required by disagreement elimination (order of minutes).

With respect to the two extreme cases in the deductive scale — *weak_simplify*/1 and *decide2*/1 — the gain in efficiency seems to have outweighed deductive losses.

* *Decide2*/1, which operates with a more powerful disagreement elimination mechanism, generates a larger decidable subclass than the other plans. The transformation of conjectures ϕ_{404} and ϕ_{405} into a decidable sublanguage has been possible only under the control of this plan.

* *Decide1*/1 outperformed *decide2*/1 in 91.7% of the cases where both succeeded (36 conjectures). The increase in running time has been significant, reaching a factor of 50 for conjecture ϕ_{103}.

* *Weak_simplify*/1, in spite of deductive losses caused by restrictions on disagreement elimination, exhibited the best performance of all four plans, consuming less than half the time required by the second fastest plan, *simplify*/1, in the average case. When attention is restricted to those conjectures for which it suc-

ceeded, a reduction in processing time w.r.t. *decide1*/1 has been observed in all but two cases (conjectures ϕ_{118} and ϕ_{119}).

In summary, the use of the simplifiers seems to be more advantageous than the deciders, for two reasons: they always deliver a rewritten formula, even when the deciders fail, and they provide a significant time improvement in those cases where context transformation is superfluous. *Decide2*/2, on the other hand, has a very poor performance w.r.t. the plans that operate with the standard disagreement elimination mechanism. Its use could therefore be confined to those cases where other plans fail. The results also suggest that equations having a variable for rhs or lhs expression should be excluded from the first disagreement elimination attempt, thus confirming the outcome of similar experiments described in section 9.1.2. As a result, in the course of simplification, the application of plans should start with *weak_simplify*/1, followed by *simplify*/1, in the event of failure, and finally *decide2*/1. Since the simplifiers always deliver a transformed formula, the output of a plan can be the input of a subsequent one. The search space for the combined plan, however, would have to be first examined, to determine how to minimise the negative effects when it fails.

10.3 Verification Conditions

The role of proof plans in the verification of program correctness depends not only on their efficiency, but also on the amount of verification conditions (v.c.) that fall in the extended decidable classes. Verification conditions for implemented programs, in contrast with those for theoretically computable functions, have to be empirically surveyed. Since tests must be limited to finite samples, certain criteria have to be met if results are to be generalised to larger domains. Bundy suggested three guidelines for the selection of conjectures in general, the first of which recommends the use of *dissimilar examples*[8]. In the domain of normalisation, it could be translated along two lines,

(a) the partition of the universe of formulae into finitely many disjoint subclasses, and

[8] Personal communication from Alan Bundy (Blue Book note 863).

(b) the uniform selection of conjectures from all subclasses[9].

Independently supplied conjectures, identified as representative by researchers in the area, according to their own perception and experience, form a second sample. Finally, *challenging examples* may be also added, although verification conditions are usually trivial problems, from the mathematical viewpoint[10]. Concerning the second guideline, according to Boyer and Moore,

> "Perhaps more realistic data is that obtained during the proof of the verification conditions for the FORTRAN version for our fast string searching algorithm. We regard this set of 53 lemmas and verification conditions to be quite representative of the verification of practical programs."
> ([Boyer & Moore 88], p. 121)[11]

The set of v.c. for this program contains seven formulae which concern properties of strings and therefore do not belong to the language of arithmetic. However, related arithmetical theorems, listed in tables G.9 and G.10, have to be proved in the process[12]. These lemmas have been supplied to *weak_simplify*/1, the most efficient of the four plans already built. An additional test has been conducted with its first subplan, *simplify1*/2, which does not eliminate disagreements, to determine the effect of RGM in deductively strengthening the weak simplifier. *Remove* rules for the new deviant symbols with respect to both $\mathcal{L}_{PrA^*}$ and $\mathcal{L}_{SMA}$ are listed in appendix H. The time performances of *simplify1*/2 and *weak_simplify1*/1 are described in tables G.12 and G.13. With respect to these results,

- *Simplify1*/2 has been successful in the reduction of more than half of all lemmas (56.8%). The inclusion of the rule generation mechanism, embedded in *simplify2*/2, the second subplan of *weak_simplify*/1, increases the success rate to $\frac{2}{3}$

[9] This guideline has been partially followed in the test of the rewriters, as it can be observed in appendix G, since the conjectures have been taken from three disjoint classes. Even though they do not cover the full set of formulae of $\mathcal{L}_{PA^*}$, a relevant domain for program verification is nonetheless included.

[10] See [Käufl 89].

[11] Emphasis has been added to the original text.

[12] See [Boyer & Moore 79], p. 291-304 (for the set of verification conditions) and the first appendix of the same book (for the additional arithmetical lemmas).

of the sample[13].

* The time consumed on the average for a successful reduction was 524s for the weak simplifier and only 3s for *simplify1*/2. If lemmas 227 and 306 are excluded, the first result falls to 25s. Besides, *weak_simplify*/1 completed 39 reductions (72.2% of all successful cases) in less than 5s, as indicated in tables 10.1 and 10.2.

* 48.1% of the sample has been reduced to $\mathcal{L}_{PrA^*}$, and 18.5%, to $\mathcal{L}_{SMA}$. In spite of the prominence of $\mathcal{L}_{PrA^*}$, $\mathcal{L}_{SMA}$ is nonetheless a relevant component of the process, and the ability to explore several decidable subclasses allows proof plans to discover shorter rewriting paths for certain expressions.

* In the case of failure, the average times consumed respectively by the plan and its subplan were of 4.5min and 20s, which suggest that their use as interfaces to an heuristic prover would not cause any significant damage to the efficiency of the resulting system.

* All the lemmas which *weak_simplify*/1 failed to reduce to a decidable sublanguage have nonetheless been successfully simplified into $\mathcal{L}_{PA}$.

* Concerning the final simplified formulae, since *remove* rules for rmdr, gcd, / and gfc tend to considerably increase the size of the rewritten formulae, the reduction of lemmas containing these symbols, such as lemmas 227 and 306, was more time consuming.

The expanded formulae in these last two cases may not provide the best option for the establishment of their validity, due to the global complexity of the reduction process and the number of quantifiers introduced for the removal of deviant symbols. Such cases confirm previous expectations that the use of total *remove* rules, although feasible in principle, may turn out to be inadequate, on certain occasions, from the point of view of global efficiency. The inclusion of additional partial *remove* rules for totally removable symbols could, for this reason, improve the performance of the simplifiers.

[13] Three lemmas (186, 187 and 305) are already members of a decidable sublanguage. If they are excluded from the sample, the success rates for *simplify1*/2 and *weak_simplify*/1 drops respectively to 55.1% and 65.4%.

Plan	Sample Size	Success Rate %	Decidable Sublanguage (%)		Time Range (% of successfully reduced lemmas)			
			$\mathcal{L}_{PrA}$	$\mathcal{L}_{SMA}$	0 - 5 s	5 - 10 s	10 - 100 s	+100 s
simplify1/2	81	56.8	44.4	12.3	82.6	10.9	6.5	0.0
weak_simplify/1	81	66.7	48.1	18.5	72.2	11.1	9.3	7.4

For more than 70% of the successful reductions of verification conditions, both the weak simplifier and its subplan *simplify1/2* completed each reduction in 5s or less.

Table 10.1: Success Rates – Simplifiers

Plan	Sample Size*	Success Rate %	Success		Failure	
			Mean (s)	Deviation (s)	Mean (s)	Deviation (s)
simplify1/2	78	55.1	3.1070	3.5113	20.6086	19.8180
weak_simplify/1	78	65.4	524.0705	2,496.7334	270.7111	151.6404

* lemmas 186, 187 & 305 excluded

The higher success rate of the weak simplifier with respect to its subplan *simplify1/2* was achieved at the expense of processing time, whose increase can be attributed to RGM.

Table 10.2: Statistical Data – Simplifiers

10.4 Randomly Generated Formulae

The boundaries of a decidable subclass extended by a simplifier can be statistically delineated once a partition for the underlying language is given, as mentioned in section 9.6. Given a random generator of formulae, if it is employed as a source of conjectures for the simplifier, a probability rate that reflects the proportion of elements that belong to the extended class is then assigned to each element of the partition[14].

The resulting statistical map allows the task of determining whether a formula belongs or not to the extended class to be decomposed into two operations. The element $\boldsymbol{P}$ of the partition to which the formula belongs is identified in the first place. The probability associated with $\boldsymbol{P}$ then determines whether the conjecture should be supplied to the simplifier or not. A low concentration of elements of a decidable subclass in $\boldsymbol{P}$ would suggest that a distinct proving strategy should be sought for expressions of that domain.

Experiments with a random generator of formulae for the language

$$\{0, 1, \mathrm{s}, +, \times, \exp, <, \leq\}$$

have been conducted with *simplify*/1. The set of formulae of this language has been decomposed into subsets according to the formula depth, and eight of such disjoint subclasses have been examined. Since formulae are random, they lack any particular mathematical interest. Results are listed in tables G.17 and G.18, and also in figures 10.5 and 10.6. A few comments follow.

- The proportion of formulae reduced to a decidable sublanguage by the simplifier decreases as the formula depth increases. For the final subclass (depth 10), the success rate of $\frac{1}{3}$ suggests that an enlarged set of *remove* rules or stronger versions of RGM should be considered instead.

- For formulae up to depth 8, on the other hand, the global success rate of at least $\frac{2}{3}$ indicates that the current rewrite rule set, in association with RGM, is a

[14] This approach is based on Monte Carlo methods, which employ modelling techniques, empirical simulation and sampling to evaluate e.g. functions and definite integrals. See [Hammersley & Handscomb 64].

relevant tool for the simplification of arithmetical expressions.

* The role of $\mathcal{L}_{SMA}$ is substantially less significant than the role of $\mathcal{L}_{PrA^*}$, both from the point of view of efficiency and relative size of the extended subclasses generated from it.

* A linearised representation for the logarithm of mean time versus formula depth, indicated in figure 10.6, reveals a gradient of 0.48, 1.41 and 0.22 respectively for the reduction of formulae to $\mathcal{L}_{PrA^*}$, to $\mathcal{L}_{SMA}$ and for undecided formulae. As a result, the average time for each of such cases is supposed to increase respectively by a factor of 1.6 ($= e^{0.48}$), 4.1 and 1.3, whenever the depth of a formula increases by 1. Once again, this result indicates the greater complexity of the task of converting a formula into an element of $\mathcal{L}_{SMA}$.

Additional experiments could provide statistical descriptions of the extended classes from the viewpoint of other syntactic parameters, such as

(i) number of occurrences of a particular (deviant) symbol in the formula,

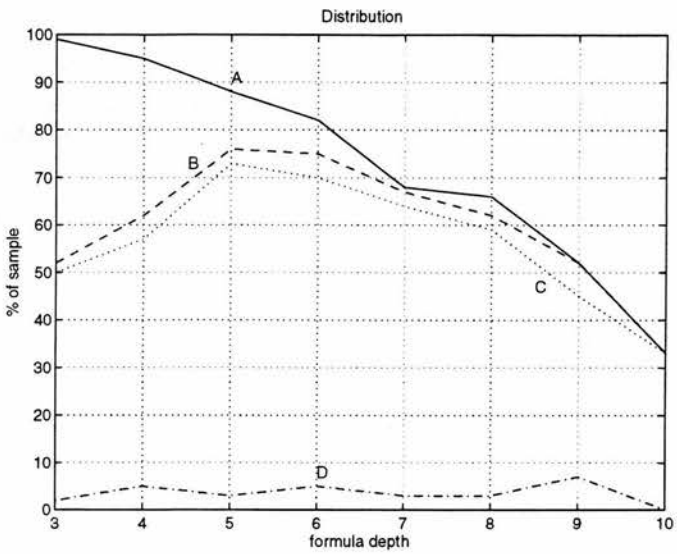
(ii) maximum depth of atoms that occur in the formula (instead of the depth of the formula itself),

(iii) the ratio of the number of occurrences of variables to the number of occurrences of individual constants, etc.

Such measures would collectively provide more accurate guidance to a theorem prover faced with the choice of attempting simplification or not.

10.5 Comparative Assessment

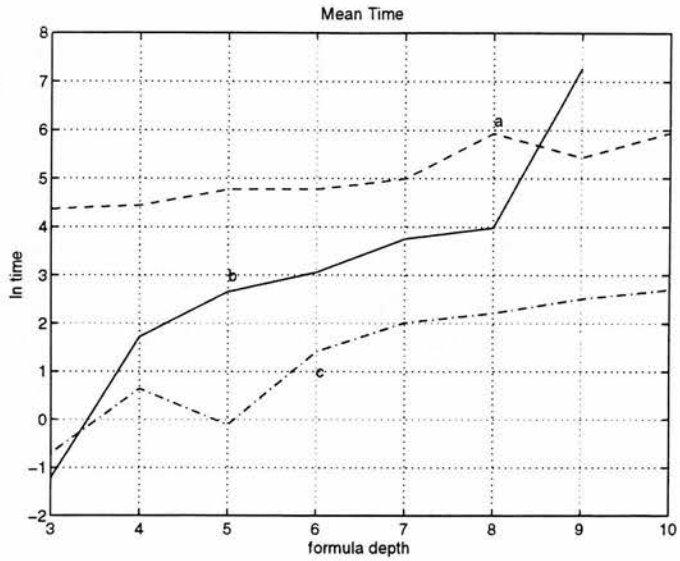
Given that both *weak_simplify*/1 and the linear procedure of *Nqthm* have been applied to a common sample of arithmetical v.c., their strengths and deficiencies in this particular domain may be compared. Although the success rate of *Nqthm* in this sample was 100%, its simplifier, as indicated in table G.11, was able to decide only 67.9% of all verification conditions, while the remaining lemmas required the use of the inductive prover. Both simplifiers therefore showed essentially the same performance in this representative set, given that the result for *weak_simplify*/1 was 66.7%. When the role

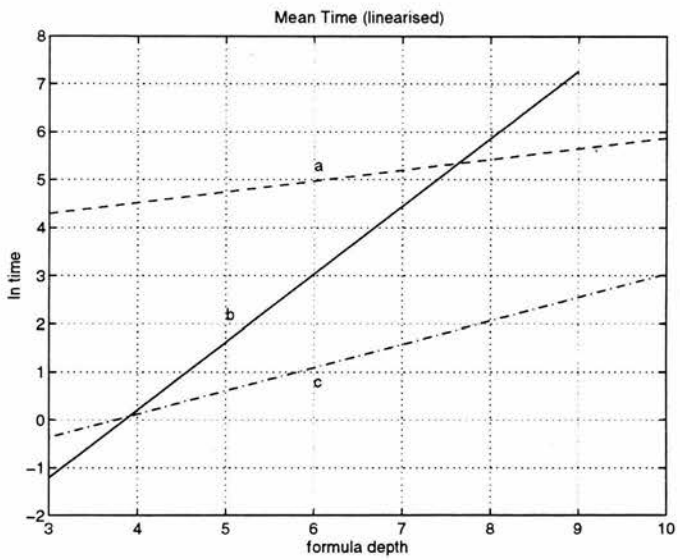


$\Sigma_{PrA^\bullet}$	extension of $\mathcal{L}_{PrA^\bullet}$
Σ_{SMA}	extension of $\mathcal{L}_{SMA}$
A	$\Sigma_{PrA^\bullet} \cup \Sigma_{SMA}$
B	$(\Sigma_{PrA^\bullet} - \mathcal{L}_{PrA^\bullet}) \cup (\Sigma_{SMA} - \mathcal{L}_{SMA})$
C	$\Sigma_{PrA^\bullet} - \mathcal{L}_{PrA^\bullet}$
D	$\Sigma_{SMA} - \mathcal{L}_{SMA}$

Up to formula depth 9, more than 50% of the sample belongs to the global extended decidable class, which includes successfully reduced formulae plus those originally inside the decidable sublanguages (graph A). The amount of formulae actually reduced into a decidable sublanguage (graph B) was in the range of 50% to 75%, again up to formula depth 9. Most of the successfully reduced formulae were converted into $\mathcal{L}_{PrA^\bullet}$ (graph C), whereas $\mathcal{L}_{SMA}$ had a comparatively marginal role in the experiment (graph D).

Figure 10.5: Randomly generated formulae (I)





$\Sigma_{PrA^\bullet}$	extension of $\mathcal{L}_{PrA^\bullet}$
Σ_{SMA}	extension of $\mathcal{L}_{SMA}$
a	undecided formulae
b	$\Sigma_{SMA} - \mathcal{L}_{SMA}$
c	$\Sigma_{PrA^\bullet} - \mathcal{L}_{PrA^\bullet}$

The time performance of the weak simplifier is influenced by the depth of the input formula (top graph). Of the two decidable sublanguages studied, $\mathcal{L}_{SMA}$ showed the highest rate of time increase with formula depth. Lesser time increases were observed for formulae outside the extended decidable class (bottom graph).

Figure 10.6: Randomly generated formulae (II)

of the decision procedures is taken into account, *Nqthm*'s simplifier exhibited however a poorer result, considering that, of the 55 v.c. that it was able to simplify to a propositional constant, only 37 (or 45.7% of the sample) actually required the intervention of the linear arithmetic procedure[15].

In spite of the almost identical performances of both simplifiers, the subsets of v.c. decided by each of them did not coincide. Some of them have been successfully transformed only by *weak_simplify*/1, as indicated below.

Example 10.5.1 *Lemma no. 180,*

$$rmdr(y, 1) = 0$$

requires the application of a remove *rule for rmdr,*

$$rmdr(v_1, v_2) = v_3 \Rightarrow (v_2 = 0 \wedge v_3 = v_1) \vee \vee (v_2 \neq 0 \wedge (\exists v_4)(v_1 = v_4 \times v_2 + v_3 \wedge v_3 < v_2))$$

and the resulting formula,

$$(1 = 0 \wedge 0 = y) \vee (1 \neq 0 \wedge (\exists v_4)(y = v_4 \times 1 + 0 \wedge 0 < 1))$$

can be reduced to an element of the decidable sublanguage PrA,*

$$(1 = 0 \wedge 0 = y) \vee (1 \neq 0 \wedge (\exists v_4)(y = v_4 + 0 \wedge 0 < 1))$$

by a remove *rule for* $\times$, $1 \times v \Rightarrow v$. *This problem is handled by the initial module of* weak_simplify*/1, since there is no disagreement to be eliminated.* □

Nqthm, on the other hand, required induction to prove the above lemma. The strength of proof planning in this context, as well as of any other rewrite-based interface, derives from its ability to deal with quantifiers, which the linear procedure of *Nqthm* lacks. As a result, several of the total *remove* rules listed in table H.1 have no use in the Boyer and Moore prover. Even in the absence of quantifiers, given a particular set

[15] All the experiments involving the Boyer & Moore prover reported in this chapter have been conducted with *Nqthm-1992*, the latest publicly available version of the system. Its main features are described in [Boyer & Moore 93].

of quantifier-free additional lemmas (or rewrite rules), there are conjectures which are successfully decided by the proof-plan-based simplifiers but not by the linear procedure, as indicated below.

Example 10.5.2 *Let ϕ be a formula in conjunctive normal form,*

$$x + x < 1 \ \vee \ x < x \times (x + x)$$

and let E be an arithmetical lemma,

$$v_1 < v_1 \times v_2 \ \equiv \ (v_1 \neq 0 \wedge 1 < v_2)$$

i. If ϕ is supplied to Nqthm, *the required linearisation hypothesis is first added to it,*

$$0 \leq x \ \supset \ (x + x < 1 \vee x < x \times (x + x))$$

then it is normalised to $x < 0 \ \vee \ x + x < 1 \ \vee \ x < x \times (x + x)$. Since the linearised form of its negation,

$$0 - x \leq 0 \ \wedge \ 1 - 2x \leq 0 \ \wedge \ x \times (x + x) - x \leq 0 \quad (*)$$

is DAG-satisfiable, additional hypotheses have to be sought*[16]*. For the heaviest multiplicand of $(*)$, $x \times (x + x)$, lemma E, or rather its restricted conditional form, $(v_1 \neq 0 \wedge 1 < v_2) \supset (v_1 < v_1 \times v_2)$, provides a suitable additional hypothesis, once it is instantiated to*

$$(x \neq 0 \wedge 1 < 2x) \ \supset \ x < x \times (x + x) \quad (**)$$

The linearised consequent of $(**)$ can then be conjoined to $(*)$,

$$0 - x \leq 0 \ \wedge \ 1 - 2x \leq 0 \ \wedge \ x \times (x + x) - x \leq 0 \ \wedge \ 1 + x - x \times (x + x) \leq 0$$

and once again supplied to the decision procedure for DAG. After the removal of the heaviest multiplicand, the resulting formula,*

[16] The term $x \times (x + x)$ can be dealt with as a new individual variable, since $\times$ is not a symbol of $\mathcal{L}_{DAG^*}$.

$$0 - x \leq 0 \ \wedge \ 1 - 2x \leq 0 \ \wedge \ 1 \leq 0$$

is found to be unsatisfiable in DAG. It remains to be proved that the condition of* $(**)$, $x \neq 0 \wedge 1 < 2x$, *follows from the current (empty) set of assumptions, i.e. that* $LA \models x \neq 0 \wedge 1 < 2x$. *The linear procedure is called once again, as follows.*

(a) non-negative status of variables.	$0 \leq x \supset (x \neq 0 \wedge 1 < 2x)$
(b) negation of resulting formula.	$0 \leq x \ \wedge \ (x = 0 \ \vee \ 2x \leq 1)$
(c) linearisation of atoms.	$0 - x \leq 0 \ \wedge \ ((0 - x \leq 0 \ \wedge \ x \leq 0) \ \vee \ 2x - 1 \leq 0)$
(d) disjunctive normal forming.	$(0 - x \leq 0 \ \wedge \ 0 - x \leq 0 \ \wedge \ x \leq 0) \vee$
	$\vee \ (0 - x \leq 0 \ \wedge \ 2x - 1 \leq 0)$

When the last formula is supplied to the decision procedure, it is reduced to

$$0 \leq 0 \ \vee \ 0 \leq 1$$

which is valid. Hence the original condition, $x \neq 0 \wedge 1 < 2x$, *is invalid in LA, and as a result the proof of unsatisfiability of* $(*)$ *has to be discarded. Given the absence of other deviant multiplicands or other additional lemmas, no alternative transformation can be carried out, hence the simplifier fails to reduce the original conjecture to a propositional constant*[17].

ii. If ϕ *is supplied to the general-purpose plan* simplify*/1, given that* ϕ *has a single occurrence of a deviant symbol w.r.t.* $\mathcal{L}_{PrA^*}$, *a* remove *rule for* $\times$ *derived from* E,

$$v_1 < v_1 \times v_2 \quad \Rightarrow \quad (v_1 \neq 0 \wedge 1 < v_2)$$

is applied to the conjecture, which is then reduced to a formula of $\mathcal{L}_{PrA^*}$,

$$x + x < 1 \ \vee \ (x \neq 0 \wedge 1 < x + x)$$

□

[17] *Nqthm-1992* can actually prove the above conjecture by induction, both with and without the intervention of E as an additional lemma. On the other hand, when the lemmas

$$v_1 \neq 0 \supset v_2 \times v_1 \not< v_2$$
$$v_1 \times (v_2 + v_3) = (v_1 \times v_2) + (v_1 \times v_3)$$

are available, its simplifier alone is able to reduce the conjecture to $\top$.

This example also indicates that there is a subdomain of $\mathcal{L}_{PA^-}$ where the proof planning approach is more efficient than the simplifier of the Boyer and Moore prover. The linear procedure required the generation of linearisation hypotheses, followed by the linearisation of atoms, the normalisation of the negated conjecture, two calls to the decision procedure, and the selection of an additional hypothesis. *Simplify*/1, on the other hand, applied a single rewrite rule, followed by a call to the decision procedure for quantifier-free Presburger arithmetic.

Concerning the verification conditions that the proof plan was unable to reduce into a decidable subclass, its failure can be justified upon formal limitations of the current rewriting approach.

* Seven cases include a subformula of the form $v_1 = v_2 \times v_3 + v_4$ and four cases include a subformula of the form $v_1 + v_2 = v_3 \times v_4$ where v_1, v_2, v_3 and v_4 represent distinct individual variables. As established in lemma 9.5.3, there is no valid atomic rule in *PA* which could reduce any of these atoms to a formula of either $\mathcal{L}_{PrA^-}$ or $\mathcal{L}_{SMA}$.

* Two cases include atoms of the form $v_1 \times v_2 = s(v_3)$ or $v_1 \times v_2 = v_3 + 1$, neither of which, according to lemma 9.5.3, could be reduced to a decidable sublanguage.

* Five cases contain pairs of atoms of the forms $v_1 \times v_2 = v_3$ and $v_4 < v_5$, and, therefore, cannot be reduced to either $\mathcal{L}_{PrA^-}$ or $\mathcal{L}_{SMA}$ by the strict use of atomic *remove* rules, since this would require either the eliminability of $\times$ in terms of $\mathcal{L}_{PrA^-}$, or the eliminability of $<$ in terms of $\mathcal{L}_{SMA}$.

* Lemmas 210 and 216 could not be solved due to the incompleteness of the rule generation mechanism, whereas lemma 292 would require the introduction of an *ad hoc* atomic *remove* rule, with lhs expression identical to the lemma itself.

* Lemmas 301 and 308 both involve expressions of the forms $v_1 \times v_2 = v_3$ and $v_4 < 1 + v_5$, and while the first atom cannot be replaced with an equivalent formula of $\mathcal{L}_{PrA}$, as shown in lemma 9.5.3, the second atom cannot be replaced with a formula of $\mathcal{L}_{SMA}$.

* Lemma 318 involves the atom $v_1 \times v_2 < v_3$, which cannot be reduced to either $\mathcal{L}_{PrA}$ or $\mathcal{L}_{SMA}$.

The inclusion of implication rewrite rules, discussed in section 3.2.3, and *remove* rules for composite formulae in $\mathcal{R}_{PA}$. would certainly increase the success rate of proof planning, but it would be first necessary to assess the size of the new search space and to determine additional search strategies.

The comparison of performances of the simplifiers cannot be restricted to the verification conditions selected by Boyer and Moore, since, to a certain extent, they may be regarded as a development sample to which the linear procedure of *Nqthm* has been tuned. A new series of experiments has, for this reason, been conducted with a random generator of arithmetical conjectures similar to the one described in section 10.4. As indicated in section G.4, the new sample has 80 formulae equally distributed amongst formula depths ranging from 3 to 10. All of them are quantifier-free, since the linear arithmetic procedure cannot handle quantifiers.

Given that *Nqthm* operates with a many-sorted universe, each conjecture must have its variables restricted to the set of natural numbers, to guarantee that the formulae have the same meaning for both simplifiers. In the language of the Boyer and Moore prover, each conjecture $\phi(v_1, \ldots, v_n)$ is then represented as

$$\bigwedge_{i=1}^{n} numberp(v_i) \supset \phi(v_1, \ldots, v_n) \quad (*)$$

where $numberp(v)$ indicates that $v \in \mathbb{N}$. In order to fully explore the deductive strength of the *Nqthm* simplifier, an additional formula,

$$\bigwedge_{i=1}^{n} numberp(v_i) \supset \neg\phi(v_1, \ldots, v_n) \quad (**)$$

is supplied to the prover jointly with each conjecture of the form $(*)$. Whenever one of the elements of the pair $\langle (*), (**) \rangle$ is LA-valid, the other is LA-invalid. The set of all 80 pairs of such formulae constitutes the *extended random sample*[18].

[18] See lemma E.6.1. The reason for the introduction of an additional formula for each conjecture is the fact that there are at least two cases in which the *Nqthm* simplifier can identify one of them as LA-valid, but not the other as LA-invalid.

Four experiments have been conducted with the quantifier-free random sample. Two of them involved the plans *simplify*/1 and *weak_simplify*/1. The other two required the extended sample and have been conducted with *Nqthm*, one with the set of lemmas described in the appendix A of [Boyer & Moore 79] available for use as additional hypotheses (or rewrite lemmas), and the other in the absence of this set. The results are detailed in tables G.19 and G.20, and also in figure 10.7. Both proof plans exhibited a better performance than *Nqthm* in this sample.

* The success rate of the weak simplifier for this sample (65.0%) remained approximately the same as for the set of v.c. described in section 10.3, whereas, for the *Nqthm* simplifier, in the absence of the set of arithmetical lemmas, it was limited to 43.8% of the (extended) sample.

* Even when the set of lemmas of [Boyer & Moore 79] is made available to *Nqthm*, its success rate for the extended sample increases only to 47.5%.

* For the original sample, the *Nqthm* simplifier was able to reduce only 17 conjectures (21.3% of the random sample) into a propositional constant.

* Of the 43.8% of the (extended) sample where the *Nqthm* simplifier was successful, only 15 cases (or 18.8% of the total sample) required the use of the linear arithmetic procedure. The number of cases that required the intervention of this procedure remained the same when the set of additional arithmetical lemmas was loaded[19].

---

i. Given a conjecture of the form $(*)$ above, if $\phi(v_1, \ldots, v_n)$ can be reduced to $\top$ by the rewriter, the same applies to $(*)$. If, on the other hand, $\phi(v_1, \ldots, v_n)$ is simplifiable to $\bot$, $(*)$ is reducible to

$$\bigvee_{i=1}^{n} \neg numberp(v_i)$$

which is invalid but satisfiable in *PA*, and therefore cannot be reduced to a propositional constant. For such cases, $\neg\phi(v_1, \ldots, v_n)$ could be rewritten to $\top$, and then, according to lemma E.6.1, the original conjecture $(*)$ could be recognised as *LA*-invalid.

ii. As discussed in chapter 3, when the decision procedure for *DAG* identifies a quantifier-free formula ϕ as valid, it follows that ϕ is also valid in *LA*. However, if it is identified as invalid by the procedure, the same conclusion cannot be extended to linear arithmetic. For such cases, it is then possible to determine whether $\neg\phi$ is *DAG*-valid, in which case ϕ is *DAG*-unsatisfiable and $(*)$ is unsatisfiable in the underlying theory.

[19] Even though the share of conjectures successfully decided by the *Nqthm* simplifier that required the

* The performance of the *Nqthm* simplifier does not seem to be sensitive to the depth of a formula, since the success rates for both experiments involving this prover fluctuated around the average rate of about 45% for all the examined depths.

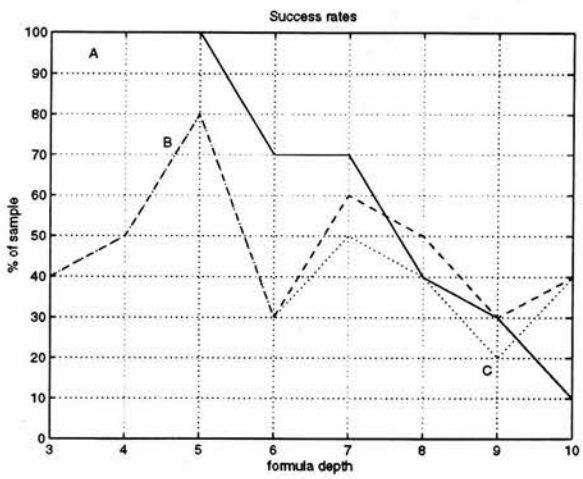
* The success rates for both plans, *weak_simplify*/1 and *simplify*/1, was identical. Table G.20 shows that only in a single case (depth 7) there has been a distinct distribution of successfully reduced conjectures. This case actually concerns conjecture qtf0704, which has been reduced to $\mathcal{L}_{PrA^*}$ by *simplify*/1 and to $\mathcal{L}_{SMA}$ by the weak simplifier. *Weak_simplify*/1 actually consumed twice as much time as the other simplifier to achieve this result, since it explored a distinct path of the rewriting tree.

For formulae of depths ranging between 3 and 7, proof planning exhibited a clear advantage over *Nqthm*, from the viewpoint of success rate. Depths 8 and 9 correspond to a region where both approaches showed a similar performance. *Nqthm* had a visible lead only for depth 10, which could be explained by the fact that the current rewriting approach embedded in *simplify*/1 and *weak_simplify*/1 is limited to the reduction of atoms. Syntactically more complex conjectures require the transformation of composite subformulae, which involves additional search problems, as already mentioned in a previous paragraph.

10.6 Conclusions

Empirical results indicate that proof plans for normalisation are effective in domains where complete sets of syntactically simple (from the point of view of length and occurrences of quantifiers) *remove* rules are available. A representative sample of arithmetical verification conditions showed that explicitly defined functions are abundant in this area, and that therefore general-purpose plans can play a relevant role in program verification. When complete sets of *remove* rules are not available or are too complex,

intervention of the linear procedure was rather limited, the role of the linear procedure is by no means restricted to the simplification module of the prover, since it can be called in the course of inductive proofs as well.



A *Simplify*/1 and *weak_simplify*/1
B *Nqthm* simplifier (with arithmetical lemmas)
C *Nqthm* simplifier

Both plans showed a high success rate (70% to 100%) in the region of formula depth 3 - 7 (graph A). The success rate for the *Nqthm* simplifier oscillated around 47% for the whole sample (graph B). The set of additional lemmas had no significant impact on *Nqthm*'s performance (graph C).

Figure 10.7: Success rates - *Nqthm* and Proof Planning

disagreement elimination procedures increase the strength of the approach through the coverage of additional contexts.

Proof plans also showed their adequacy in addressing extensions of theories. As a result, they represent a solution for one of the core problems in the integration of decision procedures and theorem provers, namely the ability to handle user-defined functions and relations, originally absent from the language of a decision procedure. The flexibility of proof planning was confirmed in the course of its application to formulae of $\mathcal{L}_{PA^{\cdot}}$, an expansion of $\mathcal{L}_{PA}$ that has been dealt with by the introduction of *remove* rules for the new deviant symbols, without any structural change to the original plans. This application has been possible due to the fact that recursive functions and relations are representable in *PA*, and, since the new extensions are conservative, the use of *remove* rules suffices for the expansion of the original decidable sublanguages.

With respect to the simplifier incorporated in *Nqthm*, the simplifiers represented by means of general-purpose proof plans exhibited an overall superior performance from the point of view of success rate. Firstly, for the sample of representative arithmetical verification conditions, to which *Nqthm* has been apparently tuned, both *weak_simplify*/1 and the *Nqthm* simplifier showed essentially the same results. Secondly, for a sample of quantifier-free random conjectures, about $\frac{2}{3}$ of the sample was successfully handled by proof plans, whereas less than half was successfully decided by the simplifier of *Nqthm*. Finally, since the *Nqthm* simplifier includes a linear procedure that is defined only in the quantifier-free fragment of the underlying language, it cannot be applied quantified formulae, whereas the proof planning approach does not share the same limitation.

An additional positive feature of proof plans was their ability to effectively explore two decidable sublanguages of PA^{*}, $\mathcal{L}_{PrA^{\cdot}}$ and $\mathcal{L}_{SMA}$, whereas other approaches tend to concentrate exclusively on the first sublanguage.

Chapter 11

Further Work

Several questions related to the use of proof planning in normalisation require additional investigation. Concerning the efficiency and deductive strength of currently implemented plans, they may be improved through parameter changes affecting, for instance, the rule set and one or more heuristic functions. As described in section 11.1, the arithmetical rule base may be extended to include, among others, general *remove* rules, implication rules and *remove* procedures. Also, the heuristic function that orders decidable classes could be redefined to take into account the complexity of the corresponding decision procedures.

Concerning new applications, tasks other than the removal of symbols still have to be examined. A tentative description for *reorganisation plans* and for a general-purpose disagreement elimination plan can be found in section 11.2. Concerning alternative architectures, systems where proof plans are not limited to the role of interfaces are described in section 11.3.

11.1 Modified Parameters

The parameters of a proof plan for normalisation are distributed in two groups. *Essential parameters* are inherited from the component primitive normalisation methods, and denote therefore the rule set and one or more lists of symbols to which a particular syntactic operation is targeted. *Inessential parameters*, on the other hand, are introduced by conditional methodicals, and include all the heuristic functions that

provide additional guidance for a complex plan[1]. New rewrite systems result from any adjustments to these parameters.

11.1.1 *Remove* Procedures

The current plans can be deductively strengthened by the inclusion of new *remove* rules in the arithmetical rule set. In particular, composite *remove* rules (i.e. rules whose lhs expressions are composite formulae), currently absent from $\mathcal{R}_{PA^*}$, might be a valuable alternative to the use of the extended disagreement elimination mechanism, incorporated in the plan *decide2*/1, which, according to the experiments conducted so far, seems to be rather inefficient. They would probably allow a more adequate handling of long conditional formulae, as those frequently found in program verification[2]. New rules could be considered even for totally removable symbols. Given that total rules for e.g. *rmdr*, *gcd*, / and *gfc* tend to considerably increase the size of the rewritten formula, (quantifier-free) partial rules would improve the performance of the plan in certain subdomains.

Another change involves the enlargement of the concept of *remove* rule, to include a variety of elements that may assist in formula transformation. A first extension takes into account the existence of *general remove rules*, which have the form

$$\delta(v_1, \ldots, v_n) \Rightarrow \delta'\{v_1, \ldots, v_n\}$$

where the set of free variables of the rhs expression is properly contained in the set of variables of the lhs expression. Their generality follows from the fact that any term that replaces a variable v_i that is missing in the rhs expression is removed in the course of rewriting. For instance, given the partial *remove* rule for $\times$,

$$v \times 0 \Rightarrow 0$$

its instances

$$\begin{aligned} (v_1 + v_2) \times 0 &\Rightarrow 0 \\ v_1^{v_2} \times 0 &\Rightarrow 0 \\ (v_1 - v_2) \times 0 &\Rightarrow 0 \end{aligned}$$

[1] Essential and inessential parameters are defined in chapters 4 and 5.

[2] See for instance [Polak 81], p. 207 - 210.

are respectively partial *remove* rules for $+$, exp and $-$ as well. A general rule, therefore, is a family of partial rules.

Apart from rules obtained from equalities and equivalences, rewrite rules derived from strict implications can also deductively enhance the proof planning strategy, even though their application must observe certain restrictions[3]. Non-arithmetical function symbols are amongst those which can benefit from this second extension of the *remove* class.

Example 11.1.1 *Let*

$$y + y < s(x) \ \wedge \ 0 < z + w \ \wedge \ x < \min(l_1) \ \wedge \ \max(l_1) < \min(l_2) \ \wedge \ \max(l_2) < y$$

be an arithmetical conjecture, where $\min(l)$ *and* $\max(l)$ *respectively denote the minimum and maximum elements of a list of natural numbers* l. *With respect to the decidable sublanguage* $\mathcal{L}_{PrA^\bullet} = \{0, 1, s, +, <\}$, *there are two deviant function symbols,* min *and* max. *Let* $\mathcal{R}$ *be a set that contains two rules derived from implications,*

$$\begin{array}{lrcl} R_1. & v < \min(l) & \Rightarrow & v < \max(l) \\ R_2. & v_1 < v_2 \wedge v_2 < v_3 & \Rightarrow & v_1 < v_3 \end{array}$$

R_2 in particular is a general rule, whose instantiation originates specific remove *rules, such as*

$$\begin{array}{lrcl} R_2'. & v_1 < \min(l) \wedge \min(l) < v_3 & \Rightarrow & v_1 < v_3 \\ R_2''. & v_1 < \max(l) \wedge \max(l) < v_3 & \Rightarrow & v_1 < v_3 \end{array}$$

Given that the current strategy for ordering deviant symbols requires that a rule does not reintroduce a deviant symbol previously removed from the conjecture, the removal of min *should antecede* max, *as follows,*

---

[3] Implicational rewrite systems and some of their properties are discussed in appendix D.

$$y + y < s(x) \ \wedge \ 0 < z + w \ \wedge \ \underline{x < \min(l_1)} \ \wedge \ \underline{\max(l_1) < \min(l_2)} \ \wedge \ \max(l_2) < y$$

$$\Downarrow 2 \times R_1$$

$$y + y < s(x) \ \wedge \ 0 < z + w \ \wedge \ \underline{x < \max(l_1) \ \wedge \ \max(l_1) < \max(l_2)} \ \wedge \ \max(l_2) < y$$

$$\Downarrow R_2''$$

$$y + y < s(x) \ \wedge \ 0 < z + w \ \wedge \ \underline{x < \max(l_2) \ \wedge \ \max(l_2) < y}$$

$$\Downarrow R_2''$$

$$y + y < s(x) \ \wedge \ 0 < z + w \ \wedge \ x < y$$

where the occurrence of each rewritten subexpression is positive. Since the resulting formula belongs to $\mathcal{L}_{PrA^*}$, it is then supplied to the corresponding decision procedure. Given that it is unsatisfiable in PA, the same applies to the original conjecture. □

A final extension introduces *remove procedures*, which may achieve the combined effect of various *remove* rules more effectively.

Example 11.1.2 *Let P be a procedure that can be informally described as*

> If a summand t occurs at least n times in each side of an equality (or inequality), then n occurrences of the summand t may be deleted from each side of the equality (or inequality).

and let R_1 and R_2 be the remove *rules*

$$\begin{array}{lll} R_1. & v_1 + v_2 = v_3 + v_2 & \Rightarrow \quad v_1 = v_3 \\ R_2. & v_1 + v_2 = v_1 + v_3 & \Rightarrow \quad v_2 = v_3 \end{array}$$

Given the equation

$$((\underline{x + 2w}) + w \times y^2) + (((\underline{x + 2w}) + 3z) + (\underline{x + 2w})) = z^3 + ((\underline{x + 2w}) + ((2z + (\underline{x + 2w})) + (z + (\underline{x + 2w}))))$$

a single application of P suffices for the removal of all underlined summands, thus generating

$$w \times y^2 + 3z = z^3 + 3z$$

One application of R_1 (or another call of P) suffices then for its conversion to a formula of $\mathcal{L}_{SMA}$. In the absence of P, however, several applications of R_1 and R_2, interleaved with calls to a disagreement elimination mechanism for reordering occurrences of summands, would be needed. □

Deductively more powerful procedures are obtained when semantic identity is taken into account. For the above example, a related procedure could be defined as

> *If a summand t, which occurs at least n times in one side of an equality (or inequality), and a summand t', which occurs at least n times in the other side of the equality (or inequality), can be proved to be equal (in a theory T), then n occurrences of t and/or t' may be respectively dropped from each side of the equality (or inequality).*

11.1.2 Expansion of the Equality Base

Specialised procedures may be also added to the equality base $\mathcal{E}_{PA^*}$ to speed up the elimination of disagreements. The disagreeing pair of symbols $\langle +, \times \rangle$, for instance, is particularly relevant in the study of arithmetical conjectures, but $\mathcal{E}_{PA^*}$ has only two explicitly adequate elimination equations for it,

$$\begin{array}{rcl} (v_1 \times v_2) + (v_1 \times v_3) & = & v_1 \times (v_2 + v_3) \\ (v_1 \times v_3) + (v_2 \times v_3) & = & (v_1 + v_2) \times v_3 \end{array}$$

A procedure that computes a family of additional equations specialised in handling terms in polynomial form in one variable could then be taken into account, whenever the above disagreeing pair requires the transformation of a polynomial into a product. Assuming it is given in canonical form,

$$a_{n+p} v^{n+p} + \cdots + a_1 v + a_0 \quad (*)$$

the procedure would factorise $(*)$ into polynomials of lower degrees, e.g.

$$(b_n v^n + \cdots + b_1 v + b_0) \times (c_p v^p + \cdots + c_1 v + c_0) \quad (**)$$

an operation that includes solving a system of equations of the form[4]

$$\begin{aligned} a_{n+p} &= b_n c_p \\ a_{n+p-1} &= b_n c_{p-1} + b_{n-1} c_p \\ &\vdots \\ a_k &= \sum_{\substack{i+j=k \\ i \leq n, j \leq p}} b_i c_j \\ &\vdots \\ a_0 &= b_0 c_0 \end{aligned}$$

Given that the family of equations of the form $(*) = (**)$ is infinite (since there is at least one equation for each $(n+p) \in \mathbb{N}$), additional constraints must be imposed on their selection in the course of disagreement elimination. A possible strategy would identify the instance of $(*)$ that is relevant for the context, i.e. the instance that can be syntactically matched against a subexpression of the conjecture, as illustrated in the next example.

Example 11.1.3 *Let ϕ be the formula*

$$x^3 + x^2 = x + 1$$

As part of an attempt to reduce it to $\mathcal{L}_{SMA}$, a remove *rule for $+$,*

$$v_1 + v_2 = 1 \quad \Rightarrow \quad (v_1 = 1 \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = 1)$$

is selected, and a fatal disagreement,

δ_1.	$v_1 + v_2$	$=$	$\boxed{1}$
ϕ.	$x^3 + x^2$	$=$	$x\boxed{+}1$

is found. As $v_3 \neq 0 \rightarrow (v_4 = 1 \equiv v_3 \times v_4 = v_3)$, or rather its instance

[4] These polynomials belong to an expansion of $\mathcal{L}_{PA}$ to which infinite many individual constant symbols representing natural numbers have been added.

$$v_5 + v_6 \neq 0 \quad \rightarrow \quad v_4 = 1 \quad \equiv \quad (v_5 + v_6) \times v_4 = v_5 + v_6$$

is the only adequate equation available in $\mathcal{E}_{PA^*}$ for the elimination of this disagreement, its lhs expression is unified with δ_1, thus generating

$$v_5 + v_6 \neq 0 \quad \rightarrow \quad (v_5 + v_6) \times (v_1 + v_2) = v_5 + v_6 \quad \Rightarrow \quad (v_1 = 1 \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = 1)$$

The next disagreement to be tackled involves the pair $\langle +, \times \rangle$.

$\delta_1'.$	$(v_5 + v_6)$	$\boxed{\times}$	$(v_1 + v_2)$	$=$	$v_5 + v_6$
$\phi.$	x^3	$\boxed{+}$	x^2	$=$	$x + 1$

The equation $v_7 \times (v_8 + v_9) = (v_7 \times v_8) + (v_7 \times v_9)$ is then selected, and further disagreements have to be solved thereafter. As indicated in figure 11.1, some of the attempts will lead to failure. The successful generation of a new rule is accomplished once equations applicable to terms, such as

$$\begin{aligned} v + 0 &= v \\ 1 \times v &= v \end{aligned}$$

take part of disagreement elimination, as shown in figure 11.2. The use of equations applicable to atoms, however, considerably expand the search space, as already mentioned in chapter 9. Concerning the unsuccessful trial, it did not explore an important feature of the conjecture, namely, that it is in canonical polynomial form, since the final derived expression, δ_1'''', is not in this form.

This deficiency could be overcome if the elimination procedure for the decomposition of polynomials is chosen for the disagreeing pair $\langle +, \times \rangle$, since it would supply the relevant options and avoid inadequate paths. For the disagreeing pair of expressions, $\langle (v_5 + v_6) \times (v_1 + v_2), x^3 + x^2 \rangle$, the selection of an elimination equation would observe the additional restriction that its rhs expression is matchable to the subexpression of the conjecture. As a result, the elimination procedure has to decompose a single polynomial, $v_{13}^3 + v_{13}^2$, as follows

$$\begin{gathered} v_{13} \times (v_{13}^2 + v_{13}) \\ v_{13}^2 \times (v_{13} + 1) \\ (v_{13}^2 + v_{13}) \times v_{13} \\ (v_{13} + 1) \times v_{13}^2 \end{gathered}$$

	Rule									*Elim Equation*	*Subst*
δ_1	v_1	$+$	v_2	$=$	$\boxed{1}$					E_1	σ_1
δ_1'	$(v_5 + v_6)$	$\boxed{\times}$	$(v_1 + v_2)$	$=$	v_5	$+$	v_6			E_2	σ_2
δ_1''	$v_5 \times (v_1 + v_2)$	$+$	$\boxed{v_6} \times (v_1 + v_2)$	$=$	v_5	$+$	$\boxed{v_6}$			$-$	σ_3
δ_1'''	$v_5 \times (v_1 \boxed{+} v_2)$	$+$	$1 \times (v_1 + v_2)$	$=$	v_5	$+$	1			E_3	σ_4
δ_1''''	$v_5 \times (v_{10} \times (v_{11} + v_{12}))$	$+$	$\boxed{1} \times (v_{10} \times v_{11} + v_{10} \times v_{12})$	$=$	v_5	$+$	1				

ϕ	$x \times (x \times x)$	$+$	$x \times x$	$=$	x	$+$	1				

$\boxed{S}$ *disagreeing symbol*

E_1. $v_4 = 1 \equiv (v_5 + v_6) \times v_4 = v_5 + v_6 \quad (v_5 + v_6 \neq 0)$

E_2. $(v_7 + v_8) \times v_9 = v_7 \times v_9 + v_8 \times v_9$

E_3. $v_{10} \times v_{11} + v_{10} \times v_{12} = v_{10} \times (v_{11} + v_{12})$

σ_1 $\{^{(v_1 + v_2)}/_{v_4}\}$

σ_2 $\{^{v_5}/_{v_7}, ^{v_6}/_{v_8}, ^{(v_1 + v_2)}/_{v_9}\}$

σ_3 $\{^{1}/_{v_6}\}$

σ_4 $\{^{(v_{10} \times v_{11})}/_{v_1}, ^{(v_{10} \times v_{12})}/_{v_2}\}$

An unremovable disagreeing pair, $\langle x, 1 \rangle$, was found between δ_1'''' and ϕ

Figure 11.1: Rule generation: a failed attempt

	Rule	*Elimination Equation*	*Substitution*
δ_1	$v_1 \quad + \quad v_2 \quad = \quad \boxed{1}$	E_1	σ_1
δ_1^1	$(v_5 + v_6) \quad \boxed{\times} \quad (v_1 + v_2) \quad = \quad v_5 + v_6$	E_2	σ_2
δ_1^2	$v_5 \times (v_1 + v_2) \quad + \quad \boxed{v_6} \times (v_1 + v_2) \quad = \quad v_5 + \boxed{v_6}$	$-$	σ_3
δ_1^3	$v_5 \times (v_1 \boxed{+} v_2) \quad + \quad 1 \times (v_1 + v_2) \quad = \quad v_5 + 1$	E_3	σ_4
δ_1^4	$v_5 \times (v_{10} \times v_{11}) \quad + \quad \boxed{1} \times (v_{10} \times v_{11}) \quad = \quad v_5 + 1$	E_4	σ_5
δ_1^5	$v_5 \times (v_{10} \times v_{11}) \quad + \quad v_{10} \times v_{11} \quad = \quad v_5 + 1$		

ϕ	$x \times (x \times x) \quad + \quad x \times x \quad = \quad x + 1$

$\boxed{S}$ *disagreeing symbol*

E_1. $v_4 = 1 \equiv (v_5 + v_6) \times v_4 = v_5 + v_6 \quad (v_5 + v_6 \neq 0)$
E_2. $(v_7 + v_8) \times v_9 = v_7 \times v_9 + v_8 \times v_9$
E_3. $(v_{10} \times v_{11}) + 0 = v_{10} \times v_{11}$
E_4. $1 \times (v_{12} \times v_{13}) = v_{12} \times v_{13}$

σ_1 $\{^{(v_1 + v_2)}/_{v_4}\}$
σ_2 $\{^{v_5}/_{v_7}, ^{v_6}/_{v_8}, ^{(v_1 + v_2)}/_{v_9}\}$
σ_3 $\{^{1}/_{v_6}\}$
σ_4 $\{^{(v_{10} \times v_{11})}/_{v_1}, ^{0}/_{v_2}\}$
σ_5 $\{^{v_{10}}/_{v_{12}}, ^{v_{11}}/_{v_{13}}\}$

Figure 11.2: Rule generation: successful reduction

The above decompositions cover all cases, in view of the requirement that all polynomials are in canonical form[5]. *When the fourth equality,* $(v_{13}^3 + v_{13}^2) = (v_{13} + 1) \times v_{13}^2$, *is selected, it can be RGM-unified with* δ_1' *after the application of the equation* $(v_{14} \times v_{15}) + 0 = v_{14} \times v_{15}$, *and the final* remove *rule,*

$$v_{13} + 1 \neq 0 \quad \rightarrow \quad v_{13}^3 + v_{13}^2 = v_{13} + 1 \quad \Rightarrow \quad (v_{13}^2 = 1 \wedge 0 = 0) \vee (v_{13}^2 = 0 \wedge 0 = 1)$$

or its version free from ground atoms,

$$v_{13}^3 + v_{13}^2 = v_{13} + 1 \quad \Rightarrow \quad v_{13}^2 = 1$$

is obtained. □

11.1.3 Selection of Decision Procedures

The measure function m_c for decidable subclasses, defined in section 6.3.2, does not take into account the complexity of the available decision procedures. Neither does it consider that decision procedures of lower degrees of complexity may exist for proper fragments of a class. When this is the case, such subsets should be regarded as new independent entries in the list of decidable classes. For some of them, the reduction process has to include logical symbols as deviant, e.g. quantifiers in the case of quantifier-free classes. The degree of complexity of each class would then have to be evaluated and incorporated in the measure function[6].

Complexity issues that affect the heuristic choice of *remove* rules also require further investigation. The consequences of the introduction of quantifiers during rewriting still have to be measured. Moreover, increases in the number of occurrences of variables in the rhs expression of a rule should discourage its selection, as discussed in section 6.3.2.

11.2 New Proof Plans

Although the current plans have been applied only to arithmetical conjectures, they are in principle adequate for any theory that admits a decidable sublanguage. Particularly

[5] The two missing options, $(v_{13}^3 + v_{13}^2) \times 1$ and $1 \times (v_{13}^3 + v_{13}^2)$, are already covered by the equations $v \times 1 = v$ and $v_1 \times v_2 = v_2 \times v_1$.

[6] See section 6.3.3.

relevant to the study of verification conditions are theories for data structures such as lists, arrays, trees and finite sets. No structural change affecting the plans is required for this purpose: only the sets of rewrite rules and elimination equations must be adjusted to each context[7].

Extending decidable classes other than sublanguages requires distinct plans, as for instance those involving the reorganisation of occurrences of symbols.

11.2.1 Reorganisation Plans

Normalisation processes such as conjunctive or disjunctive normal forming cannot be reduced to the use of *remove* rules. Given the class of formulae defined by the production

$$fm \;:=\; atom|\neg fm|fm \wedge fm|fm \vee fm$$

and its conjunctive normal form subclass,

$$\begin{array}{rcl} fm' & := & conj|fm' \wedge fm' \\ conj & := & literal|conj \vee conj \\ literal & := & atom|\neg atom \end{array}$$

the main syntactic task is the *stratification* of occurrences of connectives. A specific strategy for this case has elements that are absent from the process of removal of deviant symbols. For instance, given a conjecture generated by fm,

$$\neg\neg((p \vee q) \wedge \neg r)$$

once it is compared to the aimed normalised class, two actions have to be considered,

(i) the stratification of negations beneath conjunctions,

(ii) the removal of occurrences of double negations.

As the *remove* rule for (double) negation is immediately applicable and does not introduce any new occurrence of a symbol in the conjecture, it should be preferred over *stratify* rules[8]. Moreover, the removal of negations, unlike their stratification under

[7] For many-sorted theories, however, variable instantiation has to take into account whether a variable and an expression that is supposed to replace it belong to the same sort.

[8] Although negations are not deviant symbols w.r.t. the final subclass, nested occurrences of this symbol are not allowed.

other connectives, promotes a desirable reduction in the number of logical symbols of the rewritten conjecture. The above formula would then be rewritten to

$$(p \vee q) \wedge \neg r$$

which already belongs to the normalised class.

Besides representing normal forming procedures, *reorganisation plans* (i.e. proof plans that include the application of *reorganisation* rules) may also play a role in the extension of prefix decidable classes of the predicate calculus. This is the case of the set of Π_2-formulae, defined by

$$\begin{aligned} fm &:= exfm | (\forall v) fm \\ exfm &:= qffm | (\exists v) exfm \\ qffm &:= atom | \neg qffm | qffm \wedge qffm | qffm \vee qffm | qffm \supset qffm | qffm \equiv qffm, \end{aligned}$$

which is decidable for the predicate calculus without function symbols, as well as for some other first-order theories. A *reorder* rule such as

$$(\exists v_1)(v_2)(\phi[\not{v}_1] \wedge \psi[\not{v}_2]) \quad \Rightarrow \quad (v_2)(\exists v_1)(\phi \wedge \psi)$$

could then be employed in the reduction of non-Π_2-formulae into this subclass.

The GETFOL mechanism for the extension of the UE-class could be also modelled as a reorganisation plan. The elements of its rule base are all classifiable according to the categories devised to represent normalisation processes in general[9]. The *remove* rules employed by this system include

$$(Qv)\rho[\not{v}] \quad \Rightarrow \quad \rho[\not{v}]$$

There are also *stratify*,

$$\begin{aligned} (\exists v)(\phi \vee \psi)[v] &\Rightarrow (\exists v)\phi \vee (\exists v)\psi \\ (\forall v)(\phi \wedge \psi)[v] &\Rightarrow (\forall v)\phi \wedge (\forall v)\psi \\ (\exists v)(\rho[\not{v}] \wedge \phi[v]) &\Rightarrow \rho[\not{v}] \wedge (\exists v)\phi[v] \\ (\forall v)(\rho[\not{v}] \vee \phi[v]) &\Rightarrow \rho[\not{v}] \vee (\forall v)\phi[v] \\ (\phi \vee \psi)[v] \wedge \gamma[v] &\Rightarrow (\phi \wedge \gamma[v]) \vee (\psi \wedge \gamma[v]) \\ (\phi \wedge \psi)[v] \vee \gamma[v] &\Rightarrow (\phi \vee \gamma[v]) \wedge (\psi \vee \gamma[v]) \\ \gamma[v] \wedge (\phi \vee \psi)[v] &\Rightarrow (\gamma[v] \wedge \phi) \vee (\gamma[v] \wedge \psi) \\ \gamma[v] \vee (\phi \wedge \psi)[v] &\Rightarrow (\gamma[v] \vee \phi) \wedge (\gamma[v] \vee \psi) \end{aligned}$$

[9] The rule base of GETFOL can be found in figure 2.1.

and *reorder* rules,

$$\begin{array}{rcl} \phi[v] \circ \rho[\not v] & \Rightarrow & \rho[\not v] \circ \phi[v] \\ (\rho[\not v] \circ \phi[v]) \circ \psi[v] & \Rightarrow & \rho[\not v] \circ (\phi[v] \circ \psi[v]) \\ \phi[v] \wedge (\rho[\not v] \wedge \psi[v]) & \Rightarrow & \rho[\not v] \wedge (\phi[v] \wedge \psi[v]) \\ \phi[v] \vee (\rho[\not v] \vee \psi[v]) & \Rightarrow & \rho[\not v] \vee (\phi[v] \vee \psi[v]) \end{array}$$

where $\circ$ represents either $\wedge$ or $\vee$. A proof plan for UE-normal forming would have to *remove occurrences of existentially bounded variables from the scope of universal quantifiers.* A very simple mechanism for this task would apply *remove* rules to eliminate undesirable occurrences of existentially bounded variables. Other plans would attempt to reorganise occurrences of such variables, by means of *stratify* and *reorder* rules, until they do not take place in forbidden positions. Strategies of increasing complexity could be built up from the combination of elementary plans.

The use of proof plans for UE-normal forming would overcome potential difficulties related to the extension of the rule base, such as the non-termination and the inefficiency of exhaustive rewriting. Also, the guidelines for the classification of rules would allow the automatic selection of new rules, thus incorporating dynamicity to the system. For instance, the expansion of the rule base suggested in section 2.3 could take place strictly according to the above classification: the new rules

$$\begin{array}{rcl} (\phi \wedge \psi) \vee (\phi \wedge \neg\psi) & \Rightarrow & \phi \\ (\phi \vee \psi) \wedge (\phi \vee \neg\psi) & \Rightarrow & \phi \\ \phi \wedge \neg\phi & \Rightarrow & \bot \\ \phi \vee \neg\phi & \Rightarrow & \top \end{array}$$

would then be added to the *remove* set.

11.2.2 Disagreement Elimination

The extension of decidable sublanguages requires two operations, the removal of deviant symbols and the elimination of disagreements between rules and formulae, when necessary. The currently implemented general-purpose plans deal with each of these operations separately. Deviant symbols are removed at the conjecture level, where rewrite rules are controlled by proof plans. Disagreements, on the other hand, are eliminated at the rule level by a difference reduction procedure specialised in semantic (rule) matching, without the participation of plans.

For theories that admit quantifier elimination, both processes are in principle representable as special-purpose proof plans[10]. In the extension of sublanguages, however, even though the removal of deviant symbols is immediately captured by a relatively simple general-purpose plan, the elimination of disagreements is outside the scope of any elementary representation. Virtually all the basic normalisation operations, such as the removal, stratification and reordering of symbols, as well as similar operations restricted to the domain of individual variables, are involved in this task.

A general-purpose plan for disagreement elimination, therefore, has to start being built from a broad outline for its main components. The elimination of a disagreeing occurrence of a symbol could be described as either its *removal* from the expression or its *moving* to another position, followed by its *replacement* with a different symbol. At least the first two operations can be more immediately represented by means of proof plans, and some of the guidelines to be considered in their construction include

(i) priority of total over partial rules,

(ii) priority of *remove* rules over *stratify* rules, and of *stratify* over *reorder* rules[11].

Further studies, however, are needed to refine this description in terms of more elementary operations.

11.2.3 Combination of Subclasses

Proof plans can also represent, at least partially, the Nelson and Oppen mechanism for the combination of decision procedures[12]. The rules necessary for putting formulae into separate normal form could be included either in a *reorder* set, as for example

$$f^{\mathcal{L}_a}(\mathbf{t}_1) \lhd g^{\mathcal{L}_b}(\mathbf{t}_2) \quad \Rightarrow \quad f^{\mathcal{L}_a}(\mathbf{t}_1) \lhd v \wedge v = g^{\mathcal{L}_b}(\mathbf{t}_2)$$

or in a *stratify* set,

---

[10] See chapter 5.

[11] See section 5.3.1.

[12] The combination of decision procedures is described in section 2.4.

$$
\begin{aligned}
p^{\mathcal{L}_a}[f^{\mathcal{L}_b}(\mathbf{t})] &\Rightarrow p^{\mathcal{L}_a}[v] \wedge v = f^{\mathcal{L}_b}(\mathbf{t}) \\
\neg p^{\mathcal{L}_a}[f^{\mathcal{L}_b}(\mathbf{t})] &\Rightarrow \neg p^{\mathcal{L}_a}[v] \wedge v = f^{\mathcal{L}_b}(\mathbf{t}) \\
f^{\mathcal{L}_a}[g^{\mathcal{L}_b}(\mathbf{t})] \lhd u &\Rightarrow f^{\mathcal{L}_a}[v] \lhd u \wedge v = g^{\mathcal{L}_b}(\mathbf{t}) \\
u \lhd f^{\mathcal{L}_a}[g^{\mathcal{L}_b}(\mathbf{t})] &\Rightarrow u \lhd f^{\mathcal{L}_a}[v] \wedge v = g^{\mathcal{L}_b}(\mathbf{t})
\end{aligned}
$$

where $\lhd$ stands for either $=$ or $\neq$. Given a language $\mathcal{L}_C$, obtained from the combination of languages $\mathcal{L}_1, \ldots, \mathcal{L}_n$, its original class of terms,

$$
tm^{\mathcal{L}_C} := var|cons^{\mathcal{L}_1}|\ldots|cons^{\mathcal{L}_n}|f_1^{\mathcal{L}_1}(\mathbf{tm}^{\mathcal{L}_C})|\ldots|f_{m_1}^{\mathcal{L}_1}(\mathbf{tm}^{\mathcal{L}_C})|f_1^{\mathcal{L}_2}(\mathbf{tm}^{\mathcal{L}_C})|\ldots|f_{m_n}^{\mathcal{L}_n}(\mathbf{tm}^{\mathcal{L}_C})
$$

would then be transformed into n separate layers[13],

$$
\begin{aligned}
tm^{\mathcal{L}_n} &:= var|cons^{\mathcal{L}_n}|f_1^{\mathcal{L}_n}(\mathbf{tm}^{\mathcal{L}_n})|\ldots|f_{m_n}^{\mathcal{L}_n}(\mathbf{tm}^{\mathcal{L}_n}) \\
&\vdots \\
tm^{\mathcal{L}_1} &:= var|cons^{\mathcal{L}_1}|f_1^{\mathcal{L}_1}(\mathbf{tm}^{\mathcal{L}_1})|\ldots|f_{m_1}^{\mathcal{L}_1}(\mathbf{tm}^{\mathcal{L}_1})
\end{aligned}
$$

by a suitable separate normal forming plan.

Concerning the last stage of the procedure, the *equality propagation mechanism*, which *expands* a conjecture by the inclusion of entailed expressions as new conjuncts, it would be necessary to adopt rules of the form

$$
\begin{aligned}
\phi &\Rightarrow (\phi \wedge \delta) \quad (\dagger) \\
\psi \wedge \neg\psi &\Rightarrow \bot \\
v \neq v &\Rightarrow \bot
\end{aligned}
$$

where ϕ is in separate normal form and δ is an entailed equality (that does not occur in ϕ)[14]. Rules of type $(\dagger)$ do not seem to fit in any of the classes presently available in the proof planning framework, even though the process they represent can be easily described as the *expansion* of an expression by the introduction of new conjuncts. Although it is still not clear how common is this process in the context of normalisation, a method for expanding expressions could be created, provided that the expansion would allow e.g. a later application of *remove* or *stratify* rules.

---

[13] In other stratification processes, as for instance the transformation of formulae containing just $\neg, \wedge$ and $\vee$ into conjunctive or disjunctive normal form expressions, the resulting class consists of a hierarchy of layers, each of which provides the base set for the next class. In the present case, however, the base class is the same for all of the new layers, and none of them is contained in any of the others.

[14] The last requirement is necessary to guarantee termination for the application of such rules.

11.2.4 Weaker Forms of Equality Propagation

A modified version of the Nelson and Oppen procedure could be used even when decidable sublanguages share non-logical symbols. If $\mathcal{L}_1$ and $\mathcal{L}_2$ are non-disjoint decidable sublanguages, T is an undecidable theory in the combined language $\mathcal{L}_C$ obtained from $\mathcal{L}_1$ and $\mathcal{L}_2$, and ϕ is a quantifier-free formula of $\mathcal{L}_C$, ϕ would first have to be put into disjunctive normal form, thus assuming the form

$$\phi_1 \vee \cdots \vee \phi_n$$

where $\phi_i \stackrel{s}{=} (\gamma_{i,1} \wedge \cdots \wedge \gamma_{i,m_i})$. Each disjunct would then be converted into a conjunction of formulae of each decidable sublanguage, as follows[15].

i. If non-logical symbols of $\mathcal{L}_b$ and $(\mathcal{L}_a - \mathcal{L}_b)$, $a \neq b, a, b \in \{1, 2\}$, occur in $\gamma_{i,j}$, it is put into separate normal form with respect to $\mathcal{L}_b$ and $(\mathcal{L}_a - \mathcal{L}_b)$. Every disjunct ϕ_i can then be reduced to a conjunction ψ_i of the form

$$\psi_{i,1}^{\mathcal{L}_1} \wedge \psi_{i,2}^{\mathcal{L}_2}$$

ii. As a consequence of lemma 2.4.1, ϕ is unsatisfiable iff every ψ_i is also unsatisfiable. The decision procedure for $T \cap Fml_{\mathcal{L}_j}$ would then be applied to each conjunct $\psi_{i,j}^{\mathcal{L}_j}$ to verify its satisfiability.

iii. In the event both $\psi_{i,1}^{\mathcal{L}_1}$ and $\psi_{i,2}^{\mathcal{L}_2}$ are satisfiable, atomic formulae involving symbols of $\mathcal{L}_1 \cap \mathcal{L}_2$ and entailed by one conjunct are added to the other one, and the decision procedure for satisfiability is applied again. If the extended conjuncts remain satisfiable after a pre-established finite number of iterations of the process, no conclusion about the satisfiability of the conjunction can be derived.

Equality propagation therefore has to include terms other than just variables. Moreover, as the original theory is undecidable, the new mechanism is a *simplifier* (i.e. a rewrite

[15] Rigorously speaking, the resulting expression is not in *separate normal form*, for, according to the original definition, languages (and therefore the decidable classes of formulae, with equality left aside) must be disjoint.

system that reduces a formula to either a propositional constant or a simplified expression) rather than a decision procedure. It could be applied to Peano arithmetic and its extensions to solve some classes of problems.

Example 11.2.1 *Let*

$$u_1 \times x_1 + \cdots + u_n \times x_n = 0 \quad \wedge \quad t_1 + \cdots + t_m < x_i \quad \wedge \quad u_1 \neq 0 \quad \wedge \quad \ldots \quad \wedge \quad u_n \neq 0$$

be a formula where u_j is a term containing only 0, s and variables, t_j has $\times$ as only function symbol, and $x_i \in \{x_1, \ldots, x_n\}$. Since $\mathcal{R}_{PA^-}$ can eliminate $+$ only from the first equation, the standard reduction strategy does not work in this case. The formula then is put into 'separate' normal form w.r.t. $\mathcal{L}_{PrA^-}$ and $\mathcal{L}_{SMA}$, which share two constant symbols, 0 and 1, as shown below.

$\mathcal{L}_{SMA}$	$\mathcal{L}_{PrA^-}$	
$z_1 = y_1 \times x_1$	$y_1 = u_1$	(SNF)
$\vdots$	$\vdots$	
$z_n = y_n \times x_n$	$y_n = u_n$	
$w_1 = t_1$	$u_1 \neq 0$	
$\vdots$	$\vdots$	
$w_m = t_m$	$u_n \neq 0$	
	$z_1 + \cdots + z_n = 0$	
	$w_1 + \cdots + w_m < x_i$	
$z_i = 0$		(PE)
	$y_i = 0 \vee x_i = 0$	
	$\bot$	

SNF and PE respectively denote 'separate' normal form *and* propagated expressions. *Given that the formulae present in the top part of each column are satisfiable in their respective subtheories, equalities and disjunction of equalities entailed by each of them have to be propagated and conjoined to the other formula. In the above example, the atom $z_i = 0$ is first derived from the right column and added to the left column. Thereafter the disjunction $y_i = 0 \vee x_i = 0$ can be obtained and propagated back to the right column, thus allowing the derivation of $\bot$. Hence, the original conjecture is PA-unsatisfiable.* □

The example shows that this procedure can effectively handle certain arbitrarily large conjunctions of literals, being as a result valuable in the domain of program verification. One of its limitations is the potentially infinite number of entailed atomic formulae in the presence of shared function and/or constant symbols. For this reason, criteria for guiding their selection would have to be sought.

Proof plans could adopt this methodology as an alternative approach to the reduction of conjectures into a single decidable class. Whenever this target cannot be achieved, a second trial would consist of splitting ϕ between distinct decidable sublanguages. The list of deviant symbols would then be defined w.r.t. the combination of all such languages, rather than w.r.t. each particular decidable class. *Remove* rules would be necessary only for symbols absent from all such classes. Their removal would be followed by 'separate' normal forming and calls to the necessary decision procedures.

To sum it up, in the context of proof planning and rewriting, the procedure developed by Nelson and Oppen could be employed in two different ways.

i. As a source of decision procedures for the combination of theories in which verification conditions are commonly expressed. Proof plans would then control the process of rewriting conjectures into the decidable class delimited by this procedure.

ii. As the source of an alternative approach to the present strategy of rewriting a conjecture into a single decidable class. Rewrite rules would be needed for removing only certain deviant symbols and for decomposing a conjecture into expressions of the decidable sublanguages.

11.3 Cooperation with Other Modules

The integration of proof plans inside hybrid provers has so far been limited to the preprocessing of conjectures before they are delivered to other modules of the system, as in the case of the simplifiers[16]. Other roles involve, for instance, solving subproblems generated by semidecision procedures.

[16] This feature can be observed in figure 9.1, which describes the structure of the plan *simplify*/1.

11.3.1 Subexpression reduction

There are cases where the reduction of a conjecture to a single decidable subclass cannot be accomplished by a particular rewrite set $\mathcal{R}$, even though distinct subexpressions may be reduced to distinct decidable subclasses. A procedure for the decomposition of a conjecture ϕ into a list of subformulae, such that ϕ results from their propositional combination, could be particularly useful for a quantifier-free conjecture, since its atomic components always constitute a valid propositional partition. Once this list is obtained, the decision procedures are employed as follows.

i. Given a subformula ψ, if it is reducible to a formula ψ' of a decidable sublanguage, ψ' is supplied to the corresponding decision procedure. If it is valid, each occurrence of ψ in the original conjecture ϕ is replaced with $\top$.

ii. If ψ' is found invalid, but $\neg\psi'$ is valid, then ψ is unsatisfiable, being therefore replaceable with $\bot$ in ϕ.

iii. If both ψ' and $\neg\psi'$ are invalid, ψ is not replaced with any propositional constant.

Example 11.3.1 *Given the rules*

$$\begin{array}{lrcl} R_1. & v_1 \times v_2 < v_1 & \Rightarrow & v_1 \neq 0 \wedge v_2 < 0 \\ R_2. & v_1 \times v_2 = 0 & \Rightarrow & v_1 = 0 \vee v_2 = 0 \\ R_3. & v_1 + v_2 = v_2 + v_3 & \Rightarrow & v_1 = v_3 \end{array}$$

and the conjecture

$$x^2 \times (s(y) + z) < x^2 \quad \vee \quad x^4 \times y^2 + y \times z^2 = y \times z^2 + (y \times x^2)^2$$

each disjunct can be reduced to a distinct decidable subclass: the lhs disjunct is reducible to $\mathcal{L}_{PrA^*}$, by the application of R_1 and R_2,

$$\neg(x = 0 \vee x = 0) \wedge (s(y) + z < 0)$$

and the rhs disjunct, to $\mathcal{L}_{SMA}$, by a single application of R_3,

$$x^4 \times y^2 = (y \times x^2)^2$$

Given that the first subformula is found PA-unsatisfiable, it can be replaced with the propositional constant $\bot$. *The second one is PA-valid, being therefore replaceable with* $\top$. *A propositional decider then suffices to establish that the original conjecture is PA-valid.* □

Under these circumstances, a proof plan would be connected to an initial module that decomposes input formulae into lists of (atomic) subformulae. As in the *PVS* system, decision procedures are then employed as complex rewrite procedures. *PVS*, however, does not have a rewriting interface for the reduction of subformulae into a decidable subclass[17].

11.3.2 Inferred Formula Reduction

Another possible link can be established between proof plans and semidecision procedures. A simplifier could tackle subproblems obtained in the course of the construction of a proof or a refutation for a conjecture. The resulting proof (or refutation) tree would then involve the use of both rewrite and inference rules.

Example 11.3.2 *Let* ϕ

$$(\exists x)(y)(z)(x \neq 0 \wedge x^2 \times y + z^4 = x^3 + y \times x^2) \quad \wedge \quad (\exists z)(\exists w)(z^4 + w \times z^2 < w \times z + 5)$$

be a conjecture. Given the α*-rule for conjunctions,*

$$\frac{\phi_1 \wedge \phi_2}{\phi_1, \phi_2}$$

ϕ *is first transformed into the pair*

$$(\exists x)(y)(z)(x \neq 0 \wedge x^2 \times y + z^4 = x^3 + y \times x^2), \quad (\exists z)(\exists w)(z^4 + w \times z^2 < w \times z + 5)$$

whose lhs formula is rewritten by simplify*/1 for the elimination of occurrences of* $+$, *as follows*

[17] *PVS* is described in chapter 2. The above process does not amount to the cooperation of procedures mentioned in the previous section. The decomposition that takes place in the context of cooperation requires 'separate' normal forming, which is absent in the above case.

$$(\exists x)(y)(z)(x \neq 0 \wedge x^2 \times y + z^4 = x^3 + y \times x^2)$$
$$\Downarrow$$
$$(\exists x)(y)(z)(x \neq 0 \wedge z^4 = x^3)$$

which requires the generation of a new remove *rule,*

$$(v_1 \times v_2) + v_3 = v_4 + (v_2 \times v_1) \quad \Rightarrow \quad v_3 = v_4$$

The rewritten formula belongs to $\mathcal{L}_{SMA}$ and is SMA-unsatisfiable. Therefore, the only branch of the above semantic tree is closed, and the original formula, ϕ, is PA-unsatisfiable. □

Tatzelwurm already operates with a tableau-based inference system that is interfaced to several decision procedures. The above example illustrates, however, that proof plans can add as a new ingredient their capability of handling deviant symbols[18].

11.4 Conclusions

Some of the applications of proof planning to normalisation that have been overlooked in previous chapters require minor changes to the parameters of currently implemented general-purpose plans. Other applications require the implementation of new plans and the construction of control structures to improve their performance. Any of these plans may have alternative uses inside a complex system, such as their cooperation with semidecision procedures.

[18] *Tatzelwurm* is described in chapter 2.

Chapter 12

Conclusions

Proof plans may be used to effectively integrate decision procedures inside heuristic theorem provers. Several problems arose in the course of the development of suitable plans for this task, particularly in the domain of Peano arithmetic. Their solution represents the contribution of this study to the fields of mechanical theorem proving and the mechanical verification of program correctness.

The analysis of a series of provers that employ one or more decision procedures revealed that, in most cases, the role of such procedures inside each system could be enlarged. The *reduction* of formulae into a decidable domain suggests an additional strategy from which almost any system could benefit. *Proof plans for normalisation*, devised by Bundy for the control of rewriting processes, provide a suitable framework for exploring this strategy[1]. It involves the use of normalisation tactics and methods, as well as special sets of rewrite rules. A language in which syntactic properties of classes of formulae may be expressed allows the description of normalisers. Methods may be then combined by a planner to generate a plan that meets a description.

Two types of plans defined by Bundy, *special* and *general-purpose* plans, have been investigated. A special-purpose plan has been implemented to represent Hodes' algorithm, a decision procedure based on quantifier elimination. The study of two families of normalisation processes, *quantifier elimination* and *decidable subclass extension*, revealed the main elements required for their implementation as general-purpose plans.

[1] See [Bundy 91].

Decidable sublanguages in particular can be extended by general-purpose plans devised for the removal of deviant symbols. A detailed study of *remove* rules disclosed a set of sufficient conditions for the termination of rewriting for certain classes of such rules. The performance of the resulting normalisers improved after various heuristic functions were added to them. Heuristic guidance is provided for the selection of a target decidable sublanguage, for ordering the removal of deviant symbols, and for selecting *remove* rules. RGM, a difference reduction procedure specialised in the generation of new rules, has been interfaced to these plans to enlarge their range of application.

Plan performance has been empirically assessed by means of a set of representative arithmetical v.c. and a sample of randomly generated conjectures. The results showed that

i. the approach has an adequate performance in an important domain for program verification,

ii. the combined use of the heuristic functions increased the efficiency of the plans,

iii. two arithmetical decidable sublanguages have been effectively explored in the experiment, one of which, $\mathcal{L}_{SMA}$, is usually neglected in the study of arithmetical decision procedures,

iv. proof planning provides a higher success rate than the mechanism of additional hypothesis introduction present in the simplifier of *Nqthm*.

To further support the claim that proof planning is an original strategy for the development of normalisers, section 12.1 compares normalisation plans to other controlled rewrite systems. Section 12.2 then examines the relevance of proof planning in view of its flexibility.

12.1 Controlled Application of Rules

Rewrite rules have been traditionally used in theorem proving for simplification, symbolic evaluation and canonical normal forming[2]. At least some of the successfully

[2] See [Bundy 83], p. 150-153.

reduced conjectures may suggest that proof plans for normalisation perform just a standard combination of these tasks, without adding any new relevant element to them. For instance, the formula

$$(x)(y)(z)(w)(x \neq 0 \supset x \times y + x \times w = x \times z + x^2),$$

has been simplified by the plan *decide1*/1 to

$$(x)(y)(z)(w)(x \neq 0 \supset y + w = z + x)$$

whereas other cases are simple examples of equation solving, e.g.

$$(\exists x)(\exists y)(\exists z)(x \neq 0 \supset (y \neq 0) \supset (x \times y) \times (x + y) + (x \times y) \times (y + z) = x \times y)$$

which has been transformed by the same plan into

$$(\exists x)(\exists y)(\exists z)(x \neq 0 \supset y \neq 0 \supset ((((x = 1 \wedge y = 0) \vee (x = 0 \wedge y = 1)) \wedge y = 0 \wedge z = 0) \vee$$

$$\vee ((x = 0 \wedge y = 0) \wedge ((y = 1 \wedge z = 0) \vee (y = 0 \wedge z = 1)))))$$

All the *remove* rules for + and × present in $\mathcal{R}_{PA^*}$ are classifiable along the three categories above. Hence, any possible effective reduction of a conjecture into a decidable class is an instance of simplification, evaluation or canonical normal forming.

The new ingredient introduced by proof plans in the development of rewrite systems, however, does not take appear at the conjecture level, but in the control of the application of rules. The reduction of a formula to a decidable class by means of a proof plan cannot be identified with standard expression simplification, since exhaustive rewriting is excluded. An arithmetical formula such as

$$(x)(y)(z)(w)(x \neq 0 \supset x \times (x \times y \times z) + x \times (w^2 \times y^2 \times x^2) = x)$$

would be transformed by the decider into

$$(x)(y)(z)(w)(x \neq 0 \supset ((x \times y \times z = 1 \wedge w^2 \times y^2 \times x^2 = 0) \vee$$

$$\vee (x \times y \times z = 0 \wedge w^2 \times y^2 \times x^2 = 1)))$$

which is an element of $\mathcal{L}_{SMA}$, and rewriting would then stop, in spite of the fact that other rules of $\mathcal{R}_{PA^*}$ are still matchable to it. The arithmetical validity of the resulting formula can then be determined by a decision procedure, which is more efficient than a generic rewrite system. One of the guidelines of the proof planning approach asserts that rules should be applied *only while necessary* for a particular purpose, and not exhaustively.

Another important aspect of planned rewriting is its ability to effectively employ virtually any set of rules, given that its control structures prevent non-termination. Plans are able, therefore, to perform tasks that lie beyond the domain of exhaustive rewriting, which is limited to the use of noetherian sets.

Two forms of search control may be embedded in a proof plan[3]. Firstly, the search space of a problem can be reduced by the formal establishment of specific properties of parts of its domain, which are then strictly explored inside such subdomains. This is the case of certain results obtained about the extension of decidable sublanguages, which have been used in the construction of plans that strictly apply *remove* rules for this task, without completeness losses. Secondly, heuristic strategies may be added to a system to improve its performance, possibly at the expense of completeness. This is the case, for instance, of the mechanism of selection of decidable subclasses based on heuristic measures for the complexity of the removal of deviant symbols: the prohibition of the joint use of *remove* rules for deviant and non-deviant symbols actually prevents certain formulae from being reduced into a decidable sublanguage.

Control mechanisms are not an exclusive element of rewrite systems built through proof planning though. In *ordered rewriting*, the domain of the system is ordered, and rules may be applied in either direction, provided that the rewritten expression antecedes the original one according to the pre-established order. *Conditional rewriting* requires a conjecture to satisfy certain conditions before a rule is applied. Finally, in *priority rewriting*, a partial order is defined in the set of rules, as in the case of Markov algorithms[4].

---

[3] Search control is discussed in appendix D.6.

[4] Conditional rewriting is defined in appendix D. Ordered and priority rewriting are briefly discussed in [Dershowitz & Jouannaud 90], p. 259, 306-8.

In both ordered and conditional rewriting, the application of rewrite rules is sensitive to the conjecture alone, whereas proof plans check the properties of the underlying theory as well, such as the existence of decidable classes and the complexity of their decision procedures. Proof plans, on the other hand, have some similarities with priority rewrite systems, to the extent that rules are selected and ordered before their application starts. Since the order is sensitive to the input formula, each plan may then be regarded as a collection of priority rewriters.

Proof Plans & GETFOL

GETFOL is another system that makes use of rewrite rules to enlarge decidable domains, as for instance the class of UE-formulae. Special control structures ensure that its rule set is noetherian. A first attempt to generalise these results suggests that the same mechanism could be adopted in other domains as well.

> "The integration of the `reduce` procedures with GETFOL general-purpose rewriter is also under study. Such integration should allow to meet one of the most promising advantages in designing procedures relying on the notion of reducibility. Since the GETFOL rewriter accepts as input both the formula to rewrite and the set of rewriting rules, decision procedures for new classes of formulae can be obtained by simply changing the set of rewriting rules" ([Armando & Giunchiglia 93], p. 500)

The concrete assessment of the effects of such generalisation on termination and efficiency is, however, lacking. The plain expansion and/or change of a rule set does not necessarily generate a new decision procedure, since rewrite systems may be nonterminating. Also, the exhaustive application of rewrite rules, even when the system is noetherian, is not necessarily the most effective way of extending a class. The use of proof plans, which can apparently represent the GETFOL rewriter, as discussed in chapter 11, could overcome these deficiencies.

12.2 Flexibility versus Efficiency

Several theories admit mechanisms for the reduction of conjectures into a decidable subclass that are more efficient than proof plans. Such mechanisms may consist of several stages of canonical normal forming and simplification, not necessarily based on rewriting. The representation of Hodes' algorithm as a special purpose plan showed that, although some of its steps can be efficiently performed by rewrite rules, the same does not apply to the equational component. Efficiency, however, is not the main concern of the proof planning approach to normalisation: its first purpose is to achieve *generality* and *flexibility*.

> "The hypothesis we are trying to test is that proof plans will provide a modular representation of decision procedures that will facilitate their synthesis and modification and their interaction with the theorem provers in which they are embedded. The cost may be some inefficiency in the implementation of the decision procedures, but the Boyer-Moore experience seems to suggest that this cost is not significant within the context of the complete system." ([Bundy 91], p. 2)

Arithmetical procedures may take into account the commutativity and associativity of sum and multiplication to improve their efficiency. Such properties, however, may not apply to other operations or theories. General procedures, as a result, cannot explore the properties of specific domains, otherwise generality is lost[5].

The advantage of general but less efficient procedures can be observed in domains where new operators are frequently introduced. Explicitly defined functions and predicates do not seem to pose any real problem for a specialised procedure, since they can be usually eliminated in terms of the original language before the procedure is called. There are cases, however, in which the elimination is not possible or the explicit definitions of the new symbols are too complex from the syntactic viewpoint. When the available procedures are highly efficient but specific, they will fail to cope with such cases, and either new extended and efficient versions have to be devised for them, or a more

[5] Specialised procedures can be nonetheless incorporated in the proof planning approach under the form of *remove* or disagreement elimination procedures, as considered in chapter 11.

general approach must be sought. General-purpose procedures have an advantage over their special-purpose counterparts whenever the development of the latter cannot keep the pace of a rapidly changing theoretical background.

> "But what makes it hardest to apply the work on decision procedures to program verification is the presence of user defined functions. (...) Such functions are introduced not by the designer of the theorem prover but by the user when he is confronted with the need to specify a given program. Since decision procedures for these extended theories are not generally available, one must have more powerful proof techniques or be forced to assume the more doubtful conjectures behind a program's correctness." ([Boyer & Moore 88], p. 122)

Given the nature of general-purpose proof plans, it is possible to claim that they provide a general solution to the problem of integrating decision procedures into theorem provers. Integration mechanisms present in some provers can be actually modelled as proof plans.

Proof Plans & *Nqthm*

All the above arguments about flexibility and efficiency can be in principle applied to the comparison between proof planned interfaces and the linear procedure present in *Nqthm*. The extension mechanism chosen by Boyer and Moore, based on the introduction of additional hypotheses, is less flexible than the strategy available under proof planning, since the former requires the core quantifier-free theory to be decidable, and the extension of this theory to any expansion of the original language (without any change to the axiom set) to be decidable as well. The principles that underlie proof planning, on the other hand, are applicable to any theory, since they only require the availability of decidable classes and adequate rewrite rules.

Concerning success rates, proof plans outperformed the *Nqthm* simplifier in a sample of randomly generated formulae, and exhibited approximately the same rate as this prover in a set of representative verification conditions. The proof planning approach could be

even further strengthened by the introduction of new rules and the widening of scope of disagreement elimination, currently limited to atoms. Concerning time performances, the present implementation of the general-purpose plans compares unfavourably with the Boyer and Moore prover, not only because the latter is a well-developed system, but also in view of the specificity of its linear procedure, which is destined to be more efficient than a general-purpose mechanism, at least inside its limited domain of application. It has to be added, though, that, when the number of transformation steps alone is taken into account, there is a subdomain of formulae of $\mathcal{L}_{PA^\bullet}$ where general-purpose proof plans outperform the *Nqthm* simplifier[6].

The linear procedure of *Nqthm*, on the other hand, has two deficiencies. Firstly, it cannot deal with quantified formulae (other than universal formulae, which are valid iff their matrixes are valid as well). Secondly, it operates with a weak reduction function that ultimately relies on the use of implication rules. As a result, even when all undefined symbols are eliminated, there is no guarantee that the transformed formula belongs to the extended decidable subclass, since it is still necessary to check its validity (or rather the unsatisfiability of its negation) in the target subtheory. When the formula is not valid, the whole transformation attempt fails, and this failure does not point to any alternative solution: the only additional action is to blindly start adding other hypotheses until an unsatisfiable formula is obtained, whenever possible.

Since the introduction of additional hypotheses ends up *removing* undefined symbols from a conjecture, proof plans could possibly model the procedure adopted by Boyer and Moore[7]. It seems to be necessary though to take into account rewrite rules derived from implications. Once the properties of implicational rewrite systems have been further investigated and suitable search strategies have been integrated to them, proof planning could even improve the performance of the heuristic mechanisms present in *Nqthm*: the selection of additional hypotheses is presently conditioned by the heaviest multiplicand of a conjecture (to ensure termination), whereas a plan makes use of context-sensitive functions that may point to an alternative ordering for the removal of symbols and lead to a shorter transformation. Even if implication *remove* rules

[6] See example 10.5.2 and section 10.5.

[7] See example 10.5.2.

are added to proof plans, this would not prevent the joint use of equivalence and implication rules, where the former would always have priority over the latter, thus avoiding, under certain circumstances, the disadvantages of implicational rewriting.

The fact that proof plans can handle and introduce quantifiers in the rewritten conjecture carries as disadvantage the need to employ the more complex decision procedures for quantified classes. Short formulae, however, such as most of those selected as representative by Boyer and Moore, stay within limits where decision procedures for e.g. Presburger arithmetic have a reasonable performance, given that one of the major causes for formula expansion, the size of the prefix, has little effect in such cases. Also, in spite of the absence of quantified formulae in the list of representative problems, they can nonetheless occur in the domain of program verification, or any other domain of application for mechanical provers: the ability to handle quantifiers is significant in itself, independently of the complexity of related decision procedures. Finally, the complexity of a decision procedure does not seem to be as determining a factor for the final performance of the extended procedure as the chosen integration mechanism[8]. For several of the selected problems, proof plans provided a straightforward and efficient reduction mechanism. This aspect compensates for any disadvantage related to the use of less efficient decision procedures, at least in certain domains.

12.3 Summary

Several attempts have been made to integrate decision procedures in (heuristic) theorem provers. Most of them are too limited in scope. Proof plans provide a more adequate and general framework to solve this integration problem, based on the reduction of formulae into a decidable domain. They seem to capture the structure of at least two integration mechanisms, present in *Nqthm* and GETFOL.

Even though the reduction of formulae into decidable subclasses apparently amounts

[8] Once again according to Boyer and Moore,

> "... an instantaneous oracle for deciding linear arithmetic problems [...] would increase the speed of our theorem prover on typical program verification problems by less than 3%." ([Boyer & Moore 88], p. 90)

to a standard application of rewrite rules for expression simplification, the originality of proof planning in the domain of normalisation stems from its control devices that prevent exhaustive rewriting and ensure termination. Moreover, even though simplification processes performed by general-purpose plans are less efficient than specialised procedures, the relevance of proof plans for normalisation derives from their generality and flexibility.

Bibliography

[Anderson 70] R. Anderson. Completeness Results for E-Resolution. In *Proceedings of the AFIPS Spring Joint Computer Conference*, pages 653–656, Reston (Va), 1970. AFIPS Press.

[Anderson 79] R.B. Anderson. *Proving programs correct.* John Wiley, New York, 1979.

[Armando & Giunchiglia 93] A. Armando and E. Giunchiglia. Embedding complex decision procedures inside an interactive theorem prover. *Annals of Mathematics and Artificial Intelligence*, 8:475–502, 1993.

[Bachmair 91] L. Bachmair. *Canonical equational proofs.* Birkhäuser, Boston, Mass., 1991.

[Ben-Ari 93] M. Ben-Ari. *Mathematical Logic for Computer Science.* Prentice Hall, New York, 1993.

[Bläsius & Siekmann 88] K. H. Bläsius and J. H. Siekmann. Partial unification for graph based equational reasoning. In E. Lusk and R. Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction*, pages 397 – 414, Argonne (IL), 1988.

[Bläsius 87] K. H. Bläsius. Equality reasoning based on graphs. SEKI Report SR-87-01, Fachbereich Informatik, Universität Kaiserslautern, 1987.

[Boolos & Jeffrey 89] G. S. Boolos and R. C. Jeffrey. *Computability and logic.* Cambridge University Press, Cambridge, 1989.

[Boyer & Moore 79] R.S. Boyer and J.S. Moore. *A Computational Logic.* New York, Academic Press, 1979.

[Boyer & Moore 88] R.S. Boyer and J.S. Moore. Integrating decision procedures into heuristic theorem provers: A case study of linear arithmetic. In J. E. Hayes *et al*, editors, *Machine Intelligence*

11 (Logic and the Acquisition of Knowledge), pages 83–124. Oxford, Clarendon Press, 1988.

[Boyer & Moore 93] R.S. Boyer and J.S. Moore. *A Computational Logic Handbook (Authorized Excerpts from a Proposed Second Edition)*. 1993. Available by ftp from ftp.cli.com.

[Bundy & Welham 81] A. Bundy and B. Welham. Using meta-level inference for selective application of multiple rewrite rule sets in algebraic manipulation. *Artificial Intelligence*, 16:189–212, 1981.

[Bundy 83] A. Bundy. *The Computer Modelling of Mathematical Reasoning*. Academic Press, London, 1983.

[Bundy 88] A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction*, pages 111 – 120, Argonne, Illinois, 1988. Springer-Verlag. Longer version available from Edinburgh as DAI Research Paper No. 349, 1987.

[Bundy 91] A. Bundy. The use of proof plans for normalization. In R.S. Boyer, editor, *Essays in Honor of Woody Bledsoe*, pages 149–166. Kluwer, 1991. Also available from Edinburgh as DAI Research Paper No. 513.

[Bundy *et al* 91] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991. Earlier version available from Edinburgh as DAI Research Paper No 413.

[Chang & Keisler 73] C. C. Chang and H.J. Keisler. *Model Theory*. North-Holland, Amsterdam, 1973.

[Church 56] A. Church. *Introduction to mathematical logic*. Princeton University Press, Princeton, 1956.

[Cleve & Hutter 93] J. Cleve and D. Hutter. Guiding equational proofs by attribute functions. Technical report, German Research Center for A.I. (DFKI), 1993.

[Cooper 72] D. C. Cooper. Theorem proving in arithmetic without multiplication. In B. Meltzer and D. Michie, editors, *Machine intelligence 7*,

pages 91–99, New York, 1972. American Elsevier.

[Curry 76] H. B. Curry. *Foundations of Mathematical Logic.* Dover Publications, New York, 1976.

[Cutland 80] N.J. Cutland. *Computability (An Introduction to recursive function theory).* Cambridge University Press, Cambridge, 1980.

[Dershowitz & Jouannaud 90] N. Dershowitz and J.P. Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B (Formal Models and Semantics), pages 243 – 320. Elsevier and The MIT Press, Amsterdam and Cambridge (MA), 1990.

[Digricoli & Harrison 86] V.J. Digricoli and M.C. Harrison. Equality-based binary resolution. *J.ACM*, 33(2):253 – 289, 1986.

[Dreben & Goldfarb 79] B. Dreben and W.D. Goldfarb. *The Decision problem — Solvable classes of quantificational formulas.* Addison-Wesley, 1979.

[Enderton 72] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, New York, 1972.

[Epstein & Carnielli 89] R. L. Epstein and W. A. Carnielli. *Computability: computable functions, logic, and the foundations of mathematics.* Wadsworth & Brooks/Cole, Pacific Grove (CA), 1989.

[Gabbay & Shehtman 93] D. M. Gabbay and V. B. Shehtman. Undecidability of modal and intermediate first-order logics with two individual variables. *JSL*, 58(3):800–823, 1993.

[Gallier 87] J. H. Gallier. *Logic for Computer Science (Foundations of Automatic Theorem Proving).* John Wiley, New York, 1987.

[Giunchiglia & Traverso 91] F. Giunchiglia and P. Traverso. GETFOL User Manual. Technical Report MRG/Dist 9107-01, DIST - University of Genoa, 1991.

[Gordon *et al* 77] M. Gordon, R. Milner, and C. Wadsworth. Edinburgh LCF. Internal Report CSR-11-77 – Part I, Department of Computer Science, University of Edinburgh, 1977.

[Hammersley & Handscomb 64] J. M. Hammersley and D. C. Handscomb. *Monte Carlo methods.* Methuen, London, 1964.

[Hodes 71] L. Hodes. Solving problems by formula manipulation in logic and linear inequalities. In *Advance Papers of the 2nd International Joint Conference on A.I*, pages 553–559. The British Computer Society, 1971.

[Hopcroft & Ullman 79] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley, Reading, MA, 1979.

[Käufl 89] T. Käufl. Cooperation of Decision Procedures in a Tableau-Based Theorem Prover. Interner Bericht 19/89, Fakultät für Informatik, Universität Karlsruhe, 1989.

[Knuth & Bendix 70] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages p. 263–97. Pergamon Press, Oxford, 1970.

[Kreisel & Krivine 67a] G. Kreisel and J. L. Krivine. *Éléments de logique mathématique: théorie des modèles.* Dunod, Paris, 1967.

[Kreisel & Krivine 67b] G. Kreisel and J. L. Krivine. *Elements of mathematical logic: Model theory.* North Holland, Amsterdam, 1967.

[Lewis & Papadimitriou 81] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, Englewood Cliffs, 1981.

[Liskov & Berzins 86] B. Liskov and V. Berzins. An appraisal of program specifications. In N. Gehani & A. D. McGettrick, editor, *Software Specification Techniques.* Addision-Wesley, Wokingham, 1986.

[Liskov & Guttag 86] B. Liskov and J. Guttag. *Abstraction and Specification in Program Development.* The MIT Press, Cambridge (MA), 1986.

[Loeckx & Sieber 84] J. Loeckx and K. Sieber. *The Foundations of Program Verification.* Stuttgart, John Wiley-Teubner, 1984.

[Loveland 78] Donald W. Loveland. *Automated theorem proving: a logical basis.* North-Holland, Amsterdam, 1978.

[Luger & Stubblefield 89] G. Luger and W. Stubblefield. *Artificial Intelligence and the Design of Expert Systems.* Benjamin/Cummings, Redwood City (CA), 1989.

[Lukey 80] F. J. Lukey. Understanding and debugging programs. *Int. J. Man-Machine Studies*, 12:189–202, 1980.

[Malitz 79] J. Malitz. *Introduction to Mathematical Logic: Set Theory, Computable Functions, Model Theory.* Springer-Verlag, New York, 1979.

[Manna & Waldinger 85] Zohar Manna and R. Waldinger. *The logical basis for computer programming. Vol. 1: Deductive Reasoning.* Addison-Wesley, Reading, 1985.

[Manna 74] Zohar Manna. *Mathematical theory of computation.* McGraw-Hill, New York, 1974.

[Mendelson 87] E. Mendelson. *Introduction to mathematical logic.* Wadsworth & Brooks/Cole, Monterey (CA), 1987.

[Milner 79] R. Milner. LCF: A way of doing proofs with a machine. Internal Report CSR-41-79, Department of Computer Science, University of Edinburgh, 1979.

[Monk 76] J. D. Monk. *Mathematical Logic.* New York, Springer-Verlag, 1976.

[Morris 69] J. B. Morris. E-Resolution: Extension of resolution to include the equality relation. In D.E. Walker and L. M. Norton, editors, *Proceedings of the IJCAI*, pages 287–294, Washington, 1969.

[Mostowski 52] A. Mostowski. On direct products of theories. *Journal of Symbolic Logic*, 17:1–31, 1952.

[Nelson & Oppen 79] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Prog. Lang. Systems*, 1(2):245-257, 1979.

[Nelson & Oppen 80] G. Nelson and D. C. Oppen. Fast Decision Procedures based on Congruence Closure. *J. ACM*, 27(2):356–364, 1980.

[Owre *et al* 92] S. Owre, J.M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *Proceedings of the 11th Conference on Automated Deduction*, pages 748–752, 1992.

[Paulson 87] L. C. Paulson. *Logic and computation : interactive proof with Cambridge LCF.* Cambridge University Press, Cambridge, 1987.

[Peterson 83] G. Peterson. A Technique for Establishing Completeness Results in Theorem Proving with Equality. *SIAM Journal of Computing*, 12:82–100, 1983.

[Plaisted 90] D.A. Plaisted. Mechanical theorem proving. In R.B. Banerji, editor, *Formal Techniques in Artificial Intelligence*, pages 269–320. North Holland, Amsterdam, 1990.

[Polak 81] W. Polak. *Compiler specification and verification.* Springer-Verlag, Berlin, 1981.

[Robinson 49] J. Robinson. Definability and decision problems in arithmetic. *Journal of Symbolic Logic*, 14:98–114, 1949.

[Shapiro 92] S. C. Shapiro, editor. *Encyclopedia of Artificial Intelligence.* John Wiley, New York, 2nd edition, 1992.

[Shoenfield 67] J. R. Shoenfield. *Mathematical Logic.* Reading (MA), Addison-Wesley, 1967.

[Shostak 77] R. E. Shostak. On the SUP-INF Method for Proving Presburger Formulas. *Journal of the ACM*, 24(4):529 – 543, 1977.

[Shostak 79] R. E. Shostak. A practical decision procedure for arithmetic with function symbols. *Journal of the ACM*, 26(2):351 – 360, 1979.

[Skolem 70] T. Skolem. Über einige satzfunktionen in der arithmetik. In J. E. Fenstad, editor, *Selected Works in Logic (by Th. Skolem).* Universitetsforlaget, Oslo, 1970.

[Sommerhalder & Van Westrhenen 88] R. Sommerhalder and S. C. Van Westrhenen. *The theory of computability : programs, machines, effectiveness and feasibility.* Addison-Wesley, Wokingham, 1988.

[Van Harmelen & *et al* 93] F. Van Harmelen *et al.* The clam proof planner: User manual and programmer manual (version 2.0). Technical paper, Department of Artificial Intelligence, University of Edinburgh, 1993.

[Weyhrauch 77] R. W. Weyhrauch. A users manual for FOL. Technical Report STAN-CS-77-432, Computer Science Department, Stanford University, 1977.

Appendix A

List of Symbols

Sets and Functions

Symbol	Meaning	p.
$\wp(\mathcal{A})$	parts of a set $\mathcal{A}$	368
$f\colon \mathcal{A} \to \mathcal{B}$	total function	368
$f\colon \mathcal{A} \rightsquigarrow \mathcal{B}$	partial function	368
$f(\mathcal{A})$	image of f restricted to $\mathcal{A}$	368
$dom(f)$	domain of a function f	368
$rng(f)$	range of a function f	368
$im(f)$	image of a function f	368
$f \circ g$	composition of functions	368
$\mathbb{N}$	set of natural numbers	368
$\mathbb{Z}$	set of integers	419
$\mathbb{Q}$	set of rationals	377
$\mathbb{R}$	set of reals	377
$\ulcorner\epsilon\urcorner$	Gödel number	380
$f \restriction \mathcal{A}$	restricted function	368
$\mu v_i(R(v_1, \ldots, v_n))$	minimalisation operator	370
$\#\mathcal{A}$	cardinal number of $\mathcal{A}$	368

First-order Logic

Syntax

Symbol	Meaning	p.
$\mathcal{L}$	(first-order) language	371
$\widehat{\mathcal{L}}$	effectivised (first-order) language	380
$Sym_{\mathcal{L}}$	set of symbols of a language	372
$Sym^*_{\mathcal{L}}$	set of proper symbols of a language	372
$Fml_{\mathcal{L}}$	set of formulae of a language	372
$Stn_{\mathcal{L}}$	set of sentences of a language	374
$Trm_{\mathcal{L}}$	set of terms of a language	372
$Exp_{\mathcal{L}}$	set of expressions of a language	373
$\neq$	not equal	382
$=$	equality predicate	372
$\exists$	existential quantifier	371
$\forall$	universal quantifier	371
$(v)\phi$	universal quantifier	373
$\exists\vert$	unitary existential quantifier	371
$\wedge$	conjunction	371
$\vee$	disjunction	371
$\neg$	negation	371
$\supset$	conditional	371
$\equiv$	biconditional	371
$\rightleftharpoons$	equality or biconditional	213
$\bot$	contradiction	371
$\top$	tautology	371
$T_0^{\mathcal{L}}$	predicate calculus for $\mathcal{L}$	381
$[\phi]$	universal closure	374
$[[\phi]]$	existential closure	374
$\stackrel{s}{=}$	syntactic identify	375
$\Gamma \rightarrow \phi$	sequent	439
$FrVar(\phi)$	list of free-variables	373
$\ulcorner\epsilon\urcorner$	Gödel number	380
Δ_T	axiom set for a theory T	381
&	metatheoretical conjunction	387

Substitutions & Replacements

Symbol	Meaning	p.
$\epsilon\{v_1, \ldots, v_n\}$	list of free-variables	373
$\epsilon(v_1, \ldots, v_n)$	list of free-variables	373
$\epsilon[v_1, \ldots, v_n]$	list of free-variables	373
$\epsilon[\![\delta_1, \ldots, \delta_n]\!]$	list of subexpressions	375
$\epsilon[\![\langle \delta_1, p_1 \rangle, \ldots, \langle \delta_n, p_n \rangle]\!]$	list of subexpressions	375
$\epsilon[\![S_1, \ldots, S_n]\!]$	list of occurring symbols	375
$\epsilon[\![\langle S_1, p_1 \rangle, \ldots \langle S_n, p_n \rangle]\!]$	list of occurring symbols	375
$\epsilon[\not{p}]$	non-occurring (free) variable	376
$\epsilon[\![\not{S}]\!]$	non-occurring symbol	376
$\epsilon[\![\not{\delta}]\!]$	non-occurring expression	376
$\epsilon[\![^{S'_1}/_{S_1}, \ldots, ^{S'_n}/_{S_n}]\!]$	replacement of symbols	376
$\epsilon[\![\langle S'_1, S_1, p_1 \rangle, \ldots, \langle S'_n, S_n, p_n \rangle]\!]$	replacement of symbols	376
$\epsilon[\![^{\delta'_1}/_{\delta_1}, \ldots, ^{\delta'_n}/_{\delta_n}]\!]$	replacement of subexpressions	376
$\epsilon[\![\langle \delta'_1, \delta_1, p_1 \rangle, \ldots, \langle \delta'_n, \delta_n, p_n \rangle]\!]$	replacement of subexpressions	376
$\{^{t_1}/_{v_1}, \ldots, ^{t_n}/_{v_n}\}$	variable substitution	374

Semantics

Symbol	Meaning	p.
$\lvert\mathfrak{A}\rvert$	universe of a structure	377
$\models \phi$	valid formula	379
$\mathfrak{A} \models \phi$	model of a formula	379
$\mathfrak{A} \models \phi[\alpha]$	variable assignment in a structure	378
$\Gamma \models \phi$	Γ-validity (or logical consequence)	379
$sat_{\mathfrak{A}}(\phi)$	satisfiability of a formula in a structure	378
$sat(\phi)$	satisfiability of a formula	378
$sat_{\mathfrak{A}}(\Gamma)$	satisfiability of a set of formulae in a structure	379
$sat(\Gamma)$	satisfiability of a set of formulae	379
$T_{\mathfrak{A}}$	theory of a structure	381
$\mathfrak{N}_A$	$\langle \mathbb{N}, 0, 1, \mathrm{s}, + \rangle$	397
$\mathfrak{N}_M$	$\langle \mathbb{N}, 0, 1, \times \rangle$	377
$\mathfrak{N}_E$	$\langle \mathbb{N}, 0, \mathrm{s}, +, <, \equiv_2, \ldots, \equiv_n, \ldots \rangle$	394
$\{^{b_1}/_{v_1}, \ldots, ^{b_n}/_{v_n}\}$	variable assignment	378

Symbol	Meaning	p.
SMA_0	Basic Strictly Multiplicative Arithmetic	385
SMA	Strictly Multiplicative Arithmetic	398
PA_0	Basic Peano Arithmetic	382
PA	Peano Arithmetic	397
PA^*	Extended Peano Arithmetic	262
PrA_0	Basic Presburger Arithmetic	382
PrA	Presburger Arithmetic	398
PrA^*	Extended Presburger Arithmetic	257
LA	Linear Arithmetic	64
DAG	Densely Ordered Abelian Groups	372
DAG^*	Extended Densely Ordered Abelian Groups	65
$\mathcal{L}_{PA_0}$	$\{0, \mathrm{s}, +, \times\}$	372
$\mathcal{L}_{PA}$	$\{0, 1, \mathrm{s}, +, \times\}$	372
$\mathcal{L}_{PA^*}$	$\{0, 1, \mathrm{s}, +, \times, -, \mathrm{pr}, \exp, /, \mathrm{double}, \mathrm{Sum}_n, \mathrm{half}, \gcd, \mathrm{gfc}, \mathrm{rmdr}, \min_n, \max_n, \leq, >, \geq, \mid, \mathrm{even}, \mathrm{prime}, \mathrm{prime}_1, \equiv_n\}$	262
$\mathcal{L}_{PA^{**}}$		266
$\mathcal{L}_{PrA_0}$	$\{0, \mathrm{s}, +\}$	372
$\mathcal{L}_{PrA}$	$\{0, 1, \mathrm{s}, +\}$	372
$\mathcal{L}_{PrA^*}$	$\{0, 1, \mathrm{s}, +, <\}$	257
$\mathcal{L}_{SMA}$	$\{0, 1, \times\}$	372
$\mathcal{L}_{DAG}$	$\{0, 1, +, <\}$	372
$\mathcal{L}_{DAG^*}$	$\{0, 1, +, -, <, \leq\}$	65

Miscellaneous

Symbol	Meaning	p.
$R.\ \delta_1 \Rightarrow \delta_2$	rewrite rule	426
$R.\ \phi \rightarrow \delta_1 \Rightarrow \delta_2$	conditional rewrite rule	427
$\epsilon_1 \overset{R}{\Rightarrow} \epsilon_2$	application of a rewrite rule	426
$\epsilon_1 \overset{\mathcal{R}}{\Rightarrow} \epsilon_2$	application of a rewrite rule set	426
iff	if and only if	370
mgu	most general unifier	423
mgpu	most general partial unifier	435
pu	partial unifier	435
$\square$	empty clause	424
rhs	right hand side (expression)	113
lhs	left hand side (expression)	105
m_t	termination measure for total *remove* rules	167
m_e	measure function for rewrite rules	179
m_s	measure function for symbols	179
m_d	measure function for deviant symbols	180
m_c	measure function for decidable classes	180
m_r	measure function for *remove* rules	184
$m_{r^\bullet}$	measure function for *remove* rules (in the presence of disagreement elimination)	228
$\mathcal{R}_{PA^\bullet}$	arithmetical rule base	268
$\mathcal{E}_{PA^\bullet}$	arithmetical equality base	270
v.c.	verification condition	73

Arithmetic

Symbol	Meaning	p.
$\dot{+}$	sum	382
$\dot{-}$	subtraction	395
$-$	arithmetical subtraction	265
$\times$	multiplication	382
$/$	arithmetical division	265
s	successor	369
exp	exponentiation	265
gcd	greatest common divider	265
prime	prime natural number	265
prime_1		265
$<$	less than	265
$>$	greater than	265
$\leq$	less than or equal to	265
$\geq$	greater than or equal to	265
pr	predecessor	265
$\equiv_n$	equivalence modulo n	265
Sum_n	sum of a n-tuple	265
$\mid$	divisibility	265
even	even natural number	265
double	double	265
rmdr	remainder	265
gfc	greatest factor	265
half	arithmetical half	265
$\not<$	not less than	262
$\min_n$	minimum element of n-tuple	265
$\max_n$	maximum element of n-tuple	265

Metatheoretical many-sorted alphabet

Metavariables

S	symbols (of a first-order language)
v, w	individual variables
t, u	terms
γ, ℓ	literals
ϕ, ψ	formulae
τ, ρ	sentences
δ, ϵ	expressions
C, D	clauses
E	equations
R	rewrite rules
σ	substitutions
α	variable assignments
$\mathcal{N}$	normaliser
CF	control function

$\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$	sets
$\mathcal{U}$	universe sets
$\mathcal{S}$	set of symbols
$\mathcal{L}$	set of logical symbols (of $\mathcal{L}$)
$\mathcal{C}$	set of individual constants (of $\mathcal{L}$)
$\mathcal{F}$	set of function symbols (of $\mathcal{L}$)
$\mathcal{P}$	set of predicate symbols (of $\mathcal{L}$)
$\mathcal{V}$	set of individual variables (of $\mathcal{L}$)
$\Sigma, \Gamma, \Delta, \Phi$	sets of formulae
$\mathcal{E}$	set of expressions (equations)
Σ	(decidable) subclasses of formulae
Λ	set of sequents
Ω	set of clauses
$\mathcal{D}$	disagreement sets
$\mathcal{R}$	sets of rewrite rules

Appendix B

The Decision Problem for First-Order Theories

Decision procedures can be informally described as computable mechanisms that process finite amounts of information, generate results and halt after a finite amount of time and computation. In the context of a formal theory, a decision procedure has to establish, for each formula of the underlying language, whether it is a theorem or not. Theories which admit decision procedures for the recognition of theorems are *decidable*.

Basic notions related to the decision problem for first-order theories are introduced in section B.1, including recursive functions (upon which recursive computability is built) and first-order languages. Section B.2 defines formal theories and presents two classes of decision problems, one related to the recognition of *valid*, and the other to the recognition of *satisfiable* formulae. Section B.3 examines two methodologies for establishing the decidability of a theory, *quantifier elimination* and *model completeness*, and includes a proof for the decidability of Presburger arithmetic based on elimination of quantifiers.

B.1 Preliminary Concepts

The informal definition of decision procedures as *effectively computable* mechanisms is too imprecise to receive mathematical treatment, which is particularly relevant when it comes to the establishment of the undecidability of a problem: proving the nonexistence of a decision procedure requires determining exactly what such a procedure is in the first place. The notion of *recursive decidability*, built upon the theory of recursive functions, provides one of the possible mathematical models for this informal concept.

When a decision problem involves the recognition of theorems, informal theories themselves have to be replaced by a more rigorous concept. Processes such as formula construction or deduction then become objects of mathematical treatment as well. In the present context, first-order languages and theories are suitable substitutes for their

informal counterparts[1].

Concerning the notation for recursive functions and sets, $f\colon \mathcal{A} \rightarrow \mathcal{B}$ and $g\colon \mathcal{A} \rightsquigarrow \mathcal{B}$ respectively denote a total and a partial function with domain $\mathcal{A}$ and range $\mathcal{B}$. A function g is *partial* in a domain $\mathcal{A}$ when there exists an element $a \in \mathcal{A}$ such that $g(a)$ may not be defined: as a result, every total function is also partial. $dom(f), rng(f)$ and $im(f)$ respectively denote the domain, range and image of a function f. $f(\mathcal{A})$ represents the set $\{f(a) \mid a \in \mathcal{A}\}$, and $f \circ g$ represents the composition of f and g, such that $(f \circ g)(n) = f(g(n))$. Given a function $f\colon \mathcal{A} \rightarrow \mathcal{B}$ and a subset $\mathcal{A}' \subset \mathcal{A}$, $f \restriction \mathcal{A}'$ denotes the restriction of f to $\mathcal{A}'$. Finally, $\#\mathcal{A}$ stands for the cardinal of a set $\mathcal{A}$, and $\wp(\mathcal{A})$ is the set of all subsets of $\mathcal{A}$.

B.1.1 Recursive Functions

A (partial) function f is *computable* in a domain $\mathcal{D}$ if and only if there is an algorithm which, given $d \in \mathcal{D}$, either generates $f(d)$ after a finite amount of time and computation, when f is defined for d, or may not halt otherwise. An algorithm is a set of instructions that satisfy four properties:

(i) it must be fixed, and cannot be modified or adjusted to the input data,
(ii) it must be finite,
(iii) it must be ordered (although the order may be context-sensitive), and
(iv) the instructions are unambiguous, and invariably yield the same results for the same input.

The input data supplied at a particular instant must also be finite. If the computation terminates for every element of $\mathcal{D}$, f is *effectively computable* in $\mathcal{D}$.

There are several mathematical formalisms for the representation of computable functions: Turing machines, Herbrand-Gödel-Kleene computable functions, Markov algorithms and partial recursive functions, among others. The classes of functions defined by these formalisms are essentially equivalent[2]. Nonetheless, recursive functions are more frequently used in the analysis of decision problems for first-order theories. Partial recursive functions are generated from an initial group of elementary functions by the application of certain operations, which include definition by recursion.

Definition B.1.1 (Generation of functions)

Let $\mathbb{N}$ *be the set of natural numbers.*

i. Given the functions $f\colon \mathbb{N}^m \rightarrow \mathbb{N}$ *and* $g_i\colon \mathbb{N}^n \rightarrow \mathbb{N}$, $1 \leq i \leq m$, *the function* $h\colon \mathbb{N}^n \rightarrow \mathbb{N}$, *defined as*

$$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$$

is derived from $f, g_1, \ldots, g_m$ *by* substitution.

---

[1] This section has been mainly based on [Monk 76], parts I and II, and [Mendelson 87], chapter 5. Other texts consulted include [Gallier 87] and [Malitz 79].

[2] See [Monk 76], p. 12.

ii. Given the functions $f\colon \mathbb{N}^n \to \mathbb{N}$ *and* $g\colon \mathbb{N}^{n+2} \to \mathbb{N}$, *the function* $h\colon \mathbb{N}^{n+1} \to \mathbb{N}$, *defined as*

$$\begin{aligned} h(x_1, \ldots, x_n, 0) &= f(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n, s(y)) &= g(x_1, \ldots, x_n, y, h(x_1, \ldots, x_n, y)) \end{aligned}$$

is derived from f *and* g *by* recursion.

Total recursive functions provide a mathematical representation for *effective computability*, since their computation terminates for every n-tuple of natural numbers. Amongst total functions, the primitive recursive class contains all those which have an 'acceptable' degree of complexity[3].

Definition B.1.2 (Total recursive functions)

i. A function is primitive recursive *if and only if it belongs to the class defined as follows.*

(a) The constant function $z\colon \mathbb{N} \to \{0\}$, *the successor function* $s\colon \mathbb{N} \to \mathbb{N}^*$ *and the projection functions* $q_n^i : \mathbb{N}^n \to \mathbb{N}$, $1 \leq i \leq n$, $n \in \mathbb{N}^*$, *which satisfy the properties*

$$\begin{aligned} z(x) &= 0 \\ s(x) &= x + 1 \\ q_n^i(x_1, \ldots, x_n) &= x_i \qquad (1 \leq i \leq n) \end{aligned}$$

are primitive recursive.

(b) If $f\colon \mathbb{N}^m \to \mathbb{N}$ *and* $g_i\colon \mathbb{N}^n \to \mathbb{N}$, $1 \leq i \leq m$, *are primitive recursive, the function* $h\colon \mathbb{N}^n \to \mathbb{N}$ *derived from* $f, g_1, \ldots, g_m$ *by substitution is also primitive recursive.*

(c) If $f : \mathbb{N}^n \to \mathbb{N}$ *and* $g : \mathbb{N}^{n+2} \to \mathbb{N}$ *are primitive recursive, the function* $h\colon \mathbb{N}^{n+1} \to \mathbb{N}$ *derived from* f *and* g *by recursion is also primitive recursive.*

(d) Only the functions defined above are primitive recursive.

ii. A function is total recursive *if and only if it belongs to the class defined as follows.*

(a) Every primitive recursive function is total recursive.

(b) If $f\colon \mathbb{N}^m \to \mathbb{N}$ *and* $g_i\colon \mathbb{N}^n \to \mathbb{N}$, $1 \leq i \leq m$, *are total recursive, the function* $h\colon \mathbb{N}^n \to \mathbb{N}$ *derived from* $f, g_1, \ldots, g_m$ *by substitution is also total recursive.*

(c) If $f\colon \mathbb{N}^n \to \mathbb{N}$ *and* $g\colon \mathbb{N}^{n+2} \to \mathbb{N}$ *are total recursive, the function* $h\colon \mathbb{N}^{n+1} \to \mathbb{N}$, *derived from* f *and* g *by recursion, is also total recursive.*

(d) If $f\colon \mathbb{N}^{n+1} \to \mathbb{N}$ *is total recursive, the function* $g\colon \mathbb{N}^n \to \mathbb{N}$ *that satisfies the condition*

$$g(x_1, \ldots, x_n) = \mu y(f(x_1, \ldots, x_n, y) = 0),$$

[3] See [Gallier 87], p. 370-1.

where $\mu x_i(R(x_1, \ldots, x_i, \ldots, x_m)) = \min\{x_i \in \mathbb{N} \mid R(x_1, \ldots, x_i, \ldots, x_m)\}$, *is total recursive, provided that*

$$(x_1)\ldots(x_n)(\exists y)(f(x_1, \ldots, x_n, y) = 0)$$

is arithmetically valid.

(e) Only the functions defined above are total recursive.

The final extension of the class of recursively computable functions includes those which may not be defined for all the elements of their respective domains. Partial functions are obtained when the μ-operator is used without the restrictions originally present in the definition of total functions.

Definition B.1.3 (Partial Recursive Functions)

A function is partial recursive *if and only if it belongs to the class defined below.*

i. Every total recursive function is partial recursive.

ii. If $f\colon \mathbb{N}^m \rightharpoonup \mathbb{N}$ *and* $g_i\colon \mathbb{N}^n \rightharpoonup \mathbb{N}$, $1 \leq i \leq m$, *are partial recursive, the function* $h\colon \mathbb{N}^n \rightharpoonup \mathbb{N}$ *derived from* $f, g_1, \ldots, g_m$ *by substitution is also partial recursive.*

iii. If $f\colon \mathbb{N}^n \rightharpoonup \mathbb{N}$ *and* $g\colon \mathbb{N}^{n+2} \rightharpoonup \mathbb{N}$ *are partial recursive, the function* $h\colon \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$, *derived from* f *and* g *by recursion, is also partial recursive.*

iv. If $f\colon \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ *is partial recursive, the function* $g\colon \mathbb{N}^n \rightharpoonup \mathbb{N}$, *defined as*

$$g(x_1, \ldots, x_n) \quad = \quad \mu y(f(x_1, \ldots, x_n, y) = 0)$$

is partial recursive.

v. Only the functions defined above are partial recursive.

Partial recursive functions satisfy the informal criteria that define computable functions, whereas total recursive functions meet the more strict conditions set for effectively computable functions. Hence, recursiveness provides a sound representation for informal computability. To avoid discussions about the completeness of the recursive representation (i.e. whether every (effectively) computable function has a (total) partial recursive counterpart), *effectively computable* and *computable* can be respectively replaced by *total* and *partial* recursive in all domains restricted to natural numbers[4].

Recursive functions can be employed in the identification of *recursive* and *semi-recursive sets* of natural numbers. A subset $\mathcal{A} \subseteq \mathbb{N}$ is *recursive* iff there exists a total recursive function $f\colon \mathbb{N} \to \{0, 1\}$ such that

[4] The completeness of the recursive representation follows from Church's thesis, which states that *a number-theoretic function is effectively computable if and only if the function is (total) recursive*: see for instance [Mendelson 87], p. 165.

$$(\forall x \in \mathbb{N})(x \in \mathcal{A} \text{ iff } f(x) = 1 \text{ and } x \notin \mathcal{A} \text{ iff } f(x) = 0)$$

f can be schematically represented as

$$f(x) = \begin{cases} 1 & x \in \mathcal{A} \\ 0 & \text{otherwise} \end{cases}$$

A subset $\mathcal{A} \subseteq \mathbb{N}$ is *semi-recursive* (or *recursively enumerable*) iff there exists a partial recursive function $f\colon \mathbb{N} \rightharpoonup \{0,1\}$ such that

$$f(x) = \begin{cases} 1 & x \in \mathcal{A} \\ 0 \text{ or undefined} & \text{otherwise} \end{cases}$$

Any recursive set is also recursively enumerable. The membership relation is computable for recursively enumerable sets, and effectively computable for recursive sets: it is possible to determine in a finite amount of time whether a particular natural number belongs to a recursive subset of $\mathbb{N}$ or not.

B.1.2 Syntax of First-Order Languages

The construction of a mechanism for the recognition of theorems of a particular theory starts with the definition of the expressions of the language of the theory. Symbols, expressions and sequences of expressions of certain formal languages — first-order languages amongst them — can then be mapped into $\mathbb{N}$. Processes in these and other arithmetisable domains become thereafter representable in terms of recursive functions.

Definition B.1.4 (First-order languages)

i. *A* pure first-order language *is a quintuple* $\mathcal{L} = \langle \ell, \mathcal{V}, \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$ *where* $\ell, \mathcal{V}, \mathcal{C}, \mathcal{F}$ *and* $\mathcal{P}$ *are pairwise disjoint countable sets such that*

(a) $\ell = \{\top, \bot, \neg, \wedge, \vee, \supset, \equiv, \forall, \exists\}$ *is the set of* logical symbols[5] of $\mathcal{L}$.

(b) $\mathcal{V}$ *is the infinite set of* individual variables of $\mathcal{L}$.

(c) $\mathcal{C} \subseteq \{c_1, \ldots, c_n, \ldots\}, n \in \mathbb{N}$, *is the set of* individual constants of $\mathcal{L}$.

(d) $\mathcal{F} \subseteq \mathcal{F}_1 \cup \cdots \cup \mathcal{F}_m \cup \ldots, m \in \mathbb{N}$, *is the set of* function symbols of $\mathcal{L}$. *Each* $\mathcal{F}_i \subseteq \{f_{1,i}, \ldots, f_{j,i}, \ldots\}, j \in \mathbb{N}$ *is the* subset of function symbols of arity i.

(e) $\mathcal{P} \subseteq \mathcal{P}_1 \cup \cdots \cup \mathcal{P}_q \cup \ldots, q \in \mathbb{N}$, *is the set of* predicate symbols of $\mathcal{L}$. *Each* $\mathcal{P}_i \subseteq \{p_{1,i}, \ldots, p_{j,i}, \ldots\}, j \in \mathbb{N}$ *is the* subset of predicate symbols of arity i.

---

[5] An additional symbol, $\exists|$, which represents the existence of a single object that satisfies a property, may be useful in certain contexts. The expression $(\exists|v)\phi[v]$ can be also regarded as an abbreviation for

$$(\exists v_1)\phi[v_1] \wedge (v_2)(v_3)((\phi[^{v_2}/_{v_1}] \wedge \phi[^{v_3}/_{v_1}]) \supset v_2 = v_3)$$

ii. *A* first-order language with equality *(or* first-order language *for short) is a pure first-order language to whose set of logical symbols the symbol for equality,* $(=)$*, has been added.*

iii. *The set of* symbols of $\mathcal{L}$ *is defined as*

$$Sym_{\mathcal{L}} = \ell \cup \mathcal{V} \cup \mathcal{C} \cup \mathcal{F} \cup \mathcal{P},$$

while $Sym^{*}_{\mathcal{L}} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$ (or $\langle \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$) *represents the* non-logical symbols *(i.e. individual constant, function and predicate symbols) of* $\mathcal{L}$.

iv. *If* $\mathcal{L}' = \langle \ell, \mathcal{V}, \mathcal{C}', \mathcal{F}', \mathcal{P}' \rangle$ *and* $\mathcal{L} = \langle \ell, \mathcal{V}, \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$ *be first-order languages,* $\mathcal{L}'$ *is an* expansion *of* $\mathcal{L}$ *iff* $\mathcal{C} \subseteq \mathcal{C}', \mathcal{F} \subseteq \mathcal{F}'$ *and* $\mathcal{P} \subseteq \mathcal{P}'$.

Basic Peano arithmetic (PA_0), (first-order) Peano arithmetic (PA), basic Presburger arithmetic (PrA_0), Presburger arithmetic (PrA), strictly multiplicative arithmetic (SMA) and the theory of dense ordered Abelian groups without endpoints (DAG) are some of the theories to be examined in this appendix and in the main text. Since they can all be formulated in languages that share the same set of logical symbols and variables, their languages are unequivocally defined by means of their non-logical symbols.

$$\begin{aligned}
\mathcal{L}_{PA_0} &= \{0, \mathrm{s}, +, \times\} \\
\mathcal{L}_{PA} &= \{0, 1, \mathrm{s}, +, \times\} \\
\mathcal{L}_{PrA_0} &= \{0, \mathrm{s}, +\} \\
\mathcal{L}_{PrA} &= \{0, 1, \mathrm{s}, +\} \\
\mathcal{L}_{SMA} &= \{0, 1, \times\} \\
\mathcal{L}_{DAG} &= \{0, 1, +, <\}
\end{aligned}$$

Symbols of a first-order language can be combined to form expressions. Sequences of symbols that satisfy certain restrictions are called *well-formed expressions*, which include *terms* and *formulae.*

Definition B.1.5 (Expressions of a first-order language)

i. *The set of* (well formed) terms $Trm_{\mathcal{L}}$ *of a first-order language* $\mathcal{L}$ *is recursively defined as follows.*

(a) *An individual variable or an individual constant is a term (called* atomic*).*

(b) *If* $f_{i,j}$ *is a function symbol of arity* j *in* $\mathcal{L}$*, and* $t_1, \ldots, t_j$ *are terms, then* $f_{i,j}(t_1, \ldots, t_j)$ *is a term (called* composite*).*

(c) *Only the expressions defined above are terms of* $\mathcal{L}$.

ii. *The set of* (well formed) formulae $Fml_{\mathcal{L}}$ *of a first-order language* $\mathcal{L}$ *is recursively defined as follows.*

(a) $\top$ *and* $\bot$ *are (atomic) formulae of* $\mathcal{L}$.

(b) *If* $p_{i,j}$ *is a predicate symbol of arity* j *in* $\mathcal{L}$, *and* $t_1, \ldots, t_j$ *are terms, then* $p_{i,j}(t_1, \ldots, t_j)$ *is an (atomic) formula of* $\mathcal{L}$. *In particular, an atom of the form* $t_1{=}t_2$ *is also called an* equation *or* equality.

(c) *If* ϕ, ψ *are formulae of* $\mathcal{L}$ *and* v *is an individual variable, then*

$$\neg\phi,\ \ \phi \wedge \psi,\ \ \phi \vee \psi,\ \ \phi \supset \psi,\ \ \phi \equiv \psi,\ \ (\forall v)\phi \text{ [or } (v)\phi],\ \ (\exists v)\phi$$

are formulae of $\mathcal{L}$ *(called* composite*).*

(d) *Only the expressions defined above are formulae of* $\mathcal{L}$.

iii. The set $Exp_{\mathcal{L}}$ *of* (well formed) expressions *of* $\mathcal{L}$ *is defined as*

$$Exp_{\mathcal{L}} = Fml_{\mathcal{L}} \cup Trm_{\mathcal{L}}$$

$Exp_{\mathcal{L}}$ can be partitioned into disjoint subsets, called *syntactic sorts* or *types*. The set of formulae and the set of terms constitute syntactic types for any set of expressions of a first-order language. Atomic and composite terms are syntactic subtypes of $Trm_{\mathcal{L}}$, whereas atoms and composite formulae are syntactic subtypes of $Fml_{\mathcal{L}}$. Subclasses of terms and formulae may be defined by means of syntactic restrictions imposed on their structure, e.g. type of logical connective and/or their relative positions of occurrences.

Definition B.1.6 (Boolean Combination)

Let $\Gamma = \{\phi_1, \ldots, \phi_n, \ldots\}$ *be a set of formulae of a language* $\mathcal{L}$. *A* Boolean combination of formulae of Γ *is any formula* ψ *that satisfies one of the conditions.*

i. ψ *is an element of* Γ.

ii. ψ *has any of the forms* $(\psi_1 \wedge \psi_2)$, $(\psi_1 \vee \psi_2)$ *or* $(\neg\psi_1)$, *where* ψ_1 *and* ψ_2 *are Boolean combinations of formulae of* Γ.

B.1.3 Variable Substitution

An occurrence of a variable v in a formula ϕ is *free* if it does not take place in the scope of a quantifier $(\forall v)$ or $(\exists v)$, and is *bound* otherwise. A formula that has free occurrences of variables is *open*, whereas if all occurrences of variables are bound, it is *closed*. If $FrVar(\phi)$ denotes the set of all variables that have a free occurrence in a formula ϕ, the notation

$$\phi[v_1, \ldots, v_n]$$

indicates that $\{v_1, \ldots, v_n\}$ is a subset of the set of variables that have free occurrences in ϕ (i.e. $\{v_1, \ldots, v_n\} \subseteq FrVar(\phi)$), while in

$$\phi(v_1, \ldots, v_n)$$

$\{v_1, \ldots, v_n\}$ corresponds to all the variables that occur free in ϕ (i.e. $FrVar(\phi) = \{v_1, \ldots, v_n\}$). Finally,

$$\phi\{v_1, \ldots, v_n\}$$

indicates that the free variables of ϕ are amongst those in $\{v_1, \ldots, v_n\}$ (i.e. $FrVar(\phi) \subseteq \{v_1, \ldots, v_n\}$). A formula whose set of free variables is empty is a *sentence*. The set of sentences of a language $\mathcal{L}$ is denoted by $Stn_{\mathcal{L}}$. Subsets of quantified formulae can be delimited based on the position of occurrence of quantifiers in the formula, as well as on the pattern of such occurrences.

Definition B.1.7 (Universal & Existential Closures)

Let $\phi(v_1, \ldots, v_n)$ be a formula in a first-order language $\mathcal{L}$.

i. ϕ *is in* prenex form *iff it has the form*

$$(Q_1 v_1) \ldots (Q_m v_m)\psi$$

where Q_i is either a universal or existential quantifier, and ψ is a quantifier-free formula. $(Q_1 v_1) \ldots (Q_m v_m)$ is the prefix *of ϕ, and ψ is its* matrix.

ii. ϕ *is* universal *iff it is in prenex form and its prefix is composed entirely of universal quantifiers; similarly, it is* existential *iff only existential quantifiers occur in its prefix.*

iii. The universal closure *of ϕ is the sentence $(v_1) \ldots (v_n)\phi(v_1, \ldots, v_n)$, usually denoted by $[\phi]$.*

iv. The existential closure *of ϕ is the sentence $(\exists v_1) \ldots (\exists v_n)\phi(v_1, \ldots, v_n)$, represented as $[[\phi]]$.*

New expressions can be syntactically derived from open expressions by the application of *substitutions*. A substitution for free variables of an expression $\epsilon[v_1, \ldots, v_n]$ in $\mathcal{L}$ is a function σ: $V \rightarrow Trm_{\mathcal{L}}$, where $\{v_1, \ldots, v_n\} \subseteq V$, and, if $\sigma(v_i) = t_i$, t_i does not contain any of the variables in $\{v_1, \ldots, v_n\}$. A finite substitution is extensionally represented as

$$\sigma = \{^{t_1}/_{v_1}, \ldots, ^{t_n}/_{v_n}\}$$

The application of σ to an expression $\epsilon[v_1, \ldots, v_n]$ generates a *(substitution) instance* of ϵ, and is denoted as $\sigma\epsilon[v_1, \ldots, v_n]$ or $\epsilon[^{t_1}/_{v_1}, \ldots, ^{t_n}/_{v_n}]$. When the names of the variables being substituted for are not relevant, the resulting expression is simply represented as $\epsilon[t_1, \ldots, t_n]$. Similar notation applies to expressions of the forms $\epsilon(v_1, \ldots, v_n)$ and $\epsilon\{v_1, \ldots, v_n\}$.

New expressions can be also generated by means of the *replacement* of subexpressions or symbols. An occurrence of a subexpression can be singled out once the position of occurrence of its dominant symbol has been identified.

Definition B.1.8 (Occurrences of Symbols and Expressions)

Let ϵ be an expression of a language $\mathcal{L}$.

i. The dominant symbol S^D *of ϵ is defined as follows*[6].

(a) If ϵ is an atomic term or a propositional constant, then $S^D \stackrel{s}{=} \epsilon$.

(b) If ϵ is a composite expression of the form $S(\epsilon_1, \ldots, \epsilon_n)$, then $S^D \stackrel{s}{=} S$.

(c) If ϵ is a formula of the form $(Qv)\phi$, where Q is a quantifier, then $S^D \stackrel{s}{=} Qv$.

ii. A symbol S has an occurrence at position p in ϵ, i.e.

$$\epsilon[\![\langle S, p\rangle]\!]$$

where p is a finite list of natural numbers, iff

(a) ϵ is an individual variable, individual constant or propositional constant, $S \stackrel{s}{=} \epsilon$ and $p = [\,]$, or

(b) ϵ has the form $S(\epsilon_1, \ldots, \epsilon_n)$ and $p = [\,]$, or

(c) ϵ has the form $S'(\epsilon_1, \ldots, \epsilon_i, \ldots, \epsilon_m)$, S has an occurrence at position p' in ϵ_i, and $p = [i|p']$.

When the position of occurrence of S in ϵ is immaterial, the above notation can be simplified to $\epsilon[\![S]\!]$.

iii. An expression δ is a subexpression *of ϵ, i.e. $\epsilon[\![\delta]\!]$ iff*

(a) $\delta \stackrel{s}{=} \epsilon$, *or*
(b) $\epsilon \stackrel{s}{=} S(\epsilon_1, \ldots, \epsilon_n)$ *and δ is a subexpression of ϵ_i, for some i, $1 \leq i \leq n$, or*
(c) $\epsilon \stackrel{s}{=} (Qv)\phi$, *where Q is a quantifier, and δ is a subexpression of ϕ.*

$\epsilon[\![\delta_1, \ldots, \delta_m]\!]$ indicates that $\delta_1, \ldots, \delta_m$ are subexpressions of ϵ.

iv. An expression δ dominated by a symbol S has an occurrence at position p in ϵ, i.e. $\epsilon[\![\langle \delta, p\rangle]\!]$ iff

(a) *S occurs at position p in ϵ, and*
(b) *the occurrence of S at p dominates a subexpression of ϵ that is syntactically identical to δ.*

If the expressions $\delta_1, \ldots, \delta_m$ have occurrences in an expression ϵ respectively at positions $p_1, \ldots, p_m$, this fact can be denoted as $\epsilon[\![\langle \delta_1, p_1\rangle, \ldots, \langle \delta_m, p_m\rangle]\!]$.

The notation introduced so far covers four distinct contexts involving variables and expressions:

(i) $\epsilon[\![v]\!]$ denotes that there is an occurrence (free or bound) of v in ϵ,
(ii) $\epsilon[v]$ indicates that v has a free occurrence in ϵ,
(iii) $\epsilon(v)$ means that v is the only variable that occurs free in ϵ, and

[6] The symbol $\stackrel{s}{=}$ denotes syntactic equality.

(iv) $\epsilon\{v\}$ means that v may occur free in ϵ.

Finally, the additional metatheoretical expressions

$$\epsilon[\not v] \qquad \epsilon[\![\not S]\!] \qquad \epsilon[\![\not \delta]\!]$$

respectively indicate that

(i) v does not have any free occurrence in ϵ,
(ii) the symbol S does not occur in ϵ, and
(iii) the expression δ does not occur in ϵ.

An expression ϵ' can be generated from $\epsilon[\![S]\!]$ through the replacement of each occurrence of S with a symbol S', provided that S and S' have the same syntactic type and arity, in which case the resulting expression has the form $\epsilon[\![^{S'}/_{S}]\!]$. The replacement can be also restricted to a particular occurrence of S at position p in ϵ, i.e. $\epsilon[\![\langle S', S, p\rangle]\!]$. When multiple replacements take place, they are either exhaustive,

$$\epsilon[\![^{S'_1}/_{S_1}, \ldots, {}^{S'_n}/_{S_n}]\!]$$

or restricted to specific occurrences of symbols, as in $\epsilon[\![\langle S'_1, S_1, p_1\rangle, \ldots, \langle S'_n, S_n, p_n\rangle]\!]$. The same syntactic operations may be extended to the domain of subexpressions:

$$\epsilon[\![^{\delta'_1}/_{\delta_1}, \ldots, {}^{\delta'_n}/_{\delta_n}]\!]$$

represents the exhaustive replacement of subexpressions $\delta_1, \ldots, \delta_n$ with $\delta'_1, \ldots, \delta'_n$, whereas in $\epsilon[\![\langle \delta'_1, \delta_1, p_1\rangle, \ldots, \langle \delta'_n, \delta_n, p_n\rangle]\!]$ particular occurrences of subexpressions are selected before replacements take place.

B.1.4 Semantics of First-Order Languages

Model-theoretic properties of theories are commonly invoked in the establishment of their decidability. The semantics of first-order languages concerns their relationship with algebraic structures. Once structures for first-order languages have been defined, meaning can be attributed to their expressions[7].

Definition B.1.9 (Structures for first-order languages)

i. A structure for a first-order language $\mathcal{L} = \langle \mathcal{C}, \mathcal{F}, \mathcal{P}\rangle$ is a quadruple $\mathfrak{A} = \langle A, \boldsymbol{C}, \boldsymbol{F}, \boldsymbol{P}\rangle$ where $A \neq \emptyset$ is the domain (or universe*) of the structure, and $\boldsymbol{C}$,$\boldsymbol{F}$,$\boldsymbol{P}$ are functions such that $dom(\boldsymbol{C}) = \mathcal{C}, dom(\boldsymbol{F}) = \mathcal{F}, dom(\boldsymbol{P}) = \mathcal{P}$, and*

[7] Model theoretical concepts have been given precedence over proof theoretical notions, since studies on decidability are mainly concerned with identifying valid formulae in a theory rather than with the exhibition of proofs. According to Church,

> "The writer once proposed the name 'deducibility problem' for what is here called the decision problem for provability, the intention being to reserve the name 'decision problem' either for the semantical decision problem or for what is called (...) the *decision problem for validity.*" ([Church 56], p. 100)

(a) $\boldsymbol{C}(c_i) \in A$.

(b) $\boldsymbol{F}(f_{i,j}) \in A^{A^j}$, *where* j *is the arity of* $f_{i,j}$.

(c) $\boldsymbol{P}(p_{i,j}) \subseteq A^j$, *where* j *is the arity of* $p_{i,j}$.

$\boldsymbol{C}(c_i)$, $\boldsymbol{F}(f_{i,j})$ *and* $\boldsymbol{P}(p_{i,j})$ *are respectively denoted by* $c_i^{\mathfrak{A}}, f_{i,j}^{\mathfrak{A}}$ *and* $p_{i,j}^{\mathfrak{A}}$, *whereas* A *can be also indicated as* $|\mathfrak{A}|$.

ii. *If* $\mathfrak{A} = \langle A, \boldsymbol{C}, \boldsymbol{F}, \boldsymbol{P} \rangle$ *and* $\mathfrak{B} = \langle B, \boldsymbol{C'}, \boldsymbol{F'}, \boldsymbol{P'} \rangle$ *are structures for* $\mathcal{L}$ *such that*

(a) $B \subseteq A$,

(b) $\boldsymbol{C}(c_i) = \boldsymbol{C'}(c_i)$,

(c) $\boldsymbol{F}(f_{i,j}) \upharpoonright B^j = \boldsymbol{F'}(f_{i,j})$,

(d) $\boldsymbol{P}(p_{i,j}) \cap B^j = \boldsymbol{P'}(p_{i,j})$,

then $\mathfrak{B}$ *is a* substructure *of* $\mathfrak{A}$, *and* $\mathfrak{A}$ *is an* extension *of* $\mathfrak{B}$.

iii. *Let* $\mathcal{L}' = \langle \mathcal{C}', \mathcal{F}', \mathcal{P}' \rangle$ *be an expansion of* $\mathcal{L}$. *A structure* $\mathfrak{A}'$ *for* $\mathcal{L}'$ *is an* expansion *of* $\mathfrak{A}$ *(or* $\mathfrak{A}$ *is the* restriction *of* $\mathfrak{A}'$ *to* $\mathcal{L}$*) iff* (a) $|\mathfrak{A}| = |\mathfrak{A}'|$ *and* (b) *for every non-logical symbol* S *of* $\mathcal{L}$,

$$S^{\mathfrak{A}} = S^{\mathfrak{A}'}$$

Example B.1.1

i. $\mathcal{L}_{SMA} = \{0, 1, \times\}$ *admits the following structures*

$$\mathfrak{N}_M = \langle \mathbb{N}, 0^{\mathfrak{N}_M}, 1^{\mathfrak{N}_M}, \times^{\mathfrak{N}_M} \rangle$$

$$\mathfrak{Q}_M = \langle \mathbb{Q}, 0^{\mathfrak{Q}_M}, 1^{\mathfrak{Q}_M}, \times^{\mathfrak{Q}_M} \rangle$$

$$\mathfrak{R}_M = \langle \mathbb{R}, 0^{\mathfrak{R}_M}, 1^{\mathfrak{R}_M}, \times^{\mathfrak{R}_M} \rangle$$

where $\mathbb{N}, \mathbb{Q}$ *and* $\mathbb{R}$ *are respectively the sets of natural, rational and real numbers.*

ii. $\mathfrak{N}_M$ *is a substructure of both* $\mathfrak{Q}_M$ *and* $\mathfrak{R}_M$, *since*

(a) *they are all structures of the same language,*
(b) $\mathbb{N} \subset \mathbb{Q} \subset \mathbb{R}$,
(c) $0^{\mathfrak{N}_M} = 0^{\mathfrak{Q}_M} = 0^{\mathfrak{R}_M}$, $1^{\mathfrak{N}_M} = 1^{\mathfrak{Q}_M} = 1^{\mathfrak{R}_M}$, *and*
(d) $\times^{\mathfrak{N}_M} \upharpoonright \mathbb{N}^2 = \times^{\mathfrak{Q}_M} \upharpoonright \mathbb{N}^2 = \times^{\mathfrak{N}_M}$. □

The attribution of meaning to terms involves associating individual variables with elements of the universe of a structure. Truth values can be assigned to formulae thereafter.

Definition B.1.10 (Variable Assignments)

i. An assignment *is a function α defined over the set of individual variables of $\mathcal{L}$ into the universe of a structure $\mathfrak{A}$ for $\mathcal{L}$, which is usually represented as*

$$\alpha = \{^{a_1}/_{v_1}, \ldots, ^{a_n}/_{v_n}, \ldots\}$$

The value (or denotation) of a term t of $\mathcal{L}$ in $\mathfrak{A}$ for an assignment α is an element $t^{\mathfrak{A}}[\alpha] \in |\mathfrak{A}|$ defined as follows.

(a) $v_i^{\mathfrak{A}}[\alpha] = \alpha(v_i)$, *if v_i is an individual variable.*

(b) $c_i^{\mathfrak{A}}[\alpha] = c_i^{\mathfrak{A}}$, *if c_i is an individual constant.*

(c) $f_{i,j}(t_1, \ldots, t_j)^{\mathfrak{A}}[\alpha] = f_{i,j}^{\mathfrak{A}}(t_1^{\mathfrak{A}}[\alpha], \ldots, t_j^{\mathfrak{A}}[\alpha])$

ii. An assignment α in a structure $\mathfrak{A}$ for a language $\mathcal{L}$ satisfies *a formula ϕ of $\mathcal{L}$ (i.e. $\mathfrak{A} \models \phi[\alpha]$) iff one of the following conditions is verified.*

(a) ϕ *is an atomic formula of the form* $(t_1{=}t_2)$ *and* $t_1^{\mathfrak{A}}[\alpha] = t_2^{\mathfrak{A}}[\alpha]$

(b) ϕ *is an atomic formula of the form* $p_{i,j}(t_1, \ldots, t_j)$ *and* $\langle\, t_1^{\mathfrak{A}}[\alpha], \ldots, t_j^{\mathfrak{A}}[\alpha] \,\rangle \in p_{i,j}^{\mathfrak{A}}$

(c) $\phi \overset{s}{=} (\neg\psi)$, *and α does not satisfy ψ*

(d) $\phi \overset{s}{=} (\psi_1 \wedge \psi_2)$, *and α satisfies both ψ_1 and ψ_2.*

(e) $\phi \overset{s}{=} (\psi_1 \vee \psi_2)$, *and α satisfies at least one of the disjuncts ψ_1, ψ_2*

(f) $\phi \overset{s}{=} (\psi_1 \supset \psi_2)$, *and α satisfies $\neg\psi_1 \vee \psi_2$*

(g) $\phi \overset{s}{=} (\psi_1 \equiv \psi_2)$, *and α satisfies $(\neg\psi_1 \vee \psi_2) \wedge (\neg\psi_2 \vee \psi_1)$*

(h) $\phi \overset{s}{=} (\forall v_i)\psi$, *where v_i is an individual variable of $\mathcal{L}$, and every assignment α' in $\mathfrak{A}$ of the form $(\alpha - \{^{a_i}/_{v_i}\}) \cup \{^{a'_i}/_{v_i}\}$ satisfies ψ*

(i) $\phi \overset{s}{=} (\exists v_i)\psi$, *where v_i is an individual variable of $\mathcal{L}$, and at least one assignment α' in $\mathfrak{A}$ of the form $(\alpha - \{^{a_i}/_{v_i}\}) \cup \{^{a'_i}/_{v_i}\}$ satisfies ψ*

Satisfiability and *validity* are both related to the meaning of formulae. A formula is satisfiable when there is an assignment for its variables which turn it into a true statement in a structure. A formula is valid when it becomes a true statement for every assignment in a structure. Each notion defines a modality of the decision problem for formal languages, the *decision problem for validity* and the *decision problem for satisfiability.*

Definition B.1.11 (Satisfiability and Validity)

i. Let ϕ be a formula in a first-order language $\mathcal{L}$.

(a) ϕ is satisfiable in a structure $\mathfrak{A}$ for $\mathcal{L}$, i.e. $sat_{\mathfrak{A}}(\phi)$, iff there is an assignment α in $\mathfrak{A}$ such that α satisfies ϕ.

(b) ϕ is satisfiable, i.e. $sat(\phi)$, iff there exists a structure for $\mathcal{L}$ in which ϕ is satisfiable.

(c) *ϕ is valid in $\mathfrak{A}$, i.e. $\mathfrak{A} \models \phi$, iff it is satisfied by every assignment in $\mathfrak{A}$; the structure $\mathfrak{A}$ in this case represents a* model *for ϕ.*

(d) *ϕ is valid (or universally valid), i.e. $\models \phi$, iff it is valid in every structure for $\mathcal{L}$.*

(e) *$\top$ and $(\neg\bot)$ are valid.*

ii. *Let Γ be a set of formulae in a first-order language $\mathcal{L}$.*

(a) *Γ is satisfiable in a structure $\mathfrak{A}$ for $\mathcal{L}$, i.e. $sat_{\mathfrak{A}}(\Gamma)$, iff there is an assignment α in $\mathfrak{A}$ such that α satisfies each formula of Γ.*

(b) *Γ is satisfiable, i.e. $sat(\Gamma)$, iff there exists a structure for $\mathcal{L}$ in which it is satisfiable.*

(c) *Γ is valid in $\mathfrak{A}$, i.e. $\mathfrak{A} \models \Gamma$, iff each element of Γ is satisfied by every assignment in $\mathfrak{A}$; the structure $\mathfrak{A}$ in this case represents a* model *for Γ.*

(d) *Γ is valid (or universally valid), i.e. $\models \Gamma$, iff it is valid in every structure of $\mathcal{L}$.*

iii. *A formula ϕ of $\mathcal{L}$ is a* logical consequence *of a set of formulae Γ in $\mathcal{L}$ (or ϕ is* Γ-valid*), i.e.*

$$\Gamma \models \phi$$

iff every model of Γ is also a model of ϕ. ϕ is Γ-satisfiable *iff there is a model of Γ in which ϕ is satisfiable.*

Validity and satisfiability are linked in the next lemma.

Lemma B.1.1 *ϕ is Γ-valid iff $\neg\phi$ is Γ-unsatisfiable.*

PROOF. ϕ is Γ-valid iff for every model $\mathfrak{A}$ for Γ and every assignment α in $\mathfrak{A}$, $\mathfrak{A} \models \phi[\alpha]$, iff for every model $\mathfrak{A}$ and assignment α, $\mathfrak{A} \not\models \neg\phi[\alpha]$, iff there is no model for Γ where $\neg\phi$ is satisfiable iff $\neg\phi$ is Γ-unsatisfiable. ∎

B.2 First-Order Theories

First-order theories are sets of formulae that meet certain requirements. Under this viewpoint, the recognition of theorems becomes a special case of the problem of identifying the elements of a set. When formulae are mapped into the set of natural numbers, recursive functions assume a special role in the classification of theories. From an initially broad definition, a first proper group is then defined based on semi-recursive sets, which lead to the *axiomatisable theories*. Further restrictions lead to the *decidable theories*, which are based on recursive sets.

B.2.1 Effectivised Languages

Any countable set of symbols is translatable into the set of natural numbers. In effectivised languages, it is possible to effectively determine which symbols are function, predicate or individual constant symbols. The possibility of construction of effective procedures for the recognition of well-formed expressions is then open in such languages. An effective mechanism for the recognition of symbols, expressions and finite sequences of formulae of a language is provided by *Gödel functions.*

Definition B.2.1 (Gödel Functions)

Let $\mathcal{L} = \langle \ell, \mathcal{V}, \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$ *be a first-order language.*

i. g_s: $Sym_{\mathcal{L}} \rightarrow \mathbb{N}$ *is a* Gödel function for the symbols of $\mathcal{L}$ *iff it is bijective, and* $g_s(\mathcal{V}), g_s(\mathcal{C}), g_s(\mathcal{F})$ *and* $g_s(\mathcal{P})$ *are recursive sets.*

ii. g_e: $Exp_{\mathcal{L}} \rightarrow \mathbb{N}$ *is a* Gödel function for the expressions of $\mathcal{L}$ *(w.r.t.* g_s*) iff, given an expression* ϕ *composed of the* $\mathcal{L}$*-symbols* $S_1, \ldots, S_n$*, in this order,*

$$g_e(\phi) = \prod_{i=1}^{n} prm(i)^{1+g_s(S_i)}$$

where $prm(i)$ *represents the* i*-th natural prime number*[8].

iii. g_{se}: $LstFml_{\mathcal{L}} \rightarrow \mathbb{N}$ *is a* Gödel function for the finite sequences of formulae of $\mathcal{L}$ *(w.r.t.* g_s*) iff, for any sequence of formulae* $\phi_1, \ldots, \phi_n$*,*

$$g_{se}(\langle \phi_1, \ldots, \phi_n \rangle) = \prod_{i=1}^{n} prm(i)^{1+g_e(\phi_i)}$$

where $prm(i)$ *represents the* i*-th natural prime number.*

iv. $\widehat{\mathcal{L}} = \langle \ell, \mathcal{V}, \mathcal{C}, \mathcal{F}, \mathcal{P}, g_e \rangle$ *is an* effectivised first-order language.

The natural numbers $g_s(s), g_e(\phi)$ *and* $g_{se}(\langle \phi_1, \ldots, \phi_n \rangle)$ *can be respectively denoted as* $\ulcorner s \urcorner, \ulcorner \phi \urcorner$ *and* $\ulcorner \langle \phi_1, \ldots, \phi_n \rangle \urcorner$*. If* Γ *is a set of expressions of* $\mathcal{L}$*,* $\ulcorner \Gamma \urcorner$ *denotes the set of Gödel numbers for each element of* Γ*.*

Lemma B.2.1 *Both* g_e *and* g_{se} *are injective.*

PROOF. Let $g_e(\phi) = g_e(\phi')$. Applying the definition of g_e to the hypothesis, it is possible to derive that

$$\prod_{i=1}^{n} pr(i)^{g_s(s_i)+1} = \prod_{i=1}^{n'} pr(i)^{g_s(s'_i)+1}$$

[8] It would be actually necessary to include delimiters e.g. parentheses among the logical symbols of $\mathcal{L}$, otherwise a list of symbols would not necessarily represent a single formula.

According to the fundamental theorem of arithmetic, every natural number greater than 1 has a single decomposition into prime numbers (modulo ordering of prime factors). Since all the exponents in both sides of the equality are non-null, and since all the prime factors are in increasing order, it follows that $n = n'$ and, for each $i, 1 \leq i \leq n$,

$$g_s(s_i) + 1 = g_s(s'_i) + 1$$

$$g_s(s_i) = g_s(s'_i)$$

Since g_s is bijective, then $s_i = s'_i$ and $s_1, \ldots, s_n = s'_1, \ldots, s'_{n'}$, i.e. $\phi = \phi'$.

A similar proof applies to g_{se}. ∎

As a result, given that the images of $g_s \restriction \mathcal{V}$, $g_s \restriction \mathcal{C}$, $g_s \restriction \mathcal{F}$ and $g_s \restriction \mathcal{P}$ are recursive, and that terms and formulae of a first-order language are recursively defined, it only takes a finite amount of computation to determine whether a sequence of symbols of $\widehat{\mathcal{L}}$ is a well-formed expression or not.

B.2.2 Axiomatisable Theories

A first-order theory is any set of formulae that is closed with respect to the logical consequence relation. Some theories can be generated from proper recursive subsets, whose elements are their *axioms*.

Definition B.2.2 (Axiom Sets)

Let $\widehat{\mathcal{L}}$ be an effectivised first-order language.

i. A theory T in $\widehat{\mathcal{L}}$ is a subset of $Fml_{\widehat{\mathcal{L}}}$ such that $\phi \in T$ iff $T \models \phi$. Each element of T is a theorem of T *(or* T-theorem*). $\widehat{\mathcal{L}}$ is the* underlying *(or* subjacent*)* language of T.

ii. An axiom set for T is a subset $\Delta_T \subseteq T$ such that $\phi \in T$ iff $\Delta_T \models \phi$.

iii. T is finitely axiomatisable iff T admits a finite set of axioms. T is recursively axiomatisable iff T admits a set of axioms Δ_T such that $\ulcorner \Delta_T \urcorner$ is recursive.

iv. The theory $T_{\mathfrak{A}}$ of a structure $\mathfrak{A}$ for $\widehat{\mathcal{L}}$ is the set of all the formulae ϕ of $\widehat{\mathcal{L}}$ that are valid in $\mathfrak{A}$.

v. $T_0^{\widehat{\mathcal{L}}}$ is the set of all universally valid formulae of $\widehat{\mathcal{L}}$ (also called the (first-order) predicate calculus for $\widehat{\mathcal{L}}$*).*

Definition B.2.3 (Properties of First-order theories)

Let T be a theory in $\mathcal{L}$.

i. T *is* consistent *iff it is not the case that both* τ *and* $\neg\tau$ *belong to* T*, for any* $\tau \in Stn_{\mathcal{L}}$.

ii. T *is* maximal consistent *if, whenever* $T \not\models \tau$*, then* $T \cup \{\tau\}$ *is inconsistent.*

iii. T *is* (negation) complete *if, given any sentence* $\tau \in Stn_{\mathcal{L}}$*, either* $\tau \in T$ *or* $\neg\tau \in T$.

Lemma B.2.2 *If* T *is a consistent theory in* $\mathcal{L}$*, then there is no formula* ϕ *in* $Fml_{\mathcal{L}}$ *such that both* ϕ *and* $\neg\phi$ *are elements of* T.

PROOF. If $\phi \in Stn_{\mathcal{L}}$, the lemma is obviously true. Let then ϕ be an open such that both it and its negation belong to T. Then $[\phi]$ must also belong to T, and so

$$[[\phi]] \in T, \quad (*)$$

since $\models (v)\psi \supset (\exists v)\psi$. Also, as $\neg\phi$ is a T-theorem, the same applies to its universal closure, $[\neg\phi]$. Hence

$$\neg[[\phi]] \in T \quad (**)$$

From $(*)$ and $(**)$, it follows that T is inconsistent, in contradiction with the hypothesis. ∎

Example B.2.1 PA_0 (basic Peano arithmetic) *and* PrA_0 (basic Presburger arithmetic)*, respectively formulated in* $\mathcal{L}_{PA_0}$ *and* $\mathcal{L}_{PrA_0}$*, are two examples of axiomatisable theories. An axiom set for* PA_0 *is listed below.*

$$\begin{array}{ll}
(PA_0^1) & 0 \neq s(x) \\
(PA_0^2) & s(x) = s(y) \supset x = y \\
(PA_0^3) & x + 0 = x \\
(PA_0^4) & x + s(y) = s(x + y) \\
(PA_0^5) & x \times 0 = 0 \\
(PA_0^6) & x \times s(y) = x + x \times y \\
(PA_0^7) & (\phi[0] \wedge (x)(\phi[x] \supset \phi[s(x)])) \supset (x)\phi[x]
\end{array}$$

This set of axioms is not finite due to the presence of axiom-scheme PA_0^7. $\{PA_0^1, \ldots, PA_0^4\} \cup \{PA_0^7\}$ *represents an axiom set for* PrA_0*. Both* PA_0 *and* PrA_0 *are consistent*[9]. □

Whereas languages can be expanded by means of the introduction of new non-logical symbols, theories can be extended through the insertion of new formulae, taken either from the original or an expanded language. An extension of a theory is conservative iff none of its new elements belongs to the original underlying language. Definitional extensions concern those cases where new symbols are introduced for notational convenience: they can be always expressed in terms of the original symbols.

[9] $t \neq u$ stands for $\neg(t = u)$.

Definition B.2.4 (Conservative and Definitional Extensions)

i. Let $\mathcal{L}'$ be an expansion of $\mathcal{L}$, and let T' and T respectively be theories in $\mathcal{L}'$ and $\mathcal{L}$.

(a) T' is an extension *of T (and T is a* subtheory *of T') iff $T \subseteq T'$.*

(b) T' is a conservative extension *of T iff T' is an extension of T and*

$$T' \cap Fml_{\mathcal{L}} = T$$

ii. Let $\mathcal{L}' = \langle \mathcal{C}', \mathcal{F}', \mathcal{P}' \rangle$ be an expansion of $\mathcal{L} = \langle \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$, such that $\overline{\mathcal{C}} = \mathcal{C}' - \mathcal{C}$, $\overline{\mathcal{F}} = \mathcal{F}' - \mathcal{F}$ and $\overline{\mathcal{P}} = \mathcal{P}' - \mathcal{P}$ are non-empty sets, and let T and T' be theories respectively in $\mathcal{L}$ and $\mathcal{L}'$.

(a) A possible definition *for an individual constant symbol c_i of $\overline{\mathcal{C}}$ over T is any formula $\phi_{c_i}\{v\}$ of $\mathcal{L}$ that satisfies the following properties of existence and uniqueness*

$$(\exists v)\phi_{c_i}\{v\}$$

$$(v)(v')(\phi_{c_i}\{v\} \wedge \phi_{c_i}\{^{v'}/_{v}\} \supset v = v')$$

(b) A possible definition *for a function symbol $f_{i,j}$ of arity j over T, where $f_{i,j} \in \overline{\mathcal{F}}$, is any formula $\phi_{f_{i,j}}\{v_1, \ldots, v_{j+1}\}$ of $\mathcal{L}$ that satisfies the following properties of existence and uniqueness*

$$(v_1)\ldots(v_j)(\exists v_{j+1})\phi_{f_{i,j}}\{v_1, \ldots, v_{j+1}\}$$

$$(v_1)\ldots(v_{j+2})(\phi_{f_{i,j}}\{v_1, \ldots, v_{j+1}\} \wedge \phi_{f_{i,j}}\{v_1, \ldots, v_j, {}^{v_{j+2}}/_{v_{j+1}}\} \supset v_{j+1} = v_{j+2})$$

(c) A possible definition *for a predicate symbol $p_{i,j}$ of arity j over T, where $f_{i,j} \in \overline{\mathcal{P}}$, is any formula $\phi_{p_{i,j}}\{v_1, \ldots, v_j\}$ of $\mathcal{L}$.*

(d) T' is a definitional extension *of T if and only if, for every symbol S_i in $\overline{\mathcal{C}} \cup \overline{\mathcal{F}} \cup \overline{\mathcal{P}}$, there is a possible definition ϕ_{S_i} over T such that*

$$T' = Stn_{\mathcal{L}'} \cap \{\psi \mid T \cup \{\phi'_{S_1}, \ldots, \phi'_{S_n}, \ldots\} \models \psi\}$$

where ϕ'_{S_k} is one of the sentences

$$(v)(\phi_{c_i} \equiv (c_i{=}v))$$

$$(v_1)\ldots(v_{j+1})(\phi_{f_{i,j}} \equiv (f_{i,j}(v_1, \ldots, v_j){=}v_{j+1}))$$

$$(v_1)\ldots(v_j)(\phi_{p_{i,j}} \equiv p_{i,j}(v_1, \ldots, v_j))$$

depending on whether $S_k \stackrel{s}{=} c_i$, or $S_k \stackrel{s}{=} f_{i,j}$, or $S_k \stackrel{s}{=} p_{i,j}$.

Extensions of theories have an important role in the study of decidable theories. Particularly relevant are the properties concerning (a) negation complete theories and their extensions, and (b) definitional and conservative extensions.

Lemma B.2.3 *Let $\mathcal{L}'$ be an expansion of $\mathcal{L}$, and let T and T' respectively be theories in $\mathcal{L}$ and $\mathcal{L}'$.*

i. T is an extension of $T_0^{\mathcal{L}}$.

ii. The restriction of T' to $\mathcal{L}$ (i.e. $T' \cap Fml_{\mathcal{L}}$) is a theory.

iii. If Δ_T is an axiom set for both T and T', then T' is a conservative extension of T.

iv. If T is negation complete and T' is a consistent extension of T, then T' is a conservative extension of T.

v. If T' is a definitional extension of T, it is also a conservative extension of T.

PROOF.

i. If T is a theory in $\mathcal{L}$, then $\phi \in T$ iff $T \models \phi$, for all $\phi \in Fml_{\mathcal{L}}$. Since all universally valid formulae are valid in the models of T, it follows that $T_0^{\mathcal{L}} \subseteq T$.

ii. Let $\phi \in Fml_{\mathcal{L}}$. If $(T' \cap Fml_{\mathcal{L}}) \models \phi$, then $T' \models \phi$. Since T' is a theory, then $\phi \in T'$, hence $\phi \in T' \cap Fml_{\mathcal{L}}$.

iii. (a) If $\phi \in T$, then $\phi \in Fml_{\mathcal{L}}$ and $\Delta_T \models \phi$. Since $\mathcal{L}'$ is an expansion of $\mathcal{L}$, $\phi \in Fml_{\mathcal{L}'}$ as well, hence $\phi \in T' \cap Fml_{\mathcal{L}}$.

(b) If $\phi \in Fml_{\mathcal{L}} \cap T'$, i.e. $\phi \in Fml_{\mathcal{L}}$ is a theorem of T', then $\Delta_T \models \phi$. As T is a theory in $\mathcal{L}$ for which Δ_T is an axiom set, it follows that $\phi \in T$.

From (a) and (b), $Fml_{\mathcal{L}} \cap T' = T$.

iv. Assume $T' \cap Fml_{\mathcal{L}} \neq T$; then there is a formula $\phi \in Fml_{\mathcal{L}}$ such that $\phi \in T'$ but $\phi \notin T$. Given that $T \models \phi$ iff $T \models [\phi]$ and that T is negation complete, then $\neg[\phi] \in T$. Since $T \subset T'$, $\neg[\phi] \in T'$ as well. Therefore $[\phi]$ and $\neg[\phi]$ are theorems of T', which must then be inconsistent.

v. See [Monk 76], p. 210. ∎

The relationship between PA_0, PrA_0 and SMA_0 can be expressed in terms of the new concepts.

Example B.2.2

i. PA_0 is an extension of PrA_0, since

(a) $\mathcal{L}_{PA_0} = \{0, s, +, \times\}$ is an expansion of $\mathcal{L}_{PrA_0} = \{0, s, +\}$, and
(b) PrA_0 admits an axiom set which is a subset of the axiom set for PA_0.

ii. A simple definitional extension PA_0' of basic Peano arithmetic can be generated in $\mathcal{L}_{PA} = \{0, 1, s, +, \times\}$, with the introduction of a new axiom,

$$(PA_0^8) \quad 1{=}s(0)$$

The formula $s(0){=}v$ in $\mathcal{L}_{PA_0}$ is a possible definition of 1 in PA_0; moreover,

$$PA_0' \;=\; Stn_{\mathcal{L}_{PA}} \cap \{\psi \mid PA_0 \cup \{(v)((s(0){=}v) \equiv (1{=}v))\} \models \psi\}$$

iii. Basic strictly multiplicative arithmetic *(SMA_0) can be defined as*

$$SMA_0 = PA_0' \cap Fml_{\mathcal{L}_{SMA}}$$

where $\mathcal{L}_{SMA} = \{0, 1, \times\}$, and PA_0' is the definitional extension of PA_0 described above. PA_0' therefore is a conservative extension of SMA_0. □

B.2.3 Decidable Theories

Decision problems are present in various domains, including algebraic theories (e.g. word problems) and mathematical logic. Many of them can be abstractly represented as the problem of effectively identifying subsets of a particular universe, as well as their complements. Whenever the universe can be mapped into the set of natural numbers by means of a Gödel function, recursive functions provide the means for the effective recognition of decidable subsets[10].

Definition B.2.5 (Decision Procedures)

Let $\mathcal{A}$ be a subset of a countable universe set $\mathcal{U}$, and let $g_{\mathcal{U}}$ be a Gödel function for $\mathcal{U}$ (i.e. an effective bijective mapping from $\mathcal{U}$ into $\mathbb{N}$).

i. A decision procedure for $\mathcal{A}$ w.r.t. $\mathcal{U}$ is a total recursive function f such that, whenever $u \in \mathcal{U}$,

$$f(\ulcorner u \urcorner) = \begin{cases} 1 & \text{if } u \in \mathcal{A} \\ 0 & \text{otherwise} \end{cases}$$

ii. A semi-decision procedure for $\mathcal{A}$ w.r.t. $\mathcal{U}$ is a partial recursive function f such that, whenever $u \in \mathcal{U}$,

$$f(\ulcorner u \urcorner) = \begin{cases} 1 & \text{if } u \in \mathcal{A} \\ 0 \text{ or undefined} & \text{otherwise} \end{cases}$$

---

[10] See [Shoenfield 67], p. 106-7. The term *decidable* has a different meaning when applied to formulae: a formula is decidable in a (consistent) theory T iff either it or its negation (but not both) belongs to T. The existence of undecidable formulae in a theory does not prevent this theory from being decidable.

iii. $\mathcal{A}$ *is* (recursively) decidable *in* $\mathcal{U}$ *iff there exists a decision procedure for* $\mathcal{A}$ *w.r.t.* $\mathcal{U}$. $\mathcal{A}$ *is* (recursively) semi-decidable *in* $\mathcal{U}$ *iff there exists a semi-decision procedure for* $\mathcal{A}$ *w.r.t.* $\mathcal{U}$.

Clearly, $\mathcal{A}$ is (recursively) decidable if and only if $\ulcorner\mathcal{A}\urcorner$ is recursive; otherwise it is (recursively) undecidable (or unsolvable). If $\mathcal{A}$ is undecidable, it is (recursively) semi-decidable if and only if $\ulcorner\mathcal{A}\urcorner$ is semi-recursive (or recursively enumerable)[11].

According to definition B.2.5, any pair $\langle\mathcal{U},\mathcal{A}\rangle$, where $\mathcal{U}$ is a countable universe and $\mathcal{A}\subseteq\mathcal{U}$, determines a decision problem. In the context of a first-order language $\mathcal{L}$, the first option for a universe is the set $Sqn_{\mathcal{L}}$ of finite sequences of symbols of $\mathcal{L}$. Given a Gödel function for sequences of $\mathcal{L}$-symbols g: $Sqn_{\mathcal{L}}\rightarrow\mathbb{N}$, a decision procedure for a subset $\mathcal{A}\subseteq Sqn_{\mathcal{L}}$ is a recursive function f: $\ulcorner Sqn_{\mathcal{L}}\urcorner\rightarrow\mathbb{N}$ such that

$$f(\ulcorner s\urcorner)=\begin{cases}1 & \text{if } s\in\mathcal{A}\\ 0 & \text{otherwise}\end{cases}$$

Even when $\mathcal{A}$ is a subset of well-formed formulae, the universe cannot be restricted to $Fml_{\mathcal{L}}$ if $\mathcal{L}$ is not effectivised, since in this case $Fml_{\mathcal{L}}$ is not a decidable subset of $Sqn_{\mathcal{L}}$, and there is no effective way of establishing whether a finite sequence of symbols is a formula or not. When $\mathcal{L}$ is effectivised, on the other hand, a decision procedure for $\mathcal{A}$ can be built as follows:

(i) given a finite sequence of symbols $s\in Sqn_{\mathcal{L}}$, it is first supplied to a recogniser for wff of $\mathcal{L}$: if it is not a wff, it cannot belong to $\mathcal{A}$ either, and the process halts;
(ii) if s is a wff, it is supplied to the decision procedure for $\mathcal{A}$.

As a result, in the context of effectivised languages, it suffices to exhibit a recursive function f whose restriction to $\ulcorner Fml_{\mathcal{L}}\urcorner$ satisfies the condition

$$f(\ulcorner\phi\urcorner)=\begin{cases}1 & \text{if } \phi\in\mathcal{A}\\ 0 & \text{otherwise}\end{cases}$$

for every $\phi\in Fml_{\mathcal{L}}$.

Once the universe and a corresponding Gödel function are defined, a subset of formulae $\mathcal{A}$ may be delimited by a property expressed in a (meta)language. In particular, there is a class of logical decision problems where the definition of $\mathcal{A}$ involves either the notion of validity or the notion of satisfiability. Given a set of $\mathcal{L}$-structures $A=\{\mathfrak{A}_1,\ldots,\mathfrak{A}_i,\ldots\}$, the *decision problem for validity in* A concerns the existence of an effective mechanism for the identification of the formulae of $\mathcal{L}$ that are valid in all the structures in A; when A contains every structure of $\mathcal{L}$, it corresponds to the decision problem for validity in $\mathcal{L}$. $\mathcal{A}$ in this case can be defined as

$$\phi\in\mathcal{A}\quad\text{iff}\quad(\forall\mathfrak{A})(\mathfrak{A}\in A\supset\mathfrak{A}\models\phi)$$

The *decision problem for satisfiability in* A involves finding an effective mechanism to determine whether a formula of $\mathcal{L}$ is satisfiable in at least one structure in A. $\mathcal{A}$ then corresponds to

[11] See section B.1.1.

$$\phi \in \mathcal{A} \quad \text{iff} \quad (\exists \mathfrak{A})(\mathfrak{A} \in A \;\&\; sat_{\mathfrak{A}}(\phi))$$

Both problems are closely linked. It is actually possible to restrict the study of logical decision problems to the case of validity[12].

Lemma B.2.4 *Let $A = \{\mathfrak{A}_1, \ldots, \mathfrak{A}_i, \ldots\}$ be a set of $\mathcal{L}$-structures. The decision problem for validity in A has a positive solution iff there is a solution for the decision problem for satisfiability in A.*

PROOF. If the decision problem for validity in A has a positive solution, there exists a recursive function f such that, for all $\phi \in Fml_{\mathcal{L}}$,

$$f(\ulcorner \phi \urcorner) = \begin{cases} 1 & \text{if } \mathfrak{A}_i \models \phi, \text{ for all } \mathfrak{A}_i \in A \\ 0 & \text{otherwise} \end{cases}$$

Since the validity of ϕ implies the unsatisfiability of $\neg\phi$ (and vice versa), it follows that

$$f(\ulcorner \phi \urcorner) = \begin{cases} 1 & \text{if } \neg sat_{\mathfrak{A}_i}(\neg\phi), \text{ for all } \mathfrak{A}_i \in A \\ 0 & \text{if } sat_{\mathfrak{A}_i}(\neg\phi), \text{ for some } \mathfrak{A}_i \in A \end{cases}$$

If h is defined as $h(\ulcorner \phi \urcorner) = 1 - f(\ulcorner \neg\phi \urcorner)$, h then is a decision procedure for satisfiability in A.

A similar proof can be provided for the case where there is a positive solution to the decision problem for satisfiability in A. ∎

When a decision problem for validity involves the set of models for a theory T, since the set of formulae that are valid in all its models coincides with T itself, this is also called the *decision problem for T (w.r.t. $Fml_{\mathcal{L}}$)*. Hence T is (recursively) decidable if and only if there is a total recursive function f such that, for every formula ϕ of $\mathcal{L}$,

$$f(\ulcorner \phi \urcorner) = \begin{cases} 1 & \text{if } \phi \in T \\ 0 & \text{if } \phi \notin T \end{cases}$$

where f represents a *decision procedure for T (w.r.t. $Fml_{\mathcal{L}}$)*. Alternatively, T is (recursively) semi-decidable if and only if there is a partial recursive function h such that, for every formula ϕ of $\mathcal{L}$,

$$h(\ulcorner \phi \urcorner) = \begin{cases} 1 & \text{if } \phi \in T \\ 0 \text{ or undefined} & \text{if } \phi \notin T \end{cases}$$

[12] As it is discussed in appendix C, this property may not hold for decision problems where the universe is a proper subclass of $Fml_{\mathcal{L}}$. Also, the symbol & denotes metatheoretical conjunction.

where h corresponds to a *semi-decision procedure for T w.r.t. $Fml_{\mathcal{L}}$*. Any decision procedure is also a semi-decision procedure for T w.r.t. $Fml_{\mathcal{L}}$. Both cases are summarised in the next definition[13].

Definition B.2.6 (Recursive Decidability)

Let T be a theory in an effectivised language $\widehat{\mathcal{L}}$.

i. T is (recursively) decidable (in $Fml_{\widehat{\mathcal{L}}}$) *iff $\ulcorner T \urcorner$ is a recursive set.*

ii. T is (recursively) semi-decidable (in $Fml_{\widehat{\mathcal{L}}}$) *iff $\ulcorner T \urcorner$ is a recursively enumerable set.*

Presburger arithmetic (PrA) and Peano arithmetic (PA), defined in section B.3.2, are respectively examples of a decidable and a semi-decidable theory. Given that all the languages to be considered in this study are both first-order and effectivised, the adjectives will be normally omitted.

B.3 Decidability Proofs

There are two approaches for the solution of the decision problem for theories. Given that the existence of decision procedures is at stake, the direct or *constructive* approach consists of exhibiting one for each decidable theory, or else presenting a method for its construction. The indirect approach, on the other hand, searches for metatheoretical properties of a theory which ensure its decidability. An indirect proof may under certain circumstances lead to the identification of a generic but usually inefficient procedure[14].

B.3.1 Indirect Approach

Decidability is linked with metatheoretical properties such as negation completeness and recursive axiomatisability. Semidecision procedures, on the other hand, can be always built for recursively axiomatisable theories.

Theorem B.3.1 [Monk 76] *If T is a recursively axiomatisable theory in $\mathcal{L}$, then $\ulcorner T \urcorner$ is recursively enumerable.*

---

[13] Decidable theories do not necessarily have to be defined in effectivised languages. In a non-effectivisable language, since the definition of a Gödel function g cannot be effectively restricted to the set of well-formed formulae of $\mathcal{L}$, a decision procedure for a theory T must have $Sqn_{\mathcal{L}}$ as universe. In this case, when a sequence is not identified as a theorem, it is not always possible to determine whether it is a non-theorem (i.e. a well-formed formula of $\mathcal{L}$ that does not belong to T) or a non-expression (i.e. a sequence of symbols of $\mathcal{L}$ that does not represent a well-formed formula).

[14] This section has been mainly based on [Shoenfield 67], [Monk 76], [Chang & Keisler 73], [Kreisel & Krivine 67a] and [Enderton 72]. Indirect proofs of decidability could be also called *classical*, in the sense that a classical proof for an existential statement may simply establish the impossibility of nonexistence of a particular object.

PROOF. Let f be a recursive function that enumerates the axioms of T, and let g be a recursive function that enumerates $Fml_{\mathcal{L}}$. A procedure $P(n)$ that generates all the elements of T can be informally defined as follows.

i. $P(0) = \{f(0)\}$ (i.e. the first element of T is its first axiom, according to the enumeration provided by f).

ii. For each $n \in \mathbb{N}, n > 0$, let $\mathcal{S}$ be the set of all sequences of maximum length n, made up of formulae of $\mathcal{L}$ taken from the set $\{g(0), \ldots, g(n)\}$, and let $\{s_1, \ldots, s_m\}$ be the sequences of $\mathcal{S}$ that correspond to proofs in T in which the premisses are only formulae already generated by f or P. Then $P(n)$ is defined as the set of all the last elements of the sequences $s_1, \ldots, s_m$ (i.e. the theorems whose proofs have length less than or equal to n, and did not occur at a previous stage of the process).

From P it is possible to define a recursive function h that enumerates T. ∎

Theorem B.3.2 *Let T be a theory in $\mathcal{L}$. If T is negation complete, then T is decidable if and only if T is recursively axiomatisable.*

PROOF. If T is negation complete and decidable, $\ulcorner T \urcorner$ is recursive. Since any theory represents an axiom set for itself, it follows that T is recursively axiomatisable.

If T is negation complete and recursively axiomatisable, there are two cases to be taken into account.

i. T is inconsistent. Then it is trivially decidable (since, in this case, $T = Fml_{\mathcal{L}}$).

ii. T is consistent. Let then $\phi \in Fml_{\mathcal{L}}$ and $\tau \overset{s}{=} [\phi]$. Since T is negation complete, either $\tau \in T$ or $\neg\tau \in T$. Since T is recursively axiomatisable, $\ulcorner T \urcorner$ is recursively enumerable: there is a recursive function f_T such that $f_T(n) \in \{\ulcorner \tau \urcorner, \ulcorner \neg\tau \urcorner\}$, for some $n \in \mathbb{N}$. Hence the function g_T, defined as

$$g_T(\ulcorner \tau \urcorner) = \begin{cases} 1 & \text{if } (\exists n \in \mathbb{N})(f_T(n) = \ulcorner \tau \urcorner) \\ 0 & \text{if } (\exists n \in \mathbb{N})(f_T(n) = \ulcorner \neg\tau \urcorner) \end{cases}$$

is recursive, for $(\exists n \in \mathbb{N})(f_T(n) = \ulcorner \tau \urcorner \vee f_T(n) = \ulcorner \neg\tau \urcorner)$ is decidable. ∎

There are various results relating the decidability of a theory to the decidability of its extensions. Certain theories that admit finite models have consistent decidable extensions, as for instance in the case of the theory of groups and its extension *DAG*. Some processes of extension, on the other hand, preserve the decidability of the original theory.

Theorem B.3.3 *Let T be a theory in $\mathcal{L}$ and T' be a theory in $\mathcal{L}'$, where $\mathcal{L}'$ is an expansion of $\mathcal{L}$. If T' is a decidable conservative extension of T, then T is decidable.*

PROOF. Let $f_{T'}$ be a decision procedure for T' w.r.t. $Fml_{\mathcal{L}'}$, and let g be the recursive function

$$g(\ulcorner \phi \urcorner) = \begin{cases} 1 & \text{if } \phi \in Fml_{\mathcal{L}} \\ 0 & \text{otherwise} \end{cases}$$

The recursive function $h_T(\ulcorner \phi \urcorner) = g(\ulcorner \phi \urcorner) \times f_{T'}(\ulcorner \phi \urcorner)$ can be also represented as

$$h_T(\ulcorner \phi \urcorner) = \begin{cases} 1 & \text{if } \phi \in Fml_{\mathcal{L}} \cap T' \\ 0 & \text{otherwise} \end{cases}$$

Since T' is a conservative extension of T, $Fml_{\mathcal{L}} \cap T' = T$, hence h_T is a decision procedure for T w.r.t. $Fml_{\mathcal{L}'}$; a decision procedure for T w.r.t. $Fml_{\mathcal{L}}$ can be immediately generated from h_T. ∎

Before examining other results about extensions of theories, a few preliminary lemmas are required.

Lemma B.3.1 *Let* (i) *T be a theory in $\mathcal{L}$,* (ii) *$\mathcal{L}'$ be an expansion of $\mathcal{L}$, and* (iii) *$\mathfrak{A}$ be a $\mathcal{L}$-structure that is a model for T. Then any $\mathcal{L}'$-expansion of $\mathfrak{A}$ is also a model for T.*

PROOF. Let $\mathfrak{A}'$ be a $\mathcal{L}'$-expansion of $\mathfrak{A}$. If $\phi \in T$, then $\mathfrak{A} \models \phi$, i.e. for every assignment α in $\mathfrak{A}$,

$$\mathfrak{A} \models \phi[\alpha]$$

From the definition of an expansion of a structure, it follows that

$$\mathfrak{A}' \models \phi[\alpha]$$

since $|\mathfrak{A}'| = |\mathfrak{A}|$ and $S^{\mathfrak{A}'} = S^{\mathfrak{A}}$, for every $S \in Sym_{\mathcal{L}}$. Hence, for any $\phi \in T$, $\mathfrak{A}' \models \phi$, i.e. $\mathfrak{A}' \models T$. ∎

Lemma B.3.2 *Let $\mathcal{L}' = \langle \mathcal{C}', \mathcal{F}, \mathcal{P}, g'_e \rangle$ be an effective expansion of a language $\mathcal{L} = \langle \mathcal{C}, \mathcal{F}, \mathcal{P}, g_e \rangle$ such that $\mathcal{C} \subset \mathcal{C}'$ (i.e. $\mathcal{L}'$ is generated from $\mathcal{L}$ by the introduction of new individual constants only). Let T and T' respectively be theories in $\mathcal{L}$ and $\mathcal{L}'$ such that T' is the extension of T defined as $\{\phi \in Fml_{\mathcal{L}'} \mid T \models \phi\}$. Let $\psi[\![c_1, \ldots, c_n]\!] \in Fml_{\mathcal{L}'}$, where $\{c_1, \ldots, c_n\} \subseteq \mathcal{C}' - \mathcal{C}$. Then*

$$T' \models \psi[\![c_1, \ldots, c_n]\!] \quad \textit{iff} \quad T' \models \psi[\![^{v_1}/_{c_1}, \ldots, ^{v_n}/_{c_n}]\!] \quad (*)$$

whenever $v_1, \ldots, v_n$ are variables that do not occur in $\psi[\![c_1, \ldots, c_n]\!]$.

PROOF.

If $T' \models \psi[\![^{v_1}/_{c_1}, \ldots, ^{v_n}/_{c_n}]\!]$, any instance of $\psi[\![^{v_1}/_{c_1}, \ldots, ^{v_n}/_{c_n}]\!]$ is also T'-valid, hence $T' \models \psi[\![c_1, \ldots, c_n]\!]$.

If $T' \models \psi[\![c_1, \ldots, c_n]\!]$, then, from the definition of T', $T \models \psi[\![c_1, \ldots, c_n]\!]$. As a result, for any $\mathcal{L}'$-structure $\mathfrak{A}$ that is a model for T,

$$\mathfrak{A} \models \psi[\![c_1, \ldots, c_n]\!] \quad (*)$$

If $T' \not\models \psi[\![^{v_1}/_{c_1}, \ldots, ^{v_n}/_{c_n}]\!]$, since $T \subset T'$, it follows that $T \not\models \psi[\![^{v_1}/_{c_1}, \ldots, ^{v_n}/_{c_n}]\!]$. Hence there must be a $\mathcal{L}$-structure $\mathfrak{B}$ that is a model for T in which $\psi[\![^{v_1}/_{c_1}, \ldots, ^{v_n}/_{c_n}]\!]$ is invalid, i.e. there is an assignment $\beta = \{^{b_1}/_{v_1}, \ldots, ^{b_n}/_{v_n}\}$ in $\mathfrak{B}$, with $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n \in |\mathfrak{B}|$, such that

$$\mathfrak{B} \not\models \psi[\![^{v_1}/_{c_1}, \ldots, ^{v_n}/_{c_n}]\!][\beta]$$

Let $\mathfrak{B}'$ be a $\mathcal{L}'$-expansion of $\mathfrak{B}$ such that $c_i^{\mathfrak{B}'} = \boldsymbol{b}_i, 1 \leq i \leq n$. Then

$$\mathfrak{B}' \not\models \psi[\![c_1, \ldots, c_n]\!] \quad (**)$$

From lemma B.3.1, $\mathfrak{B}'$ is a $\mathcal{L}'$-model for T. From this result and $(*)$, it follows that $\mathfrak{B}' \models \psi[\![c_1, \ldots, c_n]\!]$, thus contradicting $(**)$. Hence $T' \models \psi[\![^{v_1}/_{c_1}, \ldots, ^{v_n}/_{c_n}]\!]$. ∎

Theorem B.3.4 *Let T and T' respectively be theories in $\mathcal{L}$ and $\mathcal{L}'$.*

i. If T is consistent and decidable, then there is a maximal consistent decidable extension of T.

ii. If T' is an effective definitional extension of T, then T is decidable iff T' is decidable.

iii. If $\mathcal{L}' = \langle \mathcal{C}', \mathcal{F}, \mathcal{P}, g'_e \rangle$ is an effective expansion of $\mathcal{L} = \langle \mathcal{C}, \mathcal{F}, \mathcal{P}, g_e \rangle$ such that $\mathcal{C} \subset \mathcal{C}'$ (i.e. $\mathcal{L}'$ is generated from $\mathcal{L}$ by the introduction of new individual constants only), and if T' is an extension of T such that

$$T' = \{\phi \in Fml_{\mathcal{L}'} \mid T \models \phi\}$$

then T is decidable iff T' is decidable[15].

PROOF.

i. See [Monk 76].

[15] This lemma has been proposed as a problem in [Monk 76], p. 277.

ii. See [Monk 76].

iii. Since T is an axiom set for T', according to lemma B.2.3, T' is a conservative extension of T, i.e. $T' \cap Fml_{\mathcal{L}} = T$.

If T is decidable, let ψ be a formula of $\mathcal{L}'$.

(a) If $\psi \in Fml_{\mathcal{L}}$, since T' is a conservative extension of T, it follows that

$$\psi \in T \text{ iff } \psi \in T' \quad (*)$$

(b) If $\psi \notin Fml_{\mathcal{L}}$, then ψ must have the form $\psi[\![c_1, \ldots, c_n]\!]$, where $\{c_1, \ldots, c_n\} \subseteq \mathcal{C}' - \mathcal{C}$. From lemma B.3.2, $T' \models \psi[\![c_1, \ldots, c_n]\!]$ iff $T' \models \psi[\![^{v_1}/_{c_1}, \ldots, ^{v_n}/_{c_n}]\!]$, where $v_1, \ldots, v_n$ are variables that do not occur in $\psi[\![c_1, \ldots, c_n]\!]$. Since $\psi[\![^{v_1}/_{c_1}, \ldots, ^{v_n}/_{c_n}]\!] \in Fml_{\mathcal{L}}$, and again considering that T' is a conservative extension of T, it follows that

$$\psi \in T' \text{ iff } \psi[\![^{v_1}/_{c_1}, \ldots, ^{v_n}/_{c_n}]\!] \in T \quad (**)$$

$(*)$ and $(**)$ guarantee that a decision procedure for T' can be derived from any decision procedure for T.

If T' is decidable, then, since it is a conservative extension of T, it follows from theorem B.3.3 that T is decidable as well. ∎

The last two results will be further examined in chapter 2, as part of the study of theories extended by the introduction of defined and undefined non-logical symbols.

In spite of the generality of the decision procedure described in theorems B.3.2 and B.3.1 — it can be applied to *all* negation complete decidable theories, provided that a recursive function that enumerates their axioms is given — it is too inefficient for any conceivable application to mechanical theorem proving. Specialised procedures, therefore, have to be sought for each individual theory.

B.3.2 Elimination of Quantifiers

Two of the standard methods for proving the decidability of a theory involve reducing the set of formulae of the subjacent language into a particular subclass for which the theory is known to be decidable. *Quantifier elimination*, as the name suggests, transforms sentences into quantifier-free formulae. It admits two alternative definitions, the first of which is more directly related to the intuitive meaning of the expression[16].

Definition B.3.1 (Quantifier Elimination – First Version)

Let T be a theory in $\mathcal{L}$.

[16] For sentences, the elimination of quantifiers amounts to the complete removal of variables as well. For this reason, some authors adopt the name *variable elimination* instead; see for instance [Shostak 79], p. 352.

i. T admits elimination of quantifiers in $\mathcal{L}$ for a formula $\phi(v_1, \ldots, v_n) \in Fml_{\mathcal{L}}$ if and only if there is a quantifier-free formula $\psi(v_1, \ldots, v_n)$ in $\mathcal{L}$ such that

$$T \models \phi(v_1, \ldots, v_n) \equiv \psi(v_1, \ldots, v_n)$$

ii. T admits elimination of quantifiers in $\mathcal{L}$ if and only if it admits elimination of quantifiers for every formula of $\mathcal{L}$.

The relevance of this property to the study of decidability stems from the following theorem.

Theorem B.3.5 *Let T be a consistent and recursively axiomatisable theory in $\mathcal{L}$ such that*

i. T admits elimination of quantifiers (first version) in $\mathcal{L}$,
ii. $\mathcal{L}$ contains at least one individual constant c, and
iii. T is negation complete w.r.t. the class Σ of variable-free formulae of $\mathcal{L}$ (i.e. for any $\phi \in \Sigma$, either $\phi \in T$ or $\neg\phi \in T$).

Then T is decidable and negation-complete.

PROOF. As T admits elimination of quantifiers in $\mathcal{L}$, for every sentence $\tau \in Stn_{\mathcal{L}}$, there is a quantifier-free sentence τ' in the same language such that $(\tau \equiv \tau') \in T$. Since T is recursively axiomatisable, all of its elements can be effectively enumerated by a recursive function f. Hence there is a $n \in \mathbb{N}$ such that $f(n) = \ulcorner \tau \equiv \tau' \urcorner$, for some $\tau' \in Fml_{\mathcal{L}}$. Once τ' is identified, since it is variable-free, either it or its negation belongs to T; f can then be used to enumerate the elements of T until one of these sentences is derived. ∎

A direct proof that a theory T admits elimination of quantifiers in $\mathcal{L}$, by the exhibition of a mechanism that generates a quantifier-free formula for each element of $Fml_{\mathcal{L}}$, yields a decision procedure for T, whenever there is a decision procedure for T w.r.t. the class of quantifier-free formulae[17].

In the second version of quantifier elimination, the number of occurrences of quantifiers is reduced rather than altogether eliminated, and the final formula is expressed in terms of a Boolean combination of *basic* formulae. The definition of basic formula is relative to each theory. When they constitute a decidable class[18], the decidability of the theory in the full language follows.

Definition B.3.2 (Quantifier Elimination – Second version)

Let T be a theory in $\mathcal{L}$. T admits elimination of quantifiers in $\mathcal{L}$ if and only if there is

[17] Some cases of indirect proofs for quantifier elimination are examined in [Shoenfield 67], p. 85-6.

[18] Decidable subclasses of formulae are further discussed in appendix C.

a recursive class $\Lambda \subset Fml_{\mathcal{L}}$*, formed by* basic formulae of T in $\mathcal{L}$*, such that, for every formula* $\phi\{v_1, \ldots, v_n\}$ *in* $\mathcal{L}$*, there is a Boolean combination* $\psi\{v_1, \ldots, v_n\}$ *of basic formulae such that*

$$T \models \phi\{v_1, \ldots, v_n\} \equiv \psi\{v_1, \ldots, v_n\}$$

All theories that admit quantifier elimination in the first sense also admit it in the second sense: it suffices to take the class of atomic formulae as basic set. The converse, however, is not always true. For instance, the theory of equality (with one constant symbol, 0) is consistent, negation incomplete and decidable, and is negation complete w.r.t. the class of variable-free formulae (i.e. all the Boolean combinations of $\{0 = 0\}$). Hence, according to theorem B.3.5, it does not admit quantifier elimination, first version, even though it is possible to show that it admits this property in the second version[19].

Proofs of quantifier elimination can be simplified in view of the fact that it suffices to consider the elimination of existential quantifiers from formulae in disjunctive normal form.

Theorem B.3.6 *Let* T *be a theory in* $\mathcal{L}$*.*

i. T *admits elimination of quantifiers (first sense) in* $\mathcal{L}$ *iff it admits quantifier elimination for each formula of the form*

$$(\exists v)(l_1 \wedge \cdots \wedge l_n)$$

where l_i *is either an atomic formula or the negation of an atomic formula.*

ii. T *admits elimination of quantifiers (second sense) in* $\mathcal{L}$ *iff it admits quantifier elimination for each formula of the form* $(\exists v)\phi$*, where* ϕ *is a Boolean combination of basic formulae for* T*.*

To illustrate how this property may lead to the construction of specialised decision procedures, the cases of the theories of the structures $\mathfrak{N}_E$ and $\mathfrak{N}_A$ are next examined.

Theorem B.3.7 *The theory* $T_{\mathfrak{N}_E}$ *of the structure* $\mathfrak{N}_E = \langle \mathbb{N}, 0, s, +, <, \equiv_2, \ldots, \equiv_n, \ldots \rangle$ *admits elimination of quantifiers (first version) in* $\mathcal{L}_{T_{\mathfrak{N}_E}}$*.*

PROOF[20]. Let $\mathcal{L}'_{PrA}$ be the expansion of $\mathcal{L}_{PrA_0}$ generated by the introduction of the binary predicate symbols $<$ and, for each $i \in \mathbb{N}$, $\equiv_i$. The new relations can be defined in the original language as

[19] See [Monk 76], p. 240-2.

[20] Based on [Enderton 72], p. 188-92.

$$
\begin{aligned}
v_1 < v_2 &\equiv (\exists v_3)(v_3 \neq 0 \wedge v_1 + v_3 = v_2) \\
v_1 \equiv_i v_2 &\equiv \bigvee_{0 \leq j < i} (\exists u_1)(\exists u_2)(v_1 = i \times u_1 + j \wedge v_2 = i \times u_2 + j)
\end{aligned}
$$

for each $i \in \mathbb{N}, i \geq 2$. The elimination of existential quantifiers takes place in two stages, one for putting formulae into canonical form, and the second one for the removal of the innermost existential quantifier from canonical formulae. Given a formula ϕ,

$$(\exists v)(l_1 \wedge \cdots \wedge l_n) \quad (*)$$

the reduction to canonical form requires four steps.

i. Elimination of negation, by the application of the equivalences

$$
\begin{aligned}
\neg\,(t_1 = t_2) &\equiv (t_1 < t_2) \vee (t_2 < t_1) \\
\neg\,(t_1 < t_2) &\equiv (t_1 = t_2) \vee (t_2 < t_1) \\
\neg\,(t_1 \equiv_i t_2) &\equiv \bigvee_{0 < j < i} (t_1 \equiv_i t_2 + j)
\end{aligned}
$$

ii. Reduction of the resulting formula to a disjunction of formulae of the form

$$(\exists v)(l'_1 \wedge \cdots \wedge l'_n) \quad (**)$$

where l' is an atomic expression of one of the forms

$$
\begin{aligned}
t(v) &\triangleright u[\not v] \\
t(v) &\triangleright u_1[\not v] \dot{-} u_2[\not v] \\
u[\not v] &< t(v) \\
u_1[\not v] \dot{-} u_2[\not v] &< t(v)
\end{aligned}
$$

and $\triangleright \in \{=, <, \equiv_2, \ldots\}$ and $t(v)$ is either v or kv. The expression $v = u \dot{-} t$ is an abbreviation for $v + t = u$.

iii. Uniformisation of coefficients of v, which requires

(a) finding the least common multiplier k of all coefficients k_i of v in $(**)$,

(b) for each atom where v occurs, defining the multiplier $k'_i = k/k_i$, where $k'_i \in \mathbb{N}$, and

(c) with the help of the following equivalences

$$
\begin{aligned}
t_1 = t_2 &\equiv k't_1 = k't_2 \\
t_1 < t_2 &\equiv k't_1 < k't_2 \\
t_1 \equiv_n t_2 &\equiv k't_1 \equiv_{k'n} k't_2
\end{aligned}
$$

replacing $(**)$ with a new formula $(\exists v)\phi''$ where all the coefficients of v are equal to k.

iv. Bound variable renaming, which requires

(a) identifying a variable v' that does not occur in $(\exists v)\phi''$, and

(b) replacing occurrences of subexpressions kv with the new variable, which can be justified by the equivalence

$$(\exists v)\psi[kv] \quad \equiv \quad (\exists v')(\psi[^{v'}/_{kv}] \wedge v' \equiv_k 0)$$

In the end, $(*)$ has been reduced to a disjunction of formulae of the form

$$(\exists v')(\alpha_1 \wedge \cdots \wedge \alpha_n)$$

where α_i is an atom of one of the forms

$$\begin{array}{rcl} v' & < & t[\not{v}'] \\ t[\not{v}'] & < & v' \\ v' & = & t[\not{v}'] \\ v' & \equiv_m & t[\not{v}'] \end{array}$$

Once it is reorganised to

$$(\exists v')\left(\bigwedge_{i=1}^{p} v' = t_i \wedge \bigwedge_{j=1}^{q} u_j < v' \wedge \bigwedge_{k=1}^{r} v' < w_k \wedge \bigwedge_{l=1}^{s}\bigwedge_{m=1}^{s'} v' \equiv_m z_l\right) \quad (\dagger)$$

the elimination of the existential quantifier has to take three cases into account.

i. $p > 0$ (presence of equations). If $v' = t_i$ is one of such equations, the elimination of the quantifier is achieved by the application of the substitution $\sigma = \{^{t_i}/_{v'}\}$ to the matrix of $(\dagger)$. An additional condition, $0 \leq t_i$, has to be introduced in those cases where t_i has the form $t_{i,1} \dot{-} t_{i,2}$. The resulting formula is

$$\sigma\alpha_1 \wedge \cdots \wedge \sigma\alpha_n \wedge 0 \leq t_i$$

ii. $p = 0$ and $s = 0$ (absence of equalities and equivalences). Since no equality or equivalence of any module occurs in $(\dagger)$, the problem is reduced to the search for a natural number which satisfies the lower and upper bounds established by the inequalities. The formula can then be replaced by the equivalent quantifier-free formula

$$\bigwedge_{j=1}^{q}\bigwedge_{k=1}^{r} \mathrm{s}(u_j) < w_k \wedge \bigwedge_{k=1}^{r} 0 < w_k$$

which is obtained by exhaustive elimination of v' from all pairs of inequalities $\langle u < v', v' < w\rangle$, based on the *PA*-equivalence

$$(\exists v)(u[\not{v}] < v \wedge v < t[\not{v}]) \quad \equiv \quad \mathrm{s}(u) < t$$

iii. $p = 0$ and $s > 0$ (absence of equalities and presence of equivalences modulo m). Let M be the least common multiplier of all moduli of equivalence symbols that occur in (†). Intuitively, if there is a solution for v' in N for a conjunction of such equivalences, it can be found by the inspection of any consecutive list of natural numbers with M elements. In the event $q > 0$ and $r > 0$, it is also necessary to take into account the lower and upper bounds set by the inequalities. The elimination of the existential quantifier is accomplished with the introduction of the quantifier-free formula

$$\bigvee_{i=1}^{M} \bigvee_{j'=1}^{q} \left(\bigwedge_{j=1}^{q} u_j < \mathrm{s}^i(u_{j'}) \wedge \bigwedge_{k=1}^{r} \mathrm{s}^i(u_{j'}) < w_k \wedge \bigwedge_{l=1}^{s} \bigwedge_{m=1}^{s'} \mathrm{s}^i(u_{j'}) \equiv_m z_l \right)$$

∎

Lemma B.3.3 *$T_{\mathfrak{N}_E}$ and $T_{\mathfrak{N}_A}$, where $\mathfrak{N}_A = \langle \mathrm{N}, 0, 1, s, + \rangle$, are decidable.*

PROOF. Since $T_{\mathfrak{N}_E}$ admits quantifier elimination, any sentence τ of $\mathcal{L}'_{PrA}$ can be converted into a quantifier-free sentence, which can be represented as a Boolean combination of atoms of the form

$$\begin{array}{lcl} \mathrm{s}^p(0) & = & \mathrm{s}^q(0) \\ \mathrm{s}^p(0) & < & \mathrm{s}^q(0) \\ \mathrm{s}^p(0) & \equiv_m & \mathrm{s}^q(0) \end{array}$$

where $p, q, m \in \mathrm{N}$. Since each of these atoms is decidable, τ is reducible to a proposition. As a result, $T_{\mathfrak{N}_E}$ is decidable. Also, since it is a conservative extension of the theory of the structure $\langle \mathrm{N}, 0, \mathrm{s}, + \rangle$, according to theorem B.3.3, the theory of $\langle \mathrm{N}, 0, \mathrm{s}, + \rangle$ is decidable. As $T_{\mathfrak{N}_A}$ is a definitional extension of the theory of $\langle \mathrm{N}, 0, \mathrm{s}, + \rangle$, according to theorem B.3.4, $T_{\mathfrak{N}_A}$ is decidable as well. ∎

The theory $T_{\mathfrak{N}_M}$ of the structure $\mathfrak{N}_M = \langle \mathrm{N}, 0, 1, \times \rangle$, on the other hand, admits elimination of quantifiers (first version) in $\mathcal{L}_{SMA}$. Since $T_{\mathfrak{N}_M}$ is negation complete w.r.t. the class of quantifier-free sentences of the underlying language, $T_{\mathfrak{N}_M}$ is decidable[21]. A new arithmetical theory can then be built as an extension of PA_0, $T_{\mathfrak{N}_A}$ and $T_{\mathfrak{N}_M}$.

Lemma B.3.4 *Let Peano arithmetic (PA) be the theory in $\mathcal{L}_{PA} = \{0, 1, s, +, \times\}$ which admits the following axiom set*

$$\Delta_{PA} = \Delta_{PA_0} \cup \Delta_{T_{\mathfrak{N}_A}} \cup \Delta_{T_{\mathfrak{N}_M}}$$

Then, provided that PA_0 is consistent, PA is consistent, recursively axiomatisable and incomplete.

---

[21] See [Mostowski 52], p. 20-3. The original proof of the decidability of this theory, based on an application of the fundamental theorem of arithmetic, can be found in [Skolem 70].

PROOF.

Consistency. Each axiom in Δ_{PA} is valid in $\mathfrak{N} = \langle \mathbb{N}, 0, 1, \mathrm{s}, +, \times \rangle$, since

(a) $\mathfrak{N}_A$ and $\mathfrak{N}_M$ are restrictions of $\mathfrak{N}$, and every formula that is valid in a structure is valid in any expansion of this structure as well, and
(b) the axioms of PA_0 are valid in $\mathfrak{N}$, an assumption that amounts to the consistency of PA_0.

Hence $\mathfrak{N}$ is a model of PA, and therefore PA is consistent[22].

Axiomatisability. PA_0 is recursively axiomatisable, and any decidable theory is recursively axiomatisable (for any theory is an axiom set for itself, and decidable theories are recursive sets of formulae). Since the union of recursive sets is recursive, Δ_{PA} is recursive as well.

Incompleteness. Since PA is a recursively axiomatisable and consistent extension of PA_0, Gödel's first incompleteness theorem is applicable. ∎

$T_{\mathfrak{N}_A}$ and $T_{\mathfrak{N}_M}$ will be respectively denoted as *PrA* (Presburger arithmetic) and *SMA* (strictly multiplicative arithmetic).

B.3.3 Model Completeness

Model completeness is another concept that can be linked to the reduction of formulae of a language into a particular subclass; in this case, the class of universal formulae. Its definition is based on the notion of *elementary extension* of a structure.

Definition B.3.3 (Model Completeness)

Let $\mathfrak{A}$ and $\mathfrak{B}$ be structures of a language $\mathcal{L}$, and let T be a theory in $\mathcal{L}$.

i. $\mathfrak{B}$ *is an* elementary extension *of* $\mathfrak{A}$ *iff*

(a) $|\mathfrak{A}| \subseteq |\mathfrak{B}|$.

(b) For every formula $\phi \in \mathcal{L}$, if α is an assignment in $\mathfrak{A}$, then

$$\mathfrak{A} \models \phi[\alpha] \quad \textit{iff} \quad \mathfrak{B} \models \phi[\alpha]$$

ii. T *is* model complete *iff, for any two models $\mathfrak{A}, \mathfrak{B}$ of T, if $\mathfrak{A}$ is a substructure of $\mathfrak{B}$, then $\mathfrak{B}$ is an elementary extension of $\mathfrak{A}$.*

Real-closed fields, Abelian groups and Boolean algebras are examples of model-complete theories. Amongst the consequences of the above definition is a theorem that provides an alternative definition for model completeness.

---

[22] An informal proof for the fact that formulae that are valid in a structure are also valid in the expansions of this structure can be found in [Shoenfield 67], p. 43-4. The remark that $\mathfrak{N}$ is a model for PA_0 is usually justified on set theoretical grounds; see for instance [Mendelson 87], p. 121.

Lemma B.3.5 *If $\mathfrak{B}$ is an elementary extension of $\mathfrak{A}$, then, for every sentence τ in $\mathcal{L}$,*

$$\mathfrak{A} \models \tau \quad \textit{iff} \quad \mathfrak{B} \models \tau$$

(i.e. $\mathfrak{A}$ and $\mathfrak{B}$ are elementary equivalent).

Theorem B.3.8 *For any consistent theory T in $\mathcal{L}$, T is model complete iff for any formula ϕ in $\mathcal{L}$ there exists a universal formula ψ in the same language such that both contain the same set of free variables and $T \models \phi \equiv \psi$.*

Theorem B.3.8 assigns a role for model completeness in the recognition of decidable theories: any model complete theory in a language $\mathcal{L}$ which is decidable in the class of universal formulae is also decidable in $Fml_{\mathcal{L}}$. As in the case of quantifier elimination, a decision procedure for the corresponding theory can be built whenever a decision procedure for the theory w.r.t. the class of universal sentences is known.

B.4 Conclusions

Total recursive functions provide a mathematical model for the informal notion of effective computability. Decision problems are abstractly characterised by means of a pair of sets $\langle \mathcal{U}, \mathcal{A} \rangle$, such that $\mathcal{U}$ is countable and $\mathcal{A} \subseteq \mathcal{U}$. The decision problem for a first-order theory T in $\mathcal{L}$ is defined as $\langle Fml_{\mathcal{L}}, T \rangle$. Once first-order languages and theories are formalised and the mechanism of Gödel numbering is introduced, decision procedures for first-order theories can be represented by means of recursive functions. Indirect proofs of decidability are based on the inspection of metatheoretical properties of a theory, whereas direct proofs, such as those derived from (constructive proofs of) quantifier elimination and model completeness, lead to the construction of specific decision procedures.

Appendix C

Decidable Subclasses for Undecidable Theories

Due to the existence of undecidable theories, mechanical theorem proving cannot be limited to the study of decision procedures and their computational complexity. Nonetheless, the existence of decidable subclasses of formulae for any first-order theory provides a role for decision procedures even in undecidable domains. Decidable theories, particularly those that admit only highly complex procedures, also benefit from the study of decidable subclasses, since restricted but less complex procedures are relevant from the point of view of efficiency[1].

Decidable subclasses, as well as some of the mechanisms for their exhibition, are investigated in this appendix. Section C.1 discusses two possible roles for decision procedures in undecidable theories and introduces the notion of essential undecidability. Section C.2 defines the concept of decidable subclass and describes some general results concerning their presence in undecidable domains. Some of the classes that are universally present in first-order theories are described in section C.3. Section C.4 examines three special cases of context-free generated decidable subclasses, represented by variable-free atoms, prefix classes and sublanguages. Other special classes, involving the intersection and extension of theories, are surveyed in section C.5.

C.1 Essentially Undecidable Theories

There are two possible levels at which a decision procedure can operate with respect to an undecidable theory T in $\mathcal{L}$: at a decidable extension of T, if there is one, or at a decidable subclass of $\mathcal{L}$. The first option has advantages over the second one: even though the process of extension inevitably introduces as new theorems formulae that do not belong to T, there is a guarantee that all the original theorems can be effectively recognised in the extended theory. Moreover, whenever T has an intended model $\mathfrak{A}$, which is usually more relevant than any particular axiomatic representation

[1] There is an application of this nature, for instance, in *Nqthm*, the Boyer and Moore theorem prover, which is examined in chapter 3.

that partially captures it, extending T does not cause any loss of relevant information, provided that $\mathfrak{A}$ remains a model for the decidable extension.

For this reason, before searching for decidable subclasses of formulae, it is important to rule out the existence of decidable extensions for T. A consistent theory T is *essentially undecidable* iff each one of its consistent extensions is undecidable. Essential undecidability is linked to the *inseparability* of a theory, which in turn is based on the notion of effective inseparability of subsets of natural numbers.

Definition C.1.1 (Recursive and Effective Inseparability)

i. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be subsets of $\mathbb{N}$, *and let*

$$W_1, \ldots, W_n, \ldots$$

be an enumeration of all recursively enumerable subsets of $\mathbb{N}$.

(a) $\mathcal{A}_1$ and $\mathcal{A}_2$ are recursively separable *iff there is a recursive set $\mathcal{A}_3 \subseteq \mathbb{N}$ such that $\mathcal{A}_1 \subseteq \mathcal{A}_3$ and $\mathcal{A}_2 \subseteq \mathbb{N} - \mathcal{A}_3$.*

(b) $\mathcal{A}_1$ and $\mathcal{A}_2$ are recursively inseparable *iff $\mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset$ and $\mathcal{A}_1$ and $\mathcal{A}_2$ are not recursively separable.*

(c) $\mathcal{A}_1$ and $\mathcal{A}_2$ are effectively inseparable *iff $\mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset$ and, for every pair $\langle W_p, W_q \rangle$ of recursively enumerable sets such that $W_p \cap W_q = \emptyset$, $\mathcal{A}_1 \subseteq W_p$ and $\mathcal{A}_2 \subseteq W_q$, there is a total recursive function $f\colon \mathbb{N}^2 \to \mathbb{N}$ such that $f(p,q) \notin W_p \cup W_q$.*

ii. A theory T in $\mathcal{L}$ is inseparable iff $\ulcorner T \urcorner$ and $\ulcorner \tilde{T} \urcorner$ are effectively inseparable, where $\tilde{T}$ is the set of all sentences of $\mathcal{L}$ of the form $\neg\tau$, such that $\tau \in T$.

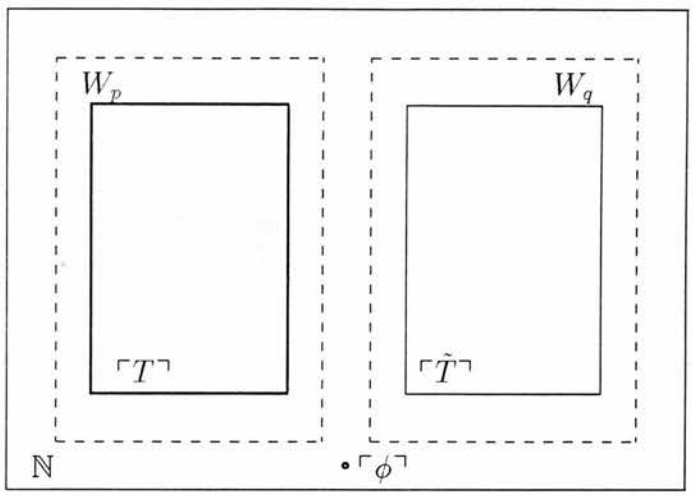
Effective inseparability is stronger than recursive inseparability, in the sense that every pair of effectively inseparable sets is also recursively inseparable, whereas the converse is not always true[2].

Given an undecidable, consistent and recursively axiomatisable theory T in $\mathcal{L}$, the search for a consistent and decidable extension can be informally described as a process of gradual expansion of both T and $\tilde{T}$. $\ulcorner T \urcorner$ and $\ulcorner \tilde{T} \urcorner$, which are disjoint and recursively enumerable, have to be enlarged in a way that their three essential properties,

(a) T is a theory,
(b) $\ulcorner T \urcorner$ and $\ulcorner \tilde{T} \urcorner$ are recursively enumerable, and
(c) T and $\tilde{T}$ are disjoint

are preserved. In the end, whenever possible, the extensions of $\ulcorner T \urcorner$ and $\ulcorner \tilde{T} \urcorner$ embrace $\mathbb{N}$. Since both final sets are recursively enumerable, and one is the complement of the other w.r.t. $\mathbb{N}$, the extended theory obtained is decidable. When T is inseparable, however, the process cannot be carried out, since, according to the definition of effective inseparability, no disjoint pair of recursively enumerable sets containing respectively $\ulcorner T \urcorner$ and $\ulcorner \tilde{T} \urcorner$ covers $\mathbb{N}$, as it is indicated in figure C.1.

[2] See [Monk 76], p. 100.



T	recursively axiomatisable and consistent theory
W_p, W_q	recursively enumerable sets

Figure C.1: Inseparable Theory

Theorem C.1.1 *Any inseparable theory is essentially undecidable.*

PROOF. Since $\ulcorner T \urcorner$ and $\ulcorner \tilde{T} \urcorner$ are effectively inseparable, $T \cap \tilde{T} = \emptyset$ and, as a result, T must be consistent. Assume that there is a consistent decidable extension T' of T. Then $\ulcorner T' \urcorner$ must be recursive, as well its complement, $\mathbb{N} - \ulcorner T' \urcorner$. Since T' is consistent and $T \subset T'$, then $\tilde{T} \cap T' = \emptyset$. As a result, there are disjoint recursively enumerable sets, $\ulcorner T' \urcorner$ and its complement, such that $\ulcorner T \urcorner \subseteq \ulcorner T' \urcorner$ and $\ulcorner \tilde{T} \urcorner \subseteq \mathbb{N} - \ulcorner T' \urcorner$, in conflict with the inseparability of T. Hence T has no consistent decidable extension, and is essentially undecidable. ∎

For inseparable theories, therefore, it is meaningless to search for decidable extensions, and the possible role of decision procedures is limited to subclasses of formulae of $\mathcal{L}$. Peano arithmetic is just one example of such theories[3].

[3] The inseparability of *PA* follows from Gödel's first incompleteness theorem, considering that, for every recursively axiomatisable extension of *PA*, if it is consistent, it is possible to effectively exhibit a sentence that is undecidable in the extended theory.

C.2 Decidable Subclasses of Formulae

From an informal viewpoint, a decidable subclass for a theory T in $\mathcal{L}$ is a set of formulae $\Sigma \subseteq Fml_{\mathcal{L}}$ such that there is an effective mechanism for determining whether an element of Σ is a theorem of T or not. An important element, however, is missing: Σ itself must be effective, i.e. the recognition of its elements must be performed within a finite amount of time and computation[4].

C.2.1 A Taxonomy for Decidable Subclasses

The decidability of a theory T in a class of formulae other than the full (effectivised) language where it is defined therefore requires

(i) an effective procedure for determining whether an element of $Fml_{\mathcal{L}}$ belongs to the subclass (i.e. the subclass must be decidable in $Fml_{\mathcal{L}}$), and
(ii) an effective procedure for discerning theorems from non-theorems in this subclass[5].

Definition B.2.5, which introduces the notion of decidable subsets, is not adequate for the current purposes, since it does not take into account both procedures required for the characterisation of a subclass that is decidable for a theory. A generalised definition for the decidability of a set avoids these difficulties.

Definition C.2.1 (Decidable Subclass)

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be subsets of a universe $\mathcal{U}$, and let $g_{\mathcal{U}}$ be a Gödel function for $\mathcal{U}$. $\mathcal{A}_1$ is (recursively) decidable for $\mathcal{A}_2$ in $\mathcal{U}$ *(or $\mathcal{A}_2$ is decidable in $\mathcal{A}_1$) iff there exists a recursive function $f_{\mathcal{A}_1}$ such that*

$$f_{\mathcal{A}_1}(g_{\mathcal{U}}(u)) = \begin{cases} 0 & \text{if } u \notin \mathcal{A}_1 \\ 1 & \text{if } u \in \mathcal{A}_1 - \mathcal{A}_2 \\ 2 & \text{if } u \in \mathcal{A}_1 \cap \mathcal{A}_2 \end{cases}$$

where $u \in \mathcal{U}$. $f_{\mathcal{A}_1}$ then is a decision procedure for $\mathcal{A}_1$ w.r.t. $\mathcal{A}_2$.

In the context of the decision problem for a theory T, $\mathcal{A}_2$ corresponds to T, while $\mathcal{U}$ is the set of formulae of the underlying language[6]. The function $f_{\mathcal{A}_1}$ effectively recognises three disjoint subsets of $Fml_{\mathcal{L}}$:

---

[4] Otherwise, given any theory T, since T is a subset of $Fml_{\mathcal{L}}$, it could be chosen as a 'decidable subclass' for itself, since there is a recursive function for the effective recognition of theorems in this class, the constant function $f(\ulcorner \phi \urcorner) = 1$, which simply indicates that every element of T is a T-theorem. However, there is no effective way of determining whether an element of $Fml_{\mathcal{L}}$ belongs to this class or not in the case of undecidable theories.

[5] See [Church 56], p. 246.

[6] To see how this definition generalises definition B.2.5, it suffices to consider that a set $\mathcal{A}_1$ is decidable in $\mathcal{U}$ (according to definition B.2.5) iff $\mathcal{A}_1$ is decidable for $\mathcal{U}$ in $\mathcal{U}$ (according to definition C.2.1 above). Both definitions, however, will be used throughout this study.

(a) the formulae that do not belong to the decidable subclass $\mathcal{A}_1$ (in which case $f_{\mathcal{A}_1}(\ulcorner\phi\urcorner) = 0$),
(b) the formulae of $\mathcal{A}_1$ that are not T-theorems ($f_{\mathcal{A}_1}(\ulcorner\phi\urcorner) = 1$), and
(c) the formulae of $\mathcal{A}_1$ that are theorems of T ($f_{\mathcal{A}_1}(\ulcorner\phi\urcorner) = 2$).

Example C.2.1 *Let $T_0^{\mathcal{L}}$ be the predicate calculus in $\mathcal{L}$.*

i. The subclass formed by every equality containing syntactically identical terms, i.e.

$$\Sigma_I = \{\phi \in Fml_{\mathcal{L}} \mid \phi \stackrel{s}{=} (t{=}t)\}$$

is decidable for $T_0^{\mathcal{L}}$ in $Fml_{\mathcal{L}}$, since every element of Σ_I is a theorem in $T_0^{\mathcal{L}}$.

ii. The subclass

$$\Sigma_{II} = Fml_{\mathcal{L}^*}$$

where $\mathcal{L}^*$ is the sublanguage of $\mathcal{L}$ obtained by the removal of all non-logical symbols, is decidable for $T_0^{\mathcal{L}}$ in $Fml_{\mathcal{L}}$, since the theory of pure equality is decidable[7]*.*

iii. Since $\mathcal{L}$ admits structures of any finite (non-nil) cardinality, the sentences

$$\begin{array}{ll}
\tau_1 & (\exists v_1)(v_2)(v_1 = v_2) \\
\tau_2 & (\exists v_1)(\exists v_2)(v_3)(v_1 \neq v_2 \wedge (v_3 = v_1 \vee v_3 = v_2)) \\
\vdots & \\
\tau_n & (\exists v_1)\ldots(\exists v_n)(v_{n+1})(\bigwedge_{1\leq(i,j)\leq n}^{i\neq j} v_i \neq v_j \wedge \bigvee_{i=1}^{n} v_{n+1} = v_i) \\
\vdots &
\end{array}$$

each of which states that the number of distinct elements in the universe of the structure is $1, 2, \ldots, n, \ldots$, are undecidable in $T_0^{\mathcal{L}}$. Given that, when a sentence is undecidable in a theory, neither it nor its negation belongs to the theory, it follows that the subclass

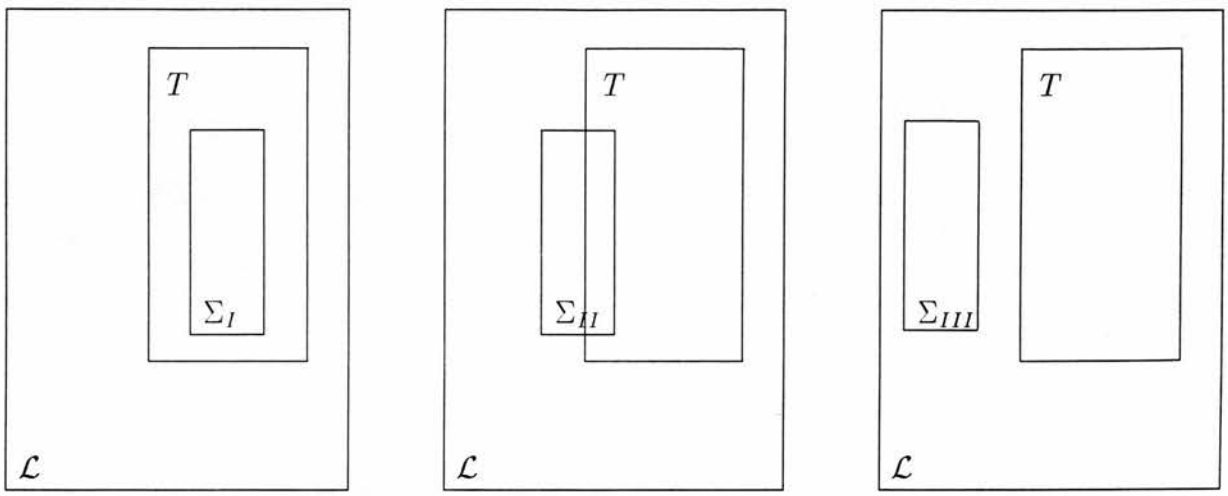
$$\Sigma_{III} = \{\tau_1, \neg\tau_1, \tau_2, \neg\tau_2, \ldots, \tau_n, \neg\tau_n, \ldots\}$$

is decidable for $T_0^{\mathcal{L}}$ in $Fml_{\mathcal{L}}$. □

These examples illustrate the existence of three distinct types of decidable subclasses.

$$\begin{array}{lrcl}
\text{Type I} & \Sigma_I \cap T & = & \Sigma_I \\
\text{Type II} & \Sigma_{II} \cap T & \subset & \Sigma_{II} \\
\text{Type III} & \Sigma_{III} \cap T & = & \emptyset
\end{array}$$

[7] Also, $T_0^{\mathcal{L}}$ is a conservative extension of the theory of pure equality; section C.5.1 discusses the role of subtheories as decidable subclasses.



I II III

Figure C.2: Types of Decidable Subclasses

Each of them is represented by a Venn diagram in figure C.2. Subclasses of type I are entirely made up of theorems, type III classes contain only non-theorems. In both cases, there is a trivial decision procedure for the recognition of theorems and non-theorems: for Σ_I,

$$f_{\Sigma_I}(\ulcorner\phi\urcorner) = 2 \times g(\ulcorner\phi\urcorner) \quad (*)$$

where g is a decision procedure for Σ_I w.r.t. $Fml_{\mathcal{L}}$, whereas for Σ_{III}

$$f_{\Sigma_{III}}(\ulcorner\phi\urcorner) = g(\ulcorner\phi\urcorner) \quad (**)$$

where g is a decision procedure for Σ_{III} w.r.t. $Fml_{\mathcal{L}}$. Concerning classes of type II, the proof of decidability of T in a subset Σ_{II} may be broken into two parts, concerning

(i) the decidability of Σ_{II} in $Fml_{\mathcal{L}}$, and
(ii) the decidability of $T \cap \Sigma_{II}$ in Σ_{II}.

If g is a decision procedure for Σ_{II} w.r.t. $Fml_{\mathcal{L}}$, and h is a decision procedure for $T \cap \Sigma_{II}$ w.r.t. Σ_{II}, a decision procedure $f_{\Sigma_{II}}$ for T w.r.t. Σ_{II} can be defined as

$$f_{\Sigma_{II}}(\ulcorner\phi\urcorner) = g(\ulcorner\phi\urcorner) + h(\ulcorner\phi\urcorner) \quad (***)$$

The next lemma guarantees that these three procedures satisfy definition C.2.1.

Lemma C.2.1 *Functions $f_{\Sigma_I}, f_{\Sigma_{II}}$ and $f_{\Sigma_{III}}$ defined above are decision procedures for T w.r.t. Σ_I, Σ_{II} and Σ_{III}.*

PROOF.

i. Considering that g is a decision procedure for Σ_I in $Fml_{\mathcal{L}}$, from $(*)$ it follows that f_{Σ_I} can be represented as

$$f_{\Sigma_I}(\ulcorner\phi\urcorner) = \begin{cases} 0 & \text{if } \phi \notin \Sigma_I \\ 1 & \text{if } \phi \in \emptyset = (\Sigma_I - T) \\ 2 & \text{if } u \in \Sigma_I = \Sigma_I \cap T \end{cases}$$

Hence, according to definition C.2.1, f_{Σ_I} is a decision procedure for T w.r.t. Σ_I.

ii. If g is a decision procedure for Σ_{II} w.r.t. $Fml_{\mathcal{L}}$ and h is a decision procedure for $(T \cap \Sigma_{II})$ w.r.t. Σ_{II}, from $(**)$ it follows that

$$f_{\Sigma_{II}}(\ulcorner\phi\urcorner) = \begin{cases} 0 & \text{if } \phi \notin \Sigma_{II} \text{ and } \phi \notin (T \cap \Sigma_{II}) \\ 1 & \text{if } \phi \in \Sigma_{II} \text{ and } \phi \notin (T \cap \Sigma_{II}) \\ & \text{or } \phi \notin \Sigma_{II} \text{ and } \phi \in (T \cap \Sigma_{II}) \\ 2 & \text{if } \phi \in \Sigma_{II} \text{ and } \phi \in (T \cap \Sigma_{II}) \end{cases}$$

which can be simplified to

$$f_{\Sigma_{II}}(\ulcorner\phi\urcorner) = \begin{cases} 0 & \text{if } \phi \notin \Sigma_{II} \cup (T \cap \Sigma_{II}) = \Sigma_{II} \\ 1 & \text{if } \phi \in (\Sigma_{II} - (T \cap \Sigma_{II})) \cup \emptyset = (\Sigma_{II} - T) \\ 2 & \text{if } \phi \in \Sigma_{II} \cap (T \cap \Sigma_{II}) = (T \cap \Sigma_{II}) \end{cases}$$

Therefore $f_{\Sigma_{II}}$ is a decision procedure for T w.r.t. Σ_{II}.

iii. If g is a decision procedure for Σ_{III} in $Fml_{\mathcal{L}}$, from $(***)$ it follows that

$$f_{\Sigma_{III}}(\ulcorner\phi\urcorner) = \begin{cases} 0 & \text{if } \phi \notin \Sigma_{III} \\ 1 & \text{if } \phi \in \Sigma_{III} = (\Sigma_{III} - T) \\ 2 & \text{if } \phi \in \emptyset = (\Sigma_{III} \cap T) \end{cases}$$

and therefore $f_{\Sigma_{III}}$ is a decision procedure for T w.r.t. Σ_{III}. ∎

Under certain circumstances, classes of type I and III can be expanded into classes of type II by the operation of closure with respect to negation.

i. If T is consistent, any class of type I can be expanded into one of type II, since the negation of a theorem is a non-theorem.

ii. A class Σ_{III} of type III can be expanded at least in two cases:

   (a) when T is negation complete, in which case the negation of every formula in Σ_{III} must be an element of T, and

(b) when all the elements of Σ_{III} are T-unsatisfiable, in which case their negations must be T-valid, i.e. elements of the theory[8].

Decidable subclasses can be classified along other lines as well, as for instance the complexity of the mechanisms required for their generation. Grammars provide a suitable framework for the introduction of three groups of effectively computable sets.

C.2.2 The Chomsky Hierarchy

Recursive functions can operate both as *recognisers* and *generators* for the elements of computable sets, given that the image of any total recursive function is recursively enumerable. Since recursive functions are only one amongst several mathematical models for informal computability, set generation can be based on alternative concepts, one of which corresponds to the notion of *grammar*.

Grammars operate with a set of symbols or alphabet, which are used in the construction of finite sequences of symbols or *strings*. The transformation of a string into a new one is achieved through the application of rules. Distinct types of grammars are delimited once restrictions are imposed upon the form of their rules[9].

Definition C.2.2 (Grammars)

i. *A* grammar *is a quadruple* $G = \langle V, \Sigma, R, S \rangle$ *such that* V *is the set of* variables, Σ *is the* alphabet, $S \in V$ *is the* start variable, *and* R *is a finite set of rules.* Σ^* *and* $(\Sigma \cup V)^*$ *respectively denote the set of all finite strings (including the empty string) made up of elements of* Σ *and* $\Sigma \cup V$. *A* rule *is any pair of strings of the form* $\langle s_1 v s_2,\ s_3 \rangle$, *where* $s_1, s_2, s_3 \in (\Sigma \cup V)^*$, *and* $v \in V$.

ii. *A string* $s \in \Sigma^*$ *is generated by* G *iff there is a finite sequence of strings of* $(\Sigma \cup V)^*$,

$$s_1, \ldots, s_n,$$

such that s_1 *is the start variable* S, s_n *is the generated string* s, *and for, each string* s_i *in the sequence,* $1 \leq i < n$, *there is a rule* $\langle s', s'' \rangle$ *in* R *such that* s' *is a substring of* s_i, *and* s_{i+1} *is obtained from* s_i *by the replacement of an occurrence of* s' *with* s''.

iii. *Let* $G = \langle V, \Sigma, R, S \rangle$ *be a grammar.*

(a) G *is* context-sensitive *iff, for every rule* $\langle s, s' \rangle \in R$, *the length of* s *is less than or equal to the length of* s', *where the length of a string is the number of symbols that occur in it.*

---

[8] If, on the other hand, T is negation incomplete, there is a formula that is undecidable in T (i.e. neither it nor its negation is a T-theorem): as a result, it is not possible to guarantee that the negation of a formula of Σ_{III} occurs in T. Also, if Σ_{III} contains a formula that is T-satisfiable, its negation, which is T-invalid, does not occur in T either.

[9] See [Lewis & Papadimitriou 81], p. 224-7, [Manna 74], p. 41-3 and [Hopcroft & Ullman 79], p. 79-82,217.

(b) G *is* context-free *iff, for every rule* $\langle s, s' \rangle \in R$, $s \in V$ *(and* $s' \in (\Sigma \cup V)^*$*).*

(c) G *is* regular *iff, for every rule* $\langle s, s' \rangle \in R$, $s \in V$, *and either* $s' \in \Sigma^*$ *or* s' *has the form* $s'_1 s'_2$, *where* $s'_1 \in \Sigma^*$ *and* $s'_2 \in V$.

The subset of strings of Σ^* generated by a grammar G is the *class of strings (or expressions) generated by* G. When the universe of all classes generated by each type of grammar is taken into account, the Chomsky hierarchy is established.

regular generated classes
↓
context-free generated classes
↓
context-sensitive generated classes
↓
effectively computable classes

Each group is properly contained in its successor; all classes generated by any of these three types of grammars are therefore effectively computable. An important property of the hierarchy is that the typical procedure that computes a class in one group is less complex than the typical procedure for computing a class in any of its successors[10]. Context-free grammars have a particular interest in the study of decision problems, since they can generate most of the targeted subclasses of formulae.

Example C.2.2 *Let* $G = \langle V, \Sigma, R, S \rangle$ *be a grammar such that*

$$\begin{array}{lcl} V & = & \{\ fml,\ atm,\ trm,\ cte,\ var\ \} \\ \Sigma & = & \{(,), \forall, \exists, \neg, \wedge, \vee, \supset, \equiv, =, \times, 0, 1, x_1, x_2, \ldots, x_n, \ldots\} \\ S & = & \{fml\} \end{array}$$

and R *is the set of rules*[11]

$$\begin{array}{rcl} fml & := & atm \mid (\forall var) fml \mid (\exists var) fml \mid \neg fml \mid fml \wedge fml \mid fml \vee fml \mid fml \supset fml \mid fml \equiv fml \\ atm & := & (trm = trm) \\ trm & := & var \mid cte \mid trm \times trm \\ cte & := & 0 \mid 1 \\ var & := & x_1 \mid x_2 \mid \ldots \mid x_n \mid \ldots \end{array}$$

Since each of the above rules belongs to $V \times (\Sigma \cup V)^*$, G *is context-free. Two examples of strings generated by* G *are indicated below.*

$$fml, \quad atm, \quad (trm = trm), \quad (var = trm), \quad (var = cte), \quad (var = 1), \quad (x_4 = 1)$$

[10] See [Hopcroft & Ullman 79], p. 285.

[11] Backus Naur notation, described e.g. in [Sommerhalder & Van Westrhenen 88], p. 38-40, has been chosen due to its simplicity.

$$fml,\quad (\exists var)fml,\quad (\exists x_3)fml,\quad (\exists x_3)\neg fml,\quad (\exists x_3)\neg atm,\quad (\exists x_3)\neg(trm = trm),$$
$$(\exists x_3)\neg(trm = var),\quad (\exists x_3)\neg(var = var),\quad (\exists x_3)\neg(var = var),\quad (\exists x_3)\neg(x_{10} = var),$$
$$(\exists x_3)\neg(x_{10} = x_8)$$

It is possible to prove that G generates the set of formulae of $\mathcal{L}_{SMA} = \{0, 1, \times\}$. □

Since context-free generated classes cover just a fraction of effectively computable domains, a stronger universe of classes is examined in chapter 4.

C.3 General Decidable Subclasses

Axiomatisable first-order theories admit both *logical* decidable subclasses, entirely composed of logically valid formulae, as well as *non-logical* subclasses, which also contain proper non-logical theorems. Concerning the second group, if T is a recursively axiomatisable theory in $\mathcal{L}$, there is a set Δ of axioms for T which is decidable for T in $Fml_{\mathcal{L}}$, considering that

(i) $\Delta \subseteq T$ (type I), and
(ii) $\ulcorner\Delta\urcorner$ is recursive, for some Δ.

With respect to the first group, since any theory includes all the valid formulae of its underlying language (i.e. $T_0^{\mathcal{L}} \subseteq T$), some of the decidable subclasses for $T_0^{\mathcal{L}}$ in $Fml_{\mathcal{L}}$ can be used to identify decidable subclasses for any theory of the same language. Certain prefix classes are known to be decidable for $T_0^{\mathcal{L}}$, as for instance the classes of

i. Universal formulae, which can be generated by a context-free grammar having the following rules

$$\begin{aligned}
unv_1 &:= mtx|(\forall var)unv_1\\
mtx &:= atm|\neg mtx|mtx \vee mtx|mtx \wedge mtx|mtx \supset mtx|mtx \equiv mtx\\
atm &:= (trm{=}trm)|p_{1,1}(trm)|\ldots|p_{i,j}(trm_1,\ldots,trm_j)|\ldots\\
trm &:= var|cte|f_{1,1}(trm)|\ldots|f_{i,j}(trm_1,\ldots,trm_j)|\ldots\\
cte &:= c_1|\ldots|c_i|\ldots\\
var &:= v_1|\ldots|v_i|\ldots
\end{aligned}$$

ii. Formulae of the form

$$(v_1)\ldots(v_n)(\exists u_1)\ldots(\exists u_m)\phi$$

$n, m \geq 0$, where the matrix ϕ is free from occurrences of function symbols, individual constant symbols and equality. The rules required in this case are

$$\begin{aligned}
unv_2 &:= ext|(\forall var)unv_2\\
ext &:= mtx|(\exists var)ext\\
mtx &:= atm|\neg mtx|mtx \vee mtx|mtx \wedge mtx|mtx \supset mtx|mtx \equiv mtx\\
atm &:= p_{1,1}(var)|\ldots|p_{i,j}(var_1,\ldots,var_j)|\ldots\\
var &:= v_1|\ldots|v_i|\ldots
\end{aligned}$$

iii. Formulae of the form

$$(v_1)\ldots(v_n)(\exists u_1)(\exists u_2)(w_1)\ldots(w_m)\phi$$

$n, m \geq 0$, where the matrix ϕ is free from occurrences of function symbols, individual constant symbols and equality, which can be generated by the rules

$$\begin{array}{rcl} unv_4 & := & (\exists var)(\exists var)unv_3|(\forall var)unv_4 \\ unv_3 & := & mtx|(\forall var)unv_3 \\ mtx & := & atm|\neg mtx|mtx \vee mtx|mtx \wedge mtx|mtx \supset mtx|mtx \equiv mtx \\ atm & := & p_{1,1}(var)|\ldots|p_{i,j}(var_1,\ldots,var_j)|\ldots \\ var & := & v_1|\ldots|v_i|\ldots \end{array}$$

A proof for the decidability of the class of universal formulae can be derived from the decidability of the same class in an equational theory. The decidability of the other two classes follows from the fact that they share the *finite model property*, according to which a set of formulae is satisfiable iff it is satisfiable in a finite structure. Other prefix decidable subclasses for $T_0^{\mathcal{L}}$ are defined by restrictions imposed on the form of the matrix, as for instance the arity of predicate symbols[12].

For each of the three logical decidable subclasses Λ above, let g_Λ denote a decision procedure for $T_0^{\mathcal{L}}$ w.r.t. Λ. The derived function

$$f_{\Sigma_I}(\ulcorner\phi\urcorner) = \begin{cases} g_\Lambda(\ulcorner\phi\urcorner) & \text{if even}(g_\Lambda(\ulcorner\phi\urcorner)) \\ 0 & \text{otherwise} \end{cases}$$

then represents a decision procedure for any theory T in $Fml_{\mathcal{L}}$ w.r.t. a decidable subclass Σ_I of type I, where Σ_I contains only the logically valid formulae of Λ: although valid formulae are theorems in any theory of the same language, invalid formulae (i.e. those for which g_Λ returns 1) do not have to correspond to non-theorems of T, since T is a non-conservative extension of $T_0^{\mathcal{L}}$. Theories and their extensions are further examined in section C.5.2.

Another example of logical subclass that is decidable for any first-order theory is composed of propositionally valid formulae. The propositional form of a formula is generated through the replacement of its propositional components with propositional symbols.

Definition C.3.1 (Propositional Form)

Let ϕ be a formulae of $\mathcal{L}$.

i. The set Φ of the propositional components of ϕ is recursively defined as follows.

(a) If ϕ is atomic, or if it has the form $Qv\psi$, where Q is a quantifier, then $\Phi = \{\phi\}$.

[12] See [Ben-Ari 93], p. 142-145. Details about other prefix classes can be found for instance in [Dreben & Goldfarb 79] and [Gabbay & Shehtman 93], p. 800-1.

(b) *If* ϕ *has the form* $\neg\psi$ *and* Φ' *is the set of propositional components of* ψ, *then* $\Phi = \Phi'$.

(c) *If* ϕ *has the form* $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $\psi_1 \supset \psi_2$ *or* $\psi_1 \equiv \psi_2$ *and* Φ_1 *and* Φ_2 *are respectively the sets of propositional components of* ψ_1 *and* ψ_2, *then* $\Phi = \Phi_1 \cup \Phi_2$.

ii. If $\Phi = \{\phi_1, \ldots, \phi_n\}$ *is the set of propositional components of* ϕ, *then its* propositional form ϕ^P *is the proposition*

$$\phi[\![^{p_1}/_{\phi_1}, \ldots, ^{p_n}/_{\phi_n}]\!]$$

where $p_1, \ldots, p_n$ *are distinct propositional symbols.*

iii. ϕ *is* propositionally valid *iff* ϕ^P *is valid.*

The class of propositionally valid formulae of $\mathcal{L}$ is decidable for any theory of $\mathcal{L}$.

Theorem C.3.1 *Let* T *be a theory in* $\mathcal{L}$, *and let* Σ *be the set of all propositionally valid formulae of* $\mathcal{L}$. *Then*

(i) $\Sigma \subset T$ *(type I), and*
(ii) Σ *is decidable for* T *in* $Fml_{\mathcal{L}}$.

PROOF.

i. Let $\phi(v_1, \ldots, v_n)$ be a (logically) invalid formula of $\mathcal{L}$ such that ϕ^P is propositionally valid. Then there is a structure $\mathfrak{A}$ for $\mathcal{L}$ and an assignment $\alpha = \{^{a_1}/_{v_1}, \ldots, ^{a_n}/_{v_n}\}$ such that

$$\mathfrak{A} \not\models \phi(v_1, \ldots, v_n)[\alpha]$$

Let $\mathcal{L}' = \mathcal{L} \cup \{c_1, \ldots, c_n\}$, and let $\mathfrak{A}'$ be a structure for $\mathcal{L}'$ that expands $\mathfrak{A}$ such that

$$c_i^{\mathfrak{A}'} = \boldsymbol{a}_i, \quad 1 \leq i \leq n$$

Then $\mathfrak{A}' \not\models \phi'$, where $\phi' \stackrel{s}{=} \phi(^{c_1}/_{v_1}, \ldots, ^{c_n}/_{v_n})$ is a sentence. Let $\Phi' = \{\phi'_1, \ldots, \phi'_m\}$ be the set of propositional components of ϕ', and let $F\colon \Phi' \to \{\top, \bot\}$ be such that $F(\phi'_i) = \top$ if $\mathfrak{A}' \models \phi'_i$, and $F(\phi'_i) = \bot$ otherwise. Clearly, $\phi'[\![^{F(\phi'_1)}/_{\phi_1}, \ldots, ^{F(\phi'_n)}/_{\phi_n}]\!]$ is also invalid in $\mathfrak{A}'$. As ϕ and ϕ' share the same propositional form, the propositional assignment $\{^{F(\phi'_1)}/_{\phi_1}, \ldots, ^{F(\phi'_n)}/_{\phi_n}\}$ must turn ϕ^P into $\bot$, in contradiction with the hypothesis. Hence, each formula in Σ must be (logically) valid and, for this reason, also an element of T.

ii. Both the construction of propositional forms and the establishment of validity for propositions are effective. If g is a recursive function that generates the propositional form ϕ^P for each formula ϕ in $\mathcal{L}$, and h is a decision procedure for

validity in the propositional calculus, then $h \circ g$ represents a decision procedure for Σ w.r.t. $Fml_{\mathcal{L}}$. According to lemma C.2.1, the function f_Σ such that

$$f_\Sigma(\ulcorner \phi \urcorner) = 2 \times h(g(\ulcorner \phi \urcorner))$$

is a decision procedure for T w.r.t. Σ, which has type I. ∎

A stronger result (i.e. a larger decidable subclass) is obtained if the set of propositional components of a formula is limited to distinct subformulae modulo bound variable renaming: formulae such as $(\exists x)\phi(x) \vee \neg(\exists y)\phi(y)$ could then be identified as propositionally valid.

C.4 Context-free Generated Classes

Apart from general subclasses, there are also those whose decidability is restricted to theories that satisfy certain conditions. The class of variable-free atoms, some prefix classes and the set of formulae of a sublanguage are just a few examples.

C.4.1 Atomic Sentences

For some theories, the class of variable-free atomic formulae admits a decision procedure entirely based on syntactic properties of atoms. This is the case, for instance, of the theory of groups, defined in the language $\mathcal{L}_G = \{0, -, +\}$, which admits a set of five axioms.

$$\begin{aligned} v_1 + (v_2 + v_3) &= (v_1 + v_2) + v_3 \\ v + 0 &= v \\ 0 + v &= v \\ v + (-v) &= 0 \\ (-v) + v &= 0 \end{aligned}$$

Any variable-free term is effectively reducible in this theory to the constant symbol 0, making therefore any variable-free atom reducible to $0 = 0$, which is logically valid.

Variable-free atoms are also decidable for PA: given an atomic formula in $\mathcal{L}_{PA}$, each variable-free term can be effectively reduced to the form $\mathrm{s}^n(0)$, where $\mathrm{s}^0(0) = 0$ and $\mathrm{s}^{n+1}(0) = \mathrm{s}(\mathrm{s}^n(0))$, by the application of the axioms for sum and multiplication as rewrite rules. Any atom in this class is then equivalent to

$$\mathrm{s}^n(0) = \mathrm{s}^m(0)$$

where $n, m \in \mathbb{N}$. The above atom is valid in PA iff $n = m$.

C.4.2 Prefix Classes

As already discussed in section C.3, decidable prefix classes of type II for $T_0^{\mathcal{L}}$ provide prefix classes of type I for any theory in the same language. Under certain circumstances, theories other than $T_0^{\mathcal{L}}$ also admit prefix classes of type II. For instance, the validity of a universal formula can be effectively decided in certain finitely axiomatisable theories.

Theorem C.4.1 *Let $\mathcal{L}^*$ be a pure first-order language that does not contain function or individual constant symbols. If T is a finitely axiomatisable theory in $\mathcal{L}^*$ that admits a set of quantifier-free axioms, then the class of quantifier-free formulae of $\mathcal{L}^*$ is decidable for T in $Fml_{\mathcal{L}^*}$.*

PROOF. Let $\psi \in Fml_{\mathcal{L}^*}$ be quantifier-free, and let $\Delta = \{\delta_1, \ldots, \delta_n\}$ be a set of quantifier-free axioms for T. From the definitions of logical consequence and validity in a structure,

$$\{\delta_1, \ldots, \delta_n\} \models \psi \quad \text{iff} \quad \{[\delta_1], \ldots, [\delta_n]\} \models [\psi]$$

and, by iterated applications of the deduction theorem,

$$[\delta_1], \ldots, [\delta_n] \models [\psi] \quad \text{iff} \quad \models [\delta_1] \supset \cdots \supset [\delta_n] \supset [\psi]$$

Hence, the set

$$\{\psi \in Fml_{\mathcal{L}^*} \mid \Delta \models \psi \ \ \& \ \ \psi \text{ is quantifier-free}\}$$

is decidable for T in $Fml_{\mathcal{L}^*}$ iff

$$\{\tau \in Stn_{\mathcal{L}^*} \mid \tau \stackrel{s}{=} [\delta_1] \supset \cdots \supset [\delta_n] \supset [\psi]\}$$

is also decidable for T in $Fml_{\mathcal{L}^*}$. Without loss of generality, if $FrVar(\psi) = \{v_1, \ldots, v_n\}$ and $FrVar(\Delta) = \{u_1, \ldots, u_l\}$, it is possible to assume that $FrVar(\psi) \cap FrVar(\Delta) = \emptyset$ (i.e. ψ does not share any variable with the axioms of T), otherwise bound variables (in $[\psi], [\delta_1], \ldots, [\delta_n]$) can be renamed. Given the sequence of logically equivalent sentences,

$$[\delta_1] \supset \cdots \supset [\delta_n] \supset [\psi]$$

$$[[\neg\delta_1]] \vee \cdots \vee [[\neg\delta_n]] \vee [\psi]$$

$$(v_1) \ldots (v_m)(((\exists u_1) \ldots (\exists u_l)(\neg\delta_1 \vee \cdots \vee \neg\delta_n)) \vee \psi)$$

$$(v_1) \ldots (v_m)(\exists u_1) \ldots (\exists u_l)(\delta_1 \supset \cdots \supset \delta_n \supset \psi),$$

it follows that

$$\Delta \models \psi \quad \text{iff} \quad \models (v_1) \ldots (v_m)(\exists u_1) \ldots (\exists u_l)(\delta_1 \supset \cdots \supset \delta_n \supset \psi)$$

As the prefix of the final expression above defines a decidable class for $T_0^{\mathcal{L}^*}$, it follows that the class of universal formulae is decidable for T in $Fml_{\mathcal{L}^*}$. ∎

Another result guarantees the decidability of the class of universal formulae for other theories.

Theorem C.4.2 *Let T be a theory in $\mathcal{L}$, such that the class of all formulae of $Fml_{\mathcal{L}}$ of the form*

$$\gamma_1 \vee \cdots \vee \gamma_n$$

where γ_i is a literal (i.e. an atomic formula or the negation of an atomic formula), is decidable for T. Then the class of quantifier-free formulae of $\mathcal{L}$ is also decidable for T.

PROOF. Let ϕ be a quantifier-free formula of $\mathcal{L}$, and let $\beta_1, \ldots, \beta_n$ be its atomic subformulae. ϕ is T-valid iff, for every model $\mathfrak{A}$ of T, $\mathfrak{A} \models \phi$, i.e. for every assignment α in $\mathfrak{A}$, $\mathfrak{A} \models \phi[\alpha]$. The validity of $\phi[\alpha]$ in $\mathfrak{A}$ can be established in two stages.

i. for each β_i, it is first determined whether $\mathfrak{A} \models \beta_i[\alpha]$. When this is the case, each occurrence of β_i in $\phi[\alpha]$ can be replaced with the propositional constant $\top$; otherwise $\phi[\alpha]$ is replaced with $\bot$.

ii. the validity of the proposition generated from $\phi[\alpha]$ in (i) can be effectively assessed by a decision procedure for the propositional calculus.

Step (i) therefore results from a *variable assignment* in $\mathfrak{A}$, whereas step (ii) is a consequence of a *propositional assignment* to the atomic components of $\phi[\alpha]$. The above process has to be repeated for every assignment α in $\mathfrak{A}$. However, in spite of the fact that there may be infinite many assignments, considering that ϕ has only n atoms, there are only 2^n propositions derivable from ϕ. Whenever it is possible to determine which are all the compatible propositional assignments derivable from any variable assignment in $\mathfrak{A}$, the validity of ϕ in $\mathfrak{A}$ can be reduced to a propositional problem, represented by the second step[13].

Since the original problem involves every model of T, it is necessary to determine all compatible propositional assignments generated from them. Let $\mathcal{V}_j$: $\{\beta_1, \ldots, \beta_n\} \rightarrow \{\top, \bot\}$ represent a possible propositional assignment to the atomic formulae in ϕ. Assume ψ_j is a conjunction of literals

[13] Not all of the possible propositional assignments have to be *compatible*, i.e. derivable from a variable assignment. For instance, given the formula

$$x = y \wedge y = x$$

even though there are two distinct component atoms, for any assignment to the variables of this formula, in any possible structure of the subjacent language, there are only two compatible derived propositional assignments, $\langle \top, \top \rangle$ and $\langle \bot, \bot \rangle$: the other two possible pairs are excluded, since no assignment in any structure could reduce the above formula to either of them.

$$\widehat{\beta}_{j,1} \wedge \cdots \wedge \widehat{\beta}_{j,n}$$

such that $\widehat{\beta}_{j,i} \stackrel{s}{=} \beta_i$, if $\mathcal{V}_j(\beta_i) = \top$, and $\widehat{\beta}_{j,i} \stackrel{s}{=} \neg\beta_i$, otherwise. If there is a model $\mathfrak{A}$ for T in which ψ_j is satisfiable, then $\mathcal{V}_j$ represents a compatible propositional assignment for $\beta_1, \ldots, \beta_n$. Given that

(a) ψ_j is satisfiable in T iff its negation, which has the form $\neg\widehat{\beta}_{1,j} \vee \cdots \vee \neg\widehat{\beta}_{n,j}$, is invalid in T, and

(b) according to the hypothesis, the class of disjunction of literals is decidable for T,

it is always possible to effectively determine whether $\mathcal{V}_j$ represents a compatible propositional assignment for the atomic components of ϕ. After the process has been repeated for every $\mathcal{V}_j$, $1 \leq j \leq 2^n$, let f be the function

$$f(\mathcal{V}_j) = \begin{cases} 1 & \text{if } T \not\models \neg\psi_j \quad \text{(i.e. } \psi_j \text{ is } T\text{-satisfiable)} \\ 0 & \text{otherwise} \end{cases}$$

If $f(\mathcal{V}_j) = 0$, since ψ_j is T-unsatisfiable, there is no variable assignment α in any model $\mathfrak{A}$ for T such that, for all $i, 1 \leq i \leq n$,

$$\mathfrak{A} \models \beta_i^{\mathfrak{A}}[\alpha] \equiv \mathcal{V}_j(\beta_i)$$

Let $\Phi = \{\sigma_j\phi \mid \sigma_j = \{^{\mathcal{V}_j(\beta_1)}/_{\beta_1}, \ldots, ^{\mathcal{V}_j(\beta_n)}/_{\beta_n}\} \;\&\; f(\mathcal{V}_j) = 1\}$. ϕ is valid in T iff every possible derived propositional assignment for the atomic components of ϕ generates a logically valid proposition. This is the case iff the set of propositions Φ is propositionally valid. Since the validity of Φ can be effectively determined and ϕ is quantifier-free, the set of quantifier-free formulae of $\mathcal{L}$ is decidable for T in $Fml_{\mathcal{L}}$. ∎

As a result, for any theory that satisfies the conditions set by the theorem, the validity of a quantifier-free formula can be established by a decision procedure for the propositional calculus, such as the method of partial assignments or truth tables. If ϕ is quantifier-free, its truth value for a particular variable assignment α in a model for T can be propositionally determined from the truth value of its atomic constituents. Given a propositional assignment to the atoms of a quantifier-free formula ϕ, it suffices to consider whether such assignment is compatible in T, i.e. if there is a variable assignment α in a model for T in which the atoms assume the truth values set by the propositional assignment. Once all compatible propositional assignments have been identified, the validity of ϕ is reduced to a propositional problem[14].

C.4.3 Sublanguages

Given that the set of formulae of any first-order language can be generated by a context-free grammar, sublanguages represent natural candidates in the search for decidable

[14] This result is used in GETFOL to provide a decision procedure for the class of quantifier-free formulae of the predicate calculus with equality but without function symbols; see [Armando & Giunchiglia 93], p. 483-4.

subclasses. If T is a theory in a language $\mathcal{L}$ that is an expansion of $\mathcal{L}'$, $\mathcal{L}'$ is a *decidable sublanguage* for T iff T is decidable in $Fml_{\mathcal{L}'}$. Given that the intersection of a theory with a sublanguage is a theory, if T admits a decidable sublanguage, then it has a decidable subtheory[15].

Theorem C.4.3 *Let T be a consistent decidable theory in $\mathcal{L}$, $\mathcal{L}'$, an expansion of $\mathcal{L}$, and T', a conservative extension of T in $\mathcal{L}'$. Then $Fml_{\mathcal{L}}$ represents a decidable class of type II for T' in $Fml_{\mathcal{L}'}$. Moreover, if f_T is a decision procedure for T w.r.t. $Fml_{\mathcal{L}}$, the function h_{II},*

$$h_{II}(\ulcorner\phi\urcorner) = \begin{cases} 0 & \text{if } \phi \in (Fml_{\mathcal{L}'} - Fml_{\mathcal{L}}) \\ 1 + f_T(\ulcorner\phi\urcorner) & \text{otherwise} \end{cases}$$

is a decision procedure for T' w.r.t. $Fml_{\mathcal{L}}$.

PROOF. h_{II} is clearly recursive, since $(Fml_{\mathcal{L}'} - Fml_{\mathcal{L}})$ is an effectively computable set[16]. Since f_T is a decision procedure for T w.r.t. $Fml_{\mathcal{L}}$, h_{II} can be represented as

$$h_{II}(\ulcorner\phi\urcorner) = \begin{cases} 0 & \text{if } \phi \notin Fml_{\mathcal{L}} \\ 1 & \text{if } \phi \in Fml_{\mathcal{L}} - T \\ 2 & \text{if } \phi \in T \end{cases}$$

As T' is a conservative extension of T, then

$$\begin{aligned} T' \cap Fml_{\mathcal{L}} &= T \\ Fml_{\mathcal{L}} - T' &= Fml_{\mathcal{L}} - T \end{aligned}$$

Hence

$$h_{II}(\ulcorner\phi\urcorner) = \begin{cases} 0 & \text{if } \phi \notin Fml_{\mathcal{L}} \\ 1 & \text{if } \phi \in Fml_{\mathcal{L}} - T' \\ 2 & \text{if } \phi \in T' \cap Fml_{\mathcal{L}} \end{cases}$$

From definition C.2.1, it follows that T' is decidable in $Fml_{\mathcal{L}}$. Since T' is a conservative extension of a consistent theory, it must also be consistent. $Fml_{\mathcal{L}}$ therefore contains both theorems and non-theorems of T', and has type II. ∎

Once again Venn diagrams provide a suitable representation for such classes, as illustrated in figure C.3. Examples of decidable sublanguages are examined in chapter 9.

---

[15] Hence, the study of possible applications of decision procedures in undecidable theories starts with the search for decidable extensions, as described in section C.1, and can then be followed by the search for decidable subtheories.

[16] If g is a decision procedure for $(Fml_{\mathcal{L}'} - Fml_{\mathcal{L}})$ in $Fml_{\mathcal{L}'}$, h_{II} can be represented as

$$h_{II}(\ulcorner\phi\urcorner) = (1 - g(\ulcorner\phi\urcorner))(1 + f_{T'}(\ulcorner\phi\urcorner))$$

which is recursive.

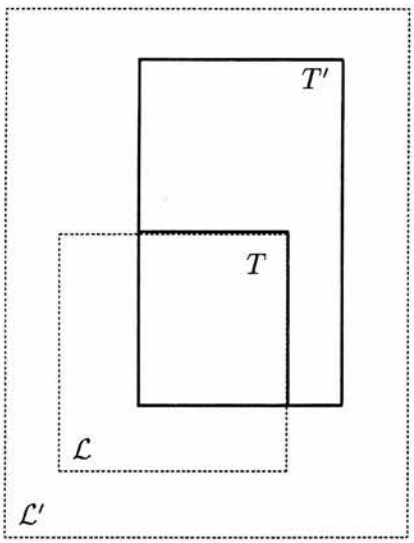


Figure C.3: Conservative Extensions of Decidable Theories

C.5 Other Decidable Subclasses

The previous section targeted a particular group of classes which have as common structural feature the fact that their construction is based on context-free grammars. Among the classes that do not share this property are those derived from decidable theories. Given an undecidable theory T in $\mathcal{L}$, any decidable theory T' in the same language is a potential source of decidable subclasses for T, provided that the logical links between both theories can be unveiled. There are two elementary cases in this respect. If T' is a subtheory of T, then T' is a decidable subclass of type I for T in $Fml_{\mathcal{L}}$, since $T' \subseteq T$. On the other hand, whenever $T \cap T' = \emptyset$, T' provides a subclass of type III for T.

Apart from decidable theories, decidable classes for other theories in $\mathcal{L}$ may also be relevant in the identification of decidable subclasses for T. This potential has already been explored in the case of $T_0^{\mathcal{L}}$, and can be extended to any other theory in the same language, whenever certain restrictions are met.

C.5.1 Intersection of Theories

When two theories have a non-empty intersection, but none of them is entirely contained in the other, model theoretic results can assist in assigning a role to each of them with respect to the identification of a decidable domain.

Theorem C.5.1 *Let T_1 and T_2 be theories in $\mathcal{L}$ which respectively admit $\mathfrak{A}_1$ and $\mathfrak{A}_2$ as models. If $\mathfrak{A}_2$ is a substructure of $\mathfrak{A}_1$, then*

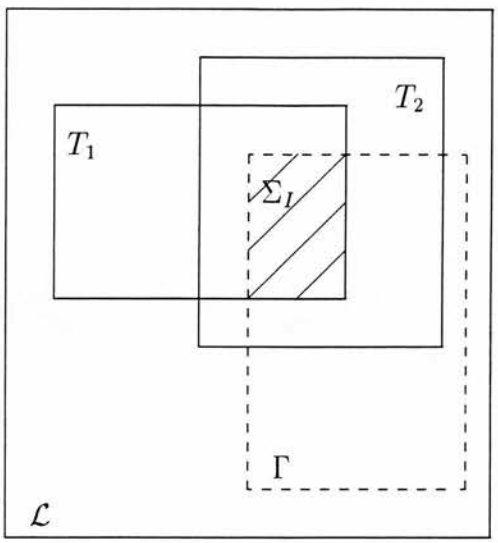
i. If T_1 is negation complete, every existential sentence that is a T_2-theorem is also a T_1-theorem.

ii. If T_2 is negation complete, every universal formula that is a T_1-theorem is also a T_2-theorem.

PROOF.

i. Let $\tau \stackrel{s}{=} (\exists v_1)\ldots(\exists v_n)\phi(v_1,\ldots,v_n)$ be an existential sentence. If τ is a theorem in T_2, it is valid in $\mathfrak{A}_2$. Hence there is an assignment α in $\mathfrak{A}_2$ such that $\mathfrak{A}_2 \models \phi[\alpha]$. Since $\mathfrak{A}_2$ is a substructure of $\mathfrak{A}_1$, α is also an assignment in A_1, so $\mathfrak{A}_1 \models \phi[\alpha]$, i.e. $\mathfrak{A}_1 \models (\exists v_1)\ldots(\exists v_n)\phi$. Since T_1 is negation complete, τ has to be amongst its theorems (otherwise $\neg\tau \in T_1$, and $\mathfrak{A}_1 \models \neg\tau$).

ii. Let $\psi \stackrel{s}{=} (v_1)\ldots(v_n)\phi(v_1,\ldots,v_n,\ldots,v_{n+m})$ be a universal formula. Assuming that it is a theorem of T_1, then, for every assignment α to the variables $v_1,\ldots,v_{n+m}$ in $\mathfrak{A}_1$, $\mathfrak{A}_1 \models \phi[\alpha]$. Since $\mathfrak{A}_2$ is a substructure of $\mathfrak{A}_1$, any assignment α to $v_1,\ldots,v_{n+m}$ in $\mathfrak{A}_2$ is also an assignment in $\mathfrak{A}_1$; therefore, for every assignment α in $\mathfrak{A}_2$, $\mathfrak{A}_2 \models \phi[\alpha]$, i.e. $\mathfrak{A}_2 \models (v_1)\ldots(v_n)\phi(v_1,\ldots,v_n,\ldots,v_{n+m})$. Since T_2 is negation complete, $\psi \in T_2$. ∎



T_1	decidable consistent theory
T_2	negation complete consistent theory
Γ	class of universal formulae
$\mathfrak{A}_1$	model for T_1
$\mathfrak{A}_2$	model for T_2
	(where $\mathfrak{A}_2$ is a substructure of $\mathfrak{A}_1$)
Σ_I	decidable subclass of type I for T_2

Figure C.4: Theories & Substructures

This result can be applied in particular to T_3 and *DAG*, the theory of densely ordered Abelian groups[17]. Figure C.4 illustrates an application of the above theorem in the delimitation of a decidable subclass.

C.5.2 Extension of Theories

Theories inherit some of the decidable subclasses for their subtheories. If T and T' are theories such that T' is an extension of T, then

[17] It suffices to consider that $\mathfrak{Z} = \langle \mathbb{Z}, 0, 1, +, < \rangle$, where $\mathbb{Z}$ is the set of integers, is a substructure of $\mathfrak{Q} = \langle \mathbb{Q}, 0, 1, +, < \rangle$, which in turn is a model for *DAG*. Although both theories are decidable, this result allows the use of decision procedures for *DAG*, which are less complex than those currently known for T_3, to recognise theorems of T_3 in a subclass of the subjacent language. See chapter 3.

* a decidable subclass Σ_I of type I for T is also a decidable subclass of type I for T', since $\Sigma_I \subset T \subset T'$.

* a decidable subclass of type III for T' is also a decidable subclass of type III for T, since $\Sigma_{III} \cap T' = \emptyset$ and $T \subset T'$.

Other types of subclasses, however, may not be preserved under extension, unless it is conservative.

Theorem C.5.2 *Let T be a theory in $\mathcal{L}$, and let T' be a conservative extension of T to $\mathcal{L}'$. Then every decidable subclass Σ of type j for T in $\mathcal{L}$ is also a decidable subclass of type j for T' in $\mathcal{L}'$. Moreover, if f_Σ is a decision procedure for T w.r.t. Σ, the function h_Σ,*

$$h_\Sigma(\ulcorner \phi \urcorner) = \begin{cases} 0 & \text{if } \phi \in (Fml_{\mathcal{L}'} - Fml_{\mathcal{L}}) \\ f_\Sigma & \text{otherwise} \end{cases}$$

is a decision procedure for T' w.r.t. Σ.

PROOF.

i. $j = I$

Trivial, since $\Sigma \subset T \subset T'$. Let f_Σ be a decision procedure for T w.r.t. Σ.

$$f_\Sigma(\ulcorner \phi \urcorner) = \begin{cases} 0 & \text{if } \phi \notin \Sigma \quad \text{(i.e. } \phi \in (Fml_{\mathcal{L}} - \Sigma)) \\ 2 & \text{if } \phi \in \Sigma \cap T = \Sigma \end{cases}$$

Then h_Σ above is equivalent to

$$h_\Sigma(\ulcorner \phi \urcorner) = \begin{cases} 0 & \text{if } \phi \in (Fml_{\mathcal{L}'} - \Sigma) \\ 2 & \text{if } \phi \in \Sigma \cap T' = \Sigma \end{cases}$$

considering that $(Fml_{\mathcal{L}'} - Fml_{\mathcal{L}}) \cup (Fml_{\mathcal{L}} - \Sigma) = (Fml_{\mathcal{L}'} - \Sigma)$. Hence h_Σ is a decision procedure for T' w.r.t. Σ in $Fml_{\mathcal{L}'}$.

ii. $j = III$

Trivial, since T' is a conservative extension of T: in this case, $\Sigma \cap T = \emptyset$ iff $\Sigma \cap T' = \emptyset$, hence Σ is a decidable subclass of type III for T' in $\mathcal{L}'$. Let f_Σ be a decision procedure for T w.r.t. Σ in $Fml_{\mathcal{L}}$.

$$f_\Sigma(\ulcorner \phi \urcorner) = \begin{cases} 0 & \text{if } \phi \notin \Sigma \quad \text{(i.e. } \phi \in (Fml_{\mathcal{L}} - \Sigma)) \\ 1 & \text{if } \phi \in \Sigma - T = \Sigma \end{cases}$$

Then h_Σ is equivalent to

$$h_\Sigma(\ulcorner \phi \urcorner) = \begin{cases} 0 & \text{if } \phi \in (Fml_{\mathcal{L}'} - \Sigma) \\ 1 & \text{if } \phi \in \Sigma - T' = \Sigma \end{cases}$$

Hence h_Σ is a decision procedure for T' w.r.t. Σ in $Fml_{\mathcal{L}'}$.

iii. $j = II$

Σ can be decomposed into two disjoint subsets, Σ_I and Σ_{III}, which respectively represent decidable subclasses of types I and III for T in $Fml_{\mathcal{L}}$. According to the first two cases of this proof, both Σ_I and Σ_{III} are decidable subclasses respectively of types I and III for T' in $Fml_{\mathcal{L}'}$. Hence $\Sigma = \Sigma_I \cup \Sigma_{III}$ is a decidable class of type II for T' in $Fml_{\mathcal{L}'}$.

Moreover, let h_{Σ_I} and $h_{\Sigma_{III}}$ be decision procedures for T' w.r.t. the disjoint subclasses.

$$h_{\Sigma_I}(\ulcorner\phi\urcorner) = \begin{cases} 0 & \text{if } \phi \in (Fml_{\mathcal{L}'} - \Sigma_I) \\ 2 & \text{if } \phi \in \Sigma_I \cap T' = \Sigma_I \end{cases}$$

$$h_{\Sigma_{III}}(\ulcorner\phi\urcorner) = \begin{cases} 0 & \text{if } \phi \in (Fml_{\mathcal{L}'} - \Sigma_{III}) \\ 1 & \text{if } \phi \in \Sigma_{III} - T' = \Sigma_{III} \end{cases}$$

Considering that $\Sigma_I \cup \Sigma_{III} = \Sigma$ and $\Sigma_I \cap \Sigma_{III} = \emptyset$, $h_\Sigma = (h_{\Sigma_I} \restriction \ulcorner\Sigma_I\urcorner) \cup (h_{\Sigma_{III}} \restriction \ulcorner\Sigma_I\urcorner)$ is a function such that

$$h_\Sigma(\ulcorner\phi\urcorner) = \begin{cases} 0 & \text{if } \phi \in (Fml_{\mathcal{L}'} - \Sigma) \\ 1 & \text{if } \phi \in \Sigma_{III} - T' = \Sigma - T' \\ 2 & \text{if } \phi \in \Sigma_I \cap T' = \Sigma \cap T' \end{cases}$$

Hence h_Σ is a decision procedure for T' w.r.t. Σ in $\mathcal{L}'$. ∎

Corollary C.5.1 *Let T be a theory in $\mathcal{L}$, and let T' be a definitional extension of T to $\mathcal{L}'$. Then every decidable subclass Σ of type j for T in $\mathcal{L}$ is also a decidable subclass of type j for T' in $\mathcal{L}'$.*

PROOF. According to lemma B.2.3 v, every definitional extension of a theory is also conservative. ∎

The main properties of the subclasses examined in this appendix are schematically described in table C.1.

C.6 Conclusions

Any essentially undecidable theory admits decidable subclasses of formulae. They may either be common to all the theories of a particular language, or may result from special properties of the theory. The identification of decidable subclasses relies on the study of other theories formulated in related languages, or on the inspection of effectively computable classes, particularly those generated by context-free grammars. The role of decision procedures, however, does not have to be restricted to such classes, since they can be expanded.

Subclass Description		Subclass Type	Theory Features
Logical	Prefix Classes	I	—
	Propositionally valid formulae	I	—
Non-logical	Axiom set	I	recursively axiomatisable
	Decidable subtheory	I	—
	Subset of universal formulae	I	negation complete extension as model for decidable theory
	Decidable sublanguage	II	conservative extension of subtheory
	Universal formulae	II	finitely axiomatisable function-symbol-free language
	Variable-free atoms	II	—

Table C.1: Decidable Subclasses for First-order Theories

Appendix D

Additional Concepts

Several of the basic concepts involved in this study come from the area of *equality reasoning*, introduced in section D.1. Rewrite rules, which are obtained from oriented equations or equivalences, and, under certain circumstances, also from implications, are defined in sections D.2 and D.3. The application of rewrite rules under a set of assumptions is described in section D.4, and section D.5 is dedicated to disagreement sets.

Control, a fundamental notion in computational processes in general, and in theorem proving in particular, specially in the presence of large proof and rewrite trees, is discussed in section D.6. *Sorts* correspond to subsets of the universe of a structure for a (first-order) language, which allow the definition of operations and relations with restricted domains. They are frequently required in the representation of statements about computer programs that deal with objects of distinct nature, such as natural numbers, arrays, lists, finite sets, etc. *Sequents* provide an alternative representation (w.r.t. the standard syntax described in appendix B) for formulae in general, and conditional statements in particular, which is convenient for conditional rewriting. The last two concepts are examined in sections D.7 and D.8.

D.1 Equality Reasoning

Resolution provides the basis for the construction of a complete inference system for the pure predicate calculus[1]. Binary resolution is the rule

$$\frac{p(t_1, \ldots, t_n) \vee C_1 \qquad \neg\, p(s_1, \ldots, s_n) \vee C_2}{\sigma C_1 \vee \sigma C_2}$$

where $p(t_1, \ldots, t_n) \vee C_1$ and $p(s_1, \ldots, s_n) \vee C_2$ are clauses that do not share variables, and σ is the mgu (most general unifier) for the complementary pair $\langle p(t_1, \ldots, t_n), \neg p(s_1, \ldots, s_n) \rangle$, i.e. $\sigma t_i \stackrel{s}{=} \sigma s_i$, for all $i, 1 \leq i \leq n$.

[1] More precisely, the resolution rule of inference suffices for the identification of unsatisfiable formulae in the class of clausal form sentences (where a clause is the universal closure of a disjunction of literals). See [Gallier 87], p. 404.

If equality is added to the underlying language, this rule alone does not guarantee deductive completeness to the system. It can be nonetheless strengthened by the replacement of syntactic identify with semantic equality in the second condition above: given a set of clauses Ω to which $p(t_1, \ldots, t_n) \vee C_1$ and $\neg p(s_1, \ldots, s_n) \vee C_2$ belong, if

$$\Omega \models (\sigma t_i = \sigma s_i)$$

for all $i, 1 \leq i \leq n$, where σ is a partial unifier for $p(t_1, \ldots, t_n)$ and $p(s_1, \ldots, s_n)$, then the above rule remains correct. Semantic equalities, however, are more complex to verify than their syntactic counterparts, due to their undecidability in the general case.

A possible way of restoring completeness to resolution in the presence of equality is the explicit introduction of equality axioms in every set of clauses. As this approach faces search problems, alternative solutions have to resort to more powerful inference rules which implicitly represent properties of equality. One of such rules,

$$\frac{\phi[\![t]\!] \qquad \psi \vee (t_1 = t_2)}{\phi[\![^{\sigma t_2}/_t]\!] \vee \sigma\psi} \quad (*)$$

involves a quantifier-free formula, $\phi[\![t]\!]$, a conditional equation, $\psi \vee (t_1 = t_2)$, that does not share variables with $\phi[\![t]\!]$, and a substitution σ such that $\sigma t_1 \stackrel{s}{=} t$ (i.e. t_1 is matchable against t). Since transformation is limited to a subexpression of ϕ, it follows that $\phi[\![t]\!] \equiv \phi[\![^{\sigma t_2}/_t]\!]$. A second rule,

$$\frac{\phi[\![t]\!] \qquad \psi \vee (t_1 = t_2)}{\sigma\phi[\![^{t_2}/_t]\!] \vee \sigma\psi} \quad (**)$$

where $\sigma t_1 \stackrel{s}{=} \sigma t$ (i.e. t_1 and t are unifiable), is such that variables may be instantiated outwith subexpression t as well. Thus, the resulting formula, $\sigma\phi[\![^{t_2}/_t]\!]$, is not necessarily equivalent to $\phi[\![t]\!]$. Inference rules of type $(*)$ are used for *demodulation* or term rewriting, as already examined in section D.2. Rules of type $(**)$ include *paramodulation*, usually represented as

$$\frac{\ell[\![t]\!] \vee C_1 \qquad u = u' \vee C_2}{\sigma\ell[\![^{u'}/_t]\!] \vee \sigma C_1 \vee \sigma C_2}$$

where ℓ is a literal, C_1 and C_2 are clauses and σ is the mgu of t and u. In conjunction with resolution, paramodulation provides the basis for a complete inference system for the class of clausal form sentences of first-order languages with equality. In this sense, paramodulation is deductively stronger than demodulation[2].

Theorem D.1.1 [Peterson 83] *Let Ω be a (finite) set of clauses of a language $\mathcal{L}$ (with equality). Ω is unsatisfiable iff the empty clause, $\square$, is derivable from*

$$\Omega \cup \{v = v\}$$

[2] Demodulation provides a complete inference system for quantifier-free equational theories, as discussed in [Gallier 87], p. 285-7.

Difference reduction procedures have been devised to control the application of paramodulation and other rules involving equality. Given a conjecture of the form

$$[\psi_1 \supset t_1 = u_1] \wedge \cdots \wedge [\psi_n \supset t_n = u_n] \supset [[t = u]]$$

whose negation can be expressed in clausal form as

$$(\neg\psi_1 \vee t_1 = u_1) \wedge \cdots \wedge (\neg\psi_n \vee t_n = u_n) \wedge (t \neq u)$$

where ψ_i is a conjunction of literals, any of such procedures gradually instantiates variables in t and u and applies conditional equations taken from the hypothesis set as rewrite rules, until two syntactically identical terms are obtained. As a result, at least some of the intermediate pairs of terms $\langle t', u' \rangle$ generated in the process must exhibit a lower amount of disagreement according to some measure, which justifies the name chosen for these mechanisms.

According to the dual form of the Herbrand theorem, a clausal sentence is unsatisfiable if and only if there exists a finite set of variable-free instances of its clauses whose conjunction is unsatisfiable. Since difference reduction procedures explore Herbrand theorem, they not only provide a refutation for the sentence above, in the event it is unsatisfiable, but also exhibit a substitution for its variables[3].

Control of the elimination of differences is provided both globally and locally. At the global level, a path in the search tree may be selected based on a description that includes the initial state, $\langle t = u, \{\ \} \rangle$, intermediate stages, and the final state, $\langle t' = u', \sigma \rangle$, where σ is the final substitution. At the local level, guidance has to be provided along three dimensions, for the selection of (i) partial unifiers, (ii) disagreement sets and (iii) elimination equalities. Partial unifiers reduce the differences between expressions by the removal of solvable disagreements. Since there may exist several partial unifiers for a single pair of non-unifiable expressions, it is necessary to choose amongst the available options. Choices cannot be limited to mgpus (most general partial unifiers), since they do not preserve completeness.

Once a unifier is chosen, there may be several possible disagreement sets between the resulting partially unified expressions. Each set represents a list of decomposed subproblems whose solution amounts to a solution for the original problem as well. Finally, conditional equations for the elimination of disagreements also have to be chosen. Each choice may introduce new difference reduction subproblems (in the event neither of the sides of the equality can be unified with the chosen subterm), hence the whole process has to be recursively applied to them.

D.2 Rewrite Rules

A first-order language is *equational* iff its set of predicate symbols is empty. A theory is *equational* whenever it is formulated in an equational language. As a result, atomic

[3] Since the Herbrand theorem applies to the pure predicate calculus, the presence of equality in the subjacent language requires the introduction of all equality axioms in the assumption set of a conjecture.

formulae in equational theories are all equations. A particular class of problems in an equational context has the form

$$\{t_1 = u_1, \ldots, t_n = u_n\} \models t_{n+1} = u_{n+1}$$

where $t_i, u_i, 1 \leq i \leq n+1$, are terms of an equational language. The equation $t_{n+1} = u_{n+1}$ is a logical consequence of the set of equational axioms $\{t_1 = u_1, \ldots, t_n = u_n\}$ iff it can be obtained from the axioms by the use of two inference rules, (*equation*) *instantiation* and *replacement of subterms by equal terms*[4]. Alternative formalisms include those based on the selection and application of *rewrite rules*, which are oriented equations used for subexpression replacement. The equality $t_{n+1} = u_{n+1}$ is established whenever the application of rewrite rules derived from equational axioms, or other equations that are logical consequences of axioms, to each of t_{n+1} and u_{n+1} leads to a pair of syntactically identical terms. A collection of such rules forms a *term rewriting set.*

This formalism can be generalised to contexts other than equational theories and languages. Rules may be also derived from equivalences, thus opening the possibility of both formula and term transformation. Additional applications of rewrite rules include simplification (where redundant semantic information is eliminated, and expressions are replaced by proper subexpressions), symbolic evaluation, and the reduction of expressions to canonical form[5].

Definition D.2.1 (Rewrite rules)

Let T be a theory in $\mathcal{L}$.

i. *A* (T-valid) rewrite rule *is an expression of the form*

$$\delta_1 \Rightarrow \delta_2$$

where $\delta_1 = \delta_2$ is T-valid (if δ_1 and δ_2 are terms) or $\delta_1 \equiv \delta_2$ is T-valid (if δ_1 and δ_2 are formulae). A rewrite set *is any set of rewrite rules.*

ii. *Given a rewrite rule R. $\delta_1 \Rightarrow \delta_2$ and an expression ϵ, ϵ' is a* rewritten expression *generated from ϵ by the application of R, i.e.*

$$\epsilon \overset{R}{\Rightarrow} \epsilon'$$

iff ϵ has a subexpression $\widehat{\epsilon}$ and there is a substitution σ such that (a) $\sigma\delta_1 \overset{s}{=} \widehat{\epsilon}$ *and* (b) $\epsilon' \overset{s}{=} \epsilon[\![^{\sigma\delta_2}/_{\widehat{\epsilon}}]\!]$.

iii. *A* rewriting sequence *for an expression ϵ and a rewrite set $\mathcal{R}$ is a sequence*

[4] See [Bachmair 91], p. 2.

[5] See [Bundy 83], p. 150-3.

$$\langle \epsilon_0, \epsilon_1, \ldots, \epsilon_n, \ldots \rangle$$

such that $\epsilon \stackrel{s}{=} \epsilon_0$ *and* $\epsilon_i \stackrel{R_i}{\Rightarrow} \epsilon_{i+1}, i \in \mathbb{N}$, *where* $R_i \in \mathcal{R}$. *An expression* ϵ' *represents a* rewritten version *of* ϵ *under* $\mathcal{R}$,

$$\epsilon \stackrel{\mathcal{R}}{\Rightarrow} \epsilon'$$

iff there is a finite rewriting sequence for ϵ *and* $\mathcal{R}$ *whose last element is* ϵ'. *In particular,* $\epsilon \stackrel{\mathcal{R}}{\Rightarrow} \epsilon$ (*empty rewriting sequence*).

iv. Given a finite rewriting sequence

$$\epsilon \stackrel{R_1}{\Rightarrow} \epsilon_1 \stackrel{R_2}{\Rightarrow} \ldots \stackrel{R_n}{\Rightarrow} \epsilon_n$$

ϵ_n *is a* normal form *for* ϵ *iff no rule of* $\mathcal{R}$ *is applicable to* ϵ_n.

v. A (T-valid) conditional rewrite rule *is an expression of the form*

$$\phi \;\rightarrow\; \delta_1 \Rightarrow \delta_2$$

where either $\phi \supset \delta_1 = \delta_2$ *(if* δ_1 *and* δ_2 *are terms) or* $\phi \supset (\delta_1 \equiv \delta_2)$ *(if* δ_1 *and* δ_2 *are formulae) is* T*-valid.*

vi. Given a formula $\psi_1 \supset \psi_2$, $\psi_1 \supset \psi_2'$ *is its* rewritten version *generated by the application of a conditional rewrite rule* $\phi \rightarrow \delta_1 \Rightarrow \delta_2$, *provided that* ψ_2 *has a subexpression* ϵ *and there is a substitution* σ *such that*

(a) $\sigma\delta_1 \stackrel{s}{=} \epsilon$,
(b) ψ_2' *is identical to* $\psi_2[\![^{\sigma\delta_2}/_{\epsilon}]\!]$, *and*
(c) $T \models \psi_1 \supset \sigma\phi$.

Example D.2.1

i. Given the formula

$$x + 0 \leq 1 \times y^0$$

it can be simplified *to* $x \leq y^0$ *by the application of rules*

$$\begin{array}{rcl} v + 0 & \Rightarrow & v \\ 1 \times v & \Rightarrow & v \end{array}$$

whereas the subterm y^0 *can be* evaluated *to* 1 *by the application of*

$$v^0 \;\Rightarrow\; 1$$

ii. The equation

$$(x^2 + x + 3) \times x = 0$$

is transformed into the canonical form equation $x^3 + x^2 + 3x = 0$ *(where* x^3 *is an abbreviation for* $(x \times x) \times x$*) by two applications of*

$$(v_1 + v_2) \times v_3 \quad \Rightarrow \quad (v_1 \times v_3) + (v_2 \times v_3)$$

iii. The arithmetical formula

$$y \times w \neq 0 \supset x^y \times w = 1 \times w$$

can be first simplified to $y \times w \neq 0 \supset x^y = 1$ *by a conditional rule,*

$$v_1 \neq 0 \quad \rightarrow \quad v_2 \times v_1 = v_3 \times v_1 \quad \Rightarrow \quad v_2 = v_3$$

or rather its instance,

$$w \neq 0 \quad \rightarrow \quad x^y \times w = 1 \times w \quad \Rightarrow \quad x^y = 1$$

given that $(y \times w \neq 0) \supset (w \neq 0)$ *is arithmetically valid. Then it can be put into* canonical form *(which in this case is represented as a conditional set of equations in polynomial form),* $y \times w \neq 0 \supset (x = 1 \vee y = 0)$*, after the application of*

$$v_1^{v_2} = 1 \quad \Rightarrow \quad v_1 = 1 \vee v_2 = 0$$

□

The role of rewrite rules in the extension of classes of formulae is discussed in section 2.3.

D.3 Implicational Rewrite Systems

The definition of rewrite systems may be extended once rules are distinguished between those derived from equivalences and those derived from implications.

Definition D.3.1 (Implicational Rewrite Systems - IRS)

Let T *be a theory in* $\mathcal{L}$ *and let* $R.\ \psi_1 \Rightarrow \psi_2$ *be a rewrite rule.*

i. R *is an* equivalence rule *(in* T*) iff*

$$T \quad \models \quad \psi_1 \equiv \psi_2$$

ii. *R is an* implication rule *(in T) iff*

$$T \models (\psi_1 \supset \psi_2) \quad \textit{and} \quad T \not\models (\psi_2 \supset \psi_1)$$

iii. *A rewrite system $\mathcal{R}$ for T is* implicational *if and only if it contains implication rewrite rules (in T). $\mathcal{R}$ is* strictly implicational *iff all its rules are derived from (T-valid) strict implications.*

No strictly implicational system therefore can be exhibited for an inconsistent theory. Neither has any of such systems a rule of the form

$$\psi \Rightarrow \bot,$$

since, for every $\psi \in Fml_{\mathcal{L}}$, if $T \models \psi \supset \bot$, then $T \models \psi \equiv \bot$.

Lemma D.3.1 *Every consistent theory T in $\mathcal{L}$ admits a strictly implicational rewrite system.*

PROOF. The set of rules

$$\mathcal{R} \;=\; \{\psi_1 \Rightarrow \top, \ldots, \psi_n \Rightarrow \top\}$$

where $\psi_i \in Fml_{\mathcal{L}}$ and $T \not\models \psi_i$, $1 \leq i \leq n$ (e.g. $\phi \wedge \neg\phi$, since T is consistent), constitutes a rewrite system for T, considering that, for any formula ϕ, $\models \phi \supset \top$, and that any logical theorem of $\mathcal{L}$ is also a T-theorem. As none of the rules is an equivalence rule, $\mathcal{R}$ is a strictly implicational rewrite system for T. ∎

The use of implication rules has to be restricted, in principle, to the *positive* occurrences of subformulae of a conjecture.

Definition D.3.2 (Subformula Polarity)

Let ϕ and ψ be formulae of a language $\mathcal{L}$.

i. *ψ has a* positive occurrence *in ϕ iff*

(a) *ϕ has any of the forms*

$$\psi,\ \psi \vee \psi',\ \psi' \vee \psi,\ \psi \wedge \psi',\ \psi \wedge \psi',\ \psi' \supset \psi,\ (v)\psi,\ (\exists v)\psi$$

for some $\psi' \in Fml_{\mathcal{L}}$, or

(b) *ψ has a positive (negative) occurrence in ϕ', which in turn has a positive (negative) occurrence in ϕ, for some proper subformula ϕ' of ϕ.*

ii. ψ *has a* negative occurrence *in* ϕ *iff*

(a) ϕ *has any of the forms*

$$\neg\psi,\ \psi \supset \psi',\ \psi \equiv \psi',\ \psi' \equiv \psi$$

for some $\psi' \in Fml_{\mathcal{L}}$, *or*

(b) ψ *has a positive (negative) occurrence in* ϕ' *and* ϕ' *has a negative (positive) occurrence in* ϕ, *for some proper subformula* ϕ' *of* ϕ.

Lemma D.3.2 *Let* $\phi[\![\psi]\!]$ *be a formula of* $\mathcal{L}$.

i. If ψ' *is obtained from* ψ *by the application of a* T*-valid equivalence rule, then*

$$T \models \phi[\![\psi]\!] \equiv \phi[\![^{\psi'}/_{\psi}]\!]$$

ii. If all the occurrences of ψ *in* ϕ *are positive, and* ψ' *is obtained from* ψ *by the application of a* T*-valid implication rule, then*

$$T \models \phi[\![\psi]\!] \supset \phi[\![^{\psi'}/_{\psi}]\!]$$

PROOF. The proof for a related result can be found in [Loveland 78], p. 40-1. ∎

As a consequence of lemma D.3.2, *IRSs* can be viewed as *refutational* procedures, representing therefore mechanisms for the recognition of unsatisfiable sentences. Concerning subexpression replacement, whenever a subformula ψ is rewritten to $\bot$, since it necessarily follows that $T \models \psi \equiv \bot$, there is no restriction about the polarity of its occurrences. Implication rules may then take part of a rewriting sequence that starts with (a negative occurrence of) ψ, as in the example below.

Example D.3.1 *Let* ϕ *be the formula*

$$\underbrace{(k < \min(l_1) \wedge \max(l_1) < k)}_{\psi} \supset (\min(l_1) + p < k)$$

and let

$$\begin{array}{lrcl} R_1 & v < \min(l) & \Rightarrow & v < \max(l) \\ R_2 & v_1 < v_2 \wedge v_2 < v_3 & \Rightarrow & v_1 < v_3 \\ R_3 & v < v & \Rightarrow & \bot \end{array}$$

be a set of implication and equivalence rewrite rules. The subexpression ψ *can then be rewritten to* $\bot$ *by the successive application of* R_1, R_2 *and* R_3. *As a result, in spite of the fact that* ψ *has a negative occurrence in* ϕ, *the conjecture can be rewritten to*

$$\bot \supset (\min(l_1) + p < k)$$

which is reducible to $\top$. □

D.4 Δ-Matching & Decidable Sublanguages

Certain undecidable theories, Peano arithmetic among them, admit decidable classes that delimit sublanguages of the underlying language $\mathcal{L}$. As a result, some of the non-logical symbols of $\mathcal{L}$ are deviant with respect to these subclasses. Any set $\mathcal{R}$ of *remove* rules for deviant symbols defines an extension of a decidable sublanguage $\mathcal{L}'$, by the application of elements of $\mathcal{R}$ to formulae of $\mathcal{L}$. The extended subclass, Σ', is recursively defined as

(i) If $\phi \in Fml_{\mathcal{L}'}$, then $\phi \in \Sigma'$.
(ii) If $\phi[\![^{\sigma\delta_2}/_{\epsilon}]\!] \in \Sigma'$ and

$$\epsilon \overset{s}{=} \sigma\delta_1$$

where $(R.\ \delta_1 \Rightarrow \delta_2) \in \mathcal{R}$ and σ is a substitution (i.e. R matches a subexpression of ϕ), then $\phi \in \Sigma'$.
(iii) Only the formulae that satisfy one of the above conditions belong to Σ'.

It consists, therefore, of the formulae of $\mathcal{L}'$ plus those of $(Fml_{\mathcal{L}} - Fml_{\mathcal{L}'})$ from which deviant symbols are completely removable by the application of $\mathcal{R}$. Whenever $\mathcal{R}$ is noetherian, Σ' is recursive[6].

Σ' is further enlarged when the more general notion of *semantic* or *T-matching*, which operates with equivalence or equality (in a theory T) instead of syntactic identity, is adopted. A rule is semantically applicable to ϕ if and only if the lhs expression of the rule has an instance that is equivalent (or equal) to a subexpression of ϕ.

Definition D.4.1 (Δ-matching)

Let Δ be a set of formulae of a language $\mathcal{L}$ (with equality), and let ϵ and δ be expressions of $\mathcal{L}$ that do not share variables.

i. δ strictly Δ-matches ϵ *iff there is a substitution σ such that* $\Delta \models (\epsilon = \sigma\delta)$.

ii. δ Δ-matches ϵ *iff there is a substitution σ and a subexpression ϵ' of ϵ, such that* $\Delta \models (\epsilon' = \sigma\delta)$.

iii. A rewrite rule $\delta_1 \Rightarrow \delta_2$ (strictly) Δ-matches ϵ *iff* δ_1 *(strictly) Δ-matches* ϵ.

When Δ represents a theory in $\mathcal{L}$, *theory-matching* is obtained as a special case of Δ-matching.

Example D.4.1 *Let δ be the expression $(x + 0)$, and let ϵ_1, ϵ_2 and ϵ_3 respectively be the expressions*

$$(y \times z) + 0$$
$$(y^2 \times 0) + y \times z^4$$
$$y^2 + 2y < 1$$

[6] See section 2.2.1.

i. δ *matches* ϵ_1 *with substitution* $\sigma_1 = \{^{y \times z}/_x\}$.

ii. δ *strictly PA-matches* ϵ_2 *with substitution e.g.* $\sigma_2 = \{^{z^4 \times y}/_x\}$, *considering that*

$$(z^4 \times y) + 0 \quad \equiv \quad 0 + (y \times z^4) \quad \equiv \quad (y^2 \times 0) + y \times z^4$$

in PA.

iii. δ *is PA-matchable against numerous subexpressions of* ϵ_3, *e.g.*

(a) y^2, *with substitution* $\sigma_3 = \{^{y^2}/_x\}$,

(b) $y^2 + 2y$, *with substitution* $\sigma_3' = \{^{2y + y^2}/_x\}$ *and*

(c) 1, *with substitution* $\sigma_3'' = \{^{1}/_x\}$. □

Definition D.4.1 is relevant for the construction of semantic extensions of subclasses. Given an undecidable theory T in $\mathcal{L}$, a decidable sublanguage $\mathcal{L}'$ for T and a set $\mathcal{R}$ of *remove* rules for deviant symbols, the *semantically extended class* Π is defined as follows.

(i) If $\phi \in Fml_{\mathcal{L}'}$, then $\phi \in \Pi$.
(ii) If $\phi[\![^{\sigma\delta_2}/_\epsilon]\!] \in \Pi$ and

$$T \quad \models \quad \epsilon = \sigma\delta_1 \quad (*)$$

where $(R.\ \delta_1 \Rightarrow \delta_2) \in \mathcal{R}$, σ is a substitution (i.e. R T-matches a subexpression of ϕ) and $=$ denotes either the equality or the biconditional symbol, then $\phi \in \Pi$.
(iii) Only the formulae that satisfy one of the above conditions belong to Π.

Since $(*)$ is generally undecidable, Π is not always recursive[7].

The task of proving equivalences or equalities in T can be in principle replaced with the less ambitious task of proving, from a finite set of T-valid hypotheses, Δ, that a subexpression of a conjecture is equivalent or equal to an instance of the lhs expression of a *remove* rule. However, since Δ-matching a rule against an expression can be undecidable even if Δ is finite, the new subclass, Π', is not recursive in general either. A feasible alternative is the construction of recursive subclasses of Π' which include Σ' as proper subsets, with the help of mechanisms that define weaker versions of Δ-matching and ensure the recursiveness of the extended class, Σ. With respect to the continuous enlargement of Σ, there are two possible strategies,

(a) the introduction of new (independent) *remove* rules in $\mathcal{R}$, which enlarges the core

---

[7] Semantic unification is discussed, for instance, in [Dershowitz & Jouannaud 90], p. 282-4. Concerning $(*)$, when both ϵ and δ_1 are formulae and $(*)$ is decidable, two derived problems,

$$\begin{aligned} \{\epsilon\} \cup T \quad &\models \quad \sigma\delta_1 \\ \{\sigma\delta_1\} \cup T \quad &\models \quad \epsilon \end{aligned}$$

must be decidable as well. In particular, when $\sigma\delta_1$ is a theorem of T, then

$$T \models \epsilon$$

As ϵ can be virtually any formula of $Fml_{\mathcal{L}} - Fml_{\mathcal{L}'}$, T would have to be decidable, in contradiction with the original hypothesis.

class Σ', and

(b) the inclusion of new hypotheses in Δ, which does not have any effect on Σ' but only on $\Sigma - \Sigma'$ (since Σ' results from the initial decidable class by the syntactic application of *remove* rules, a process in which Δ has no role).

The relationship between all the classes described above is represented in figure D.1.

Δ-matching subsumes a restricted version of disagreement elimination. Let ϕ be a conjecture that contains a deviant symbol S, R be a *remove* rule for S which is not applicable to ϕ, and ϵ be a subexpression of ϕ that contains S and has the same syntactic type as the lhs expression of R. If the application of additional rules for disagreement elimination is confined to ϵ, a T-equivalent expression, ϵ', is then generated, given that rewrite rules in the current context are all derived from equivalences valid in a theory T as well. In the event that R can be (syntactically) matched against ϵ', then, according to definition D.4.1, R is Δ-matchable against ϕ, where Δ is the set of additional rules.

When confined to the scope of a subexpression of the conjecture, disagreement elimination is deductively weaker than Δ-matching, for disagreement elimination is limited to the application of oriented equations (i.e. rewrite rules), whereas Δ-matching requires a complete inference apparatus. On the other hand, unrestricted disagreement elimination is not reducible to a special case of Δ-matchability, given that additional rules can be applied to subexpressions other the one against which a *remove* rule is eventually matched. Such cases require a stronger notion of matching, which is examined in section 8.2.3.

Since the enlargement of decidable classes has to preserve their recursiveness, the extension of a decidable sublanguage has to target classes such as Σ, as described in figure D.1. A possible strategy for their construction can be based on procedures that generate extensions such as classes Π or Π' in the first place, upon which restrictions may then be imposed until recursive extensions are obtained. Such procedures include RUE-resolution, ECOP and other difference reduction mechanisms.

D.5 Disagreement Sets

Theorem provers for theories that contain equality tend to be rather complex in their formulation, involving the terminology inherited from resolution and a new set of proper concepts, such as *partial unifiers* and *disagreement sets.*

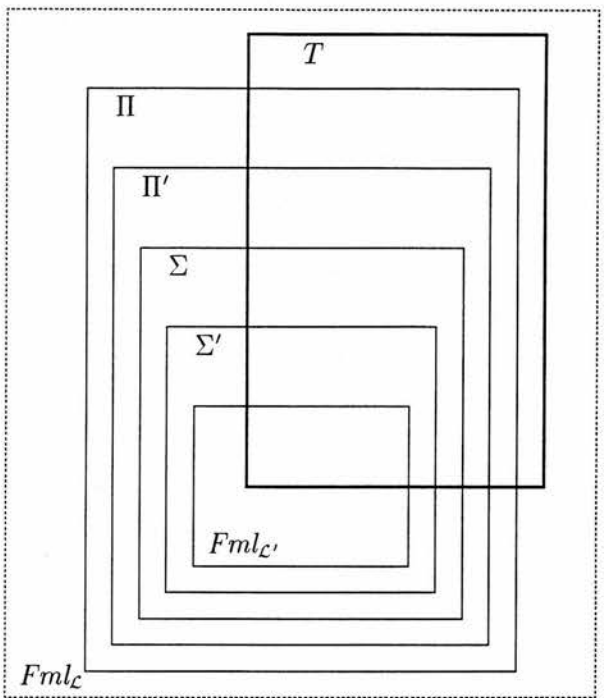
Definition D.5.1 (Partial unifiers)

Let ϵ_1 and ϵ_2 be syntactically distinct expressions that do not share variables.

i. ϵ_1 and ϵ_2 disagree at position p iff

(a) ϵ_i is an individual variable, individual constant or propositional constant, for some $i \in \{1, 2\}$, and $p = [\,]$, or

(b) ϵ_1 and ϵ_2 have respectively the forms $S_1(\delta_{1,1}, \ldots, \delta_{1,n_1})$ and $S_2(\delta_{2,1}, \ldots, \delta_{2,n_2})$, $S_1 \stackrel{S}{\neq} S_2$ and $p = [\,]$, or



$\mathcal{L}'$ Initial decidable sublanguage

Σ' Recursive extension of $Fml_{\mathcal{L}'}$
Syntactic application of *remove* rules
(syntactic matching)

Σ Recursive extension of Σ'
Controlled semantic application (with respect to Δ) of *remove* rules
(controlled Δ-matching)

Π' Non-recursive extension of Σ
Semantic application (with respect to Δ) of *remove* rules
(Δ-matching)

Π Non-recursive extension of Π'
Semantic application of *remove* rules
(T-matching)

Figure D.1: Extension of Decidable Sublanguages

(c) *ϵ_1 and ϵ_2 have respectively the forms $S(\delta_1, \ldots, \delta_n)$ and $S(\delta'_1, \ldots, \delta'_n)$, δ_i and δ'_i disagree at position p', and $p = [i|p']$.*

Whenever ϵ_1 and ϵ_2 disagree at position p, $\langle \epsilon_1, \epsilon_2 \rangle$ is a disagreement pair *and $\langle \epsilon'_1, \epsilon'_2 \rangle^p$ represents a* disagreeing pair for ϵ_1 and ϵ_2, *where ϵ'_1 and ϵ'_2 occurs respectively in ϵ_1 and ϵ_2 at position p.*

ii. A disagreeing pair $\langle \epsilon'_1, \epsilon'_2 \rangle^p$ between ϵ_1 and ϵ_2 is solvable *iff at least one of the elements of the pair is a variable, and is* fatal *otherwise.*

iii. A partial unifier *(pu) for ϵ_1 and ϵ_2 is a substitution σ that eliminates all solvable disagreements between ϵ_1 and ϵ_2.*

iv. A most general partial unifier *(mgpu) for ϵ_1 and ϵ_2 is a substitution σ such that*

(a) *σ is a pu for ϵ_1 and ϵ_2, and*

(b) *given a pu σ' for ϵ_1 and ϵ_2, if there is a substitution σ'' such that $\sigma = \sigma''\sigma'$, then σ'' is a variable-pure substitution.*

Definition D.5.2 (Disagreement Sets)

i. A disagreement set *$\mathcal{D}$ for a pair of terms $\langle t, s \rangle$ is any set of pairs of subterms that satisfies one of the following conditions.*

(a) *$\mathcal{D} = \{\ \}$, if $t \stackrel{s}{=} s$.*

(b) *$\mathcal{D} = \{\langle t, s \rangle\}$, if $t \stackrel{s}{\neq} s$* (origin disagreement set).

(c) *$\mathcal{D} = \{\langle t_1, s_1 \rangle\}, \ldots, \langle t_n, s_n \rangle\}$, if $t \stackrel{s}{=} f(t_1, \ldots, t_{n+p})$, $s \stackrel{s}{=} f(s_1, \ldots, s_{n+p})$, and $t_j \stackrel{s}{\neq} s_j$, for $1 \leq j \leq n$ only* (topmost disagreement set).

(d) *If $\mathcal{D}$ is a disagreement set for $\langle t, s \rangle$ such that $\langle t', s' \rangle \in \mathcal{D}$, and $\mathcal{D}'$ is a disagreement set for $\langle t', s' \rangle$, then $\mathcal{D}^*$, defined as*

$$(\mathcal{D} - \{\langle t', s' \rangle\}) \cup \mathcal{D}'$$

is also a disagreement set for $\langle t, s \rangle$.

ii. A disagreement set *for complementary literals $\langle p(t_1, \ldots, t_n), \neg p(s_1, \ldots, s_n) \rangle$ is defined as*

$$\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}_i$$

where $\mathcal{D}_i$ is a disagreement set for the pair $\langle t_i, s_i \rangle$.

A pair of expressions may admit more than one mgpu when a variable has multiple occurrences in at least one of the expressions[8].

[8] The left mgpu for a pair of expressions can be obtained by the same algorithm presented e.g. in [Gallier 87] for the computation of the most general unifier, with the proviso that it does not fail when

Example D.5.1

i. Given the pair

$$\langle g(x,x,y,y),\ g(a,h(z),h(w),b)\rangle$$

where a, b are individual constants, $\sigma_1 = \{^{a}/_{x}, {}^{h(w)}/_{y}\}$ and $\sigma_2 = \{^{h(z)}/_{x}, {}^{b}/_{y}\}$ are both mgpu for it, while $\sigma_3 = \{^{a}/_{x}, {}^{h(b)}/_{y}, {}^{b}/_{w}\}$ and $\sigma_4 = \{^{h(a)}/_{x}, {}^{b}/_{y}, {}^{a}/_{z}\}$ are pu but not mgpu, since $\sigma_3 = \{^{b}/_{w}\}\sigma_1$ and $\sigma_4 = \{^{a}/_{z}\}\sigma_2$.

ii. The disagreement sets for the pair $\langle f(g(a,x,h(b,y))), f(g(b,h(b,c),h(b,c)))\rangle$ are

$$\begin{aligned}
\mathcal{D}_1 &= \{\langle f(g(a,x,h(b,y))), f(g(b,h(b,c),h(b,c)))\rangle\}\\
\mathcal{D}_2 &= \{\langle g(a,x,h(b,y)), g(b,h(b,c),h(b,c))\rangle\}\\
\mathcal{D}_3 &= \{\langle a,b\rangle,\ \langle x,h(b,c)\rangle,\ \langle h(b,y),h(b,c)\rangle\}\\
\mathcal{D}_4 &= \{\langle a,b\rangle,\ \langle x,h(b,c)\rangle,\ \langle y,c\rangle\}
\end{aligned}$$

Each one of these disagreement sets has three disagreeing pairs, one fatal, $\langle a,b\rangle$, and two solvable, $\langle x,h(b,c)\rangle$ and $\langle y,c\rangle$. □

Each disagreement set represents a possible decomposition of an equality problem into simpler subproblems, as justified by the following lemma.

Lemma D.5.1 *If $\mathcal{D} = \{\langle t_1,u_1\rangle,\ldots,\langle t_1,u_1\rangle\}$ is a disagreement set for a pair of terms $\langle t,u\rangle$, then*

$$\models\ (t \neq u)\ \supset\ ((t_1 \neq u_1) \vee \cdots \vee (t_n \neq u_n))$$

PROOF. Immediate consequence of the substitution axioms for function symbols and the definition of disagreement set. ∎

D.6 Search Control

Given a set of instructions (or actions), an algorithm is defined only when an order for their application is determined. Such ordering represents the *control* of a computation.

---

a fatal disagreement is found. An algorithm for the generation of all mgpus for a pair of expressions, on the other hand, can be derived from any left mgpu algorithm, provided it is amended to include the following condition.

Given a pair of terms $\langle f(t_1,\ldots,t_n), f(s_1,\ldots,s_n)\rangle$, any mgpu for the pair $\langle f(t_{p(1)},\ldots,t_{p(n)}), f(s_{p(1)},\ldots,s_{p(n)})\rangle$, where $p\colon \{1,\ldots,n\} \to \{1,\ldots,n\}$ is a permutation function, is also a mgpu for $\langle f(t_1,\ldots,t_n), f(s_1,\ldots,s_n)\rangle$.

Either a unique order for every possible input data is chosen, or else a context-sensitive ordering that takes into account features of both input data and intermediate results has to be determined.

In mechanical theorem proving, instructions or actions include the application of inference or rewrite rules, and the inclusion of additional hypotheses in a conjecture ϕ. The process of recognition of theorems can be modelled as a succession of *states* that starts with ϕ, where derived states are obtained from previous ones after a (legal) action takes place. The universe of all possible states associated with ϕ forms the *search space* for ϕ. Control, under these circumstances, determines an order for the generation of this space.

In the context of decidable subclass extension, a rewrite rule set or a list of potential additional hypotheses implicitly define an extension for a decidable class. Given a formula ϕ of the underlying language, the process of determining whether ϕ is a member of the extended class or not defines a search space for ϕ. It has then to be scrutinised until a transformed formula that belongs to the original decidable (or reduction) class is found.

Formula rewriting is usually associated with the *exhaustive* application of the rules of a set $\mathcal{R}$. Control mechanisms, even though necessary, are immaterial from the extensional point of view: provided that the rewriting tree is finite, and that every path leads to an expression in normal form, any strategy (e.g. depth first, breadth first or iterative deepening) generates the same results. In the presence of infinite or large search spaces, however, exhaustive uniform search may be replaced by context-sensitive mechanisms.

The formal analysis of subdomains may reveal properties which can lead to the creation of specific search strategies for them. Also, properties that are known to be valid for some elements of a domain can be generalised to others, even if no formal validation for such generalisation is actually available, as part of the *heuristic approach* to problem solving. Statistical criteria are then invoked to assess the merits of such generalisations. Both formal analysis and heuristic approaches affect the search space; in the heuristic case, the completeness of a representation may be lost, since guidelines can exclude solutions[9].

D.7 Many-sorted Theories

From the semantic viewpoint, many-sorted (first-order) languages differ from their standard counterparts due to the presence of a collection of non-empty universes in their structures. At the syntactic level, the alphabet of a many-sorted language $\mathcal{L}^*$ reflects the fact that individual objects may belong to distinct universes. The set of non-logical symbols of $\mathcal{L}^*$ includes a subset of *sorts* $\mathcal{S} = \{s_1, \ldots, s_n, \ldots\}$, and an infinite enumerable set of individual variables $\{v_1^{s_i}, \ldots, v_n^{s_i}, \ldots\}$ is assigned to each sort s_i. Individual constant symbols, if present, are also assigned a sort. The arity (or rank) of a function or a predicate symbol, instead of a natural number, is a finite ordered

[9] The use of heuristics in problem solving is considered, for instance, in [Luger & Stubblefield 89], p. 38.

list of sorts. For predicate symbols, each element in its rank identifies the sort of the corresponding argument. In the case of a function symbol f, the last element of the rank identifies the sort of the object denoted by a f-dominated term[10].

The definition of well-formed atomic formulae has to take into account the sort of the arguments of a predicate symbol. If p has rank $(s_1, \ldots, s_n)$, the expression $p(t_1, \ldots, t_n)$ is a well-formed atomic formula iff t_i is a term of sort s_i, $1 \leq i \leq n$. For equations, $t_1 = t_2$ is well-formed iff t_1 and t_2 have the same sort. Composite formulae are defined in the same way as in standard first-order languages. It is possible to translate formulae of a many-sorted language $\mathcal{L}^*$ into a standard language $\mathcal{L}$ as follows.

i. Excluding the symbols for sorts, all the non-logical symbols of $\mathcal{L}^*$ are also non-logical symbols of $\mathcal{L}$, with the proviso that, if f is a function symbol of rank $(s_1, \ldots, s_{n+1})$ in $\mathcal{L}^*$, it has rank n in $\mathcal{L}$, and if p is a predicate symbol of rank $(s_1, \ldots, s_n)$ in $\mathcal{L}^*$, p has rank n in $\mathcal{L}$.

ii. For each sort s_i in $\mathcal{L}^*$, a new unary predicate letter, p_{s_i} is introduced in $\mathcal{L}$.

iii. Only the above symbols are non-logical symbols of $\mathcal{L}$.

Given a formula ϕ in $\mathcal{L}^*$, its translated version ϕ^t in $\mathcal{L}$ is defined as follows:

(i) if ϕ is quantifier-free, then $\phi^t \stackrel{s}{=} \phi$,
(ii) if $\phi \stackrel{s}{=} (\forall v^s)\psi$, then $\phi^t \stackrel{s}{=} (\forall v)(p_s(v) \supset \psi^t)$, and
(iii) if $\phi \stackrel{s}{=} (\exists v^s)\psi$, then $\phi^t \stackrel{s}{=} (\exists v)(p_s(v) \wedge \psi^t)$.

Intuitively, the introduction of $p_s(v)$ for each quantifier (Qv) has the purpose of simulating the existence of multiple universes in the structures for $\mathcal{L}^*$: each universe in a many-sorted structure corresponds to the subset of the universe for a structure of $\mathcal{L}$ that satisfies the predicate p_s.

A structure $\mathfrak{A}$ for $\mathcal{L}$ can be built from a structure $\mathfrak{A}^*$ of $\mathcal{L}^*$, provided that

(i) the universe of $\mathfrak{A}$ is the union of the universes for $\mathfrak{A}^*$, and
(ii) the operations and relations in $\mathfrak{A}$ are extensions of the corresponding operations and relations in $\mathfrak{A}^*$.

$\mathfrak{A}$ then is a model for the set of $\mathcal{L}$-sentences

$$(\exists v)p_{s_i}(v), \quad 1 \leq i \leq (n+1)$$

$$(v_1)\ldots(v_{n+1})\left(\bigwedge_{i=1}^{n} p_{s_i}(v_i) \supset p_{s_{n+1}}(f(v_1, \ldots, v_n))\right)$$

where f has rank $(s_1, \ldots, s_{n+1})$ in $\mathcal{L}^*$. The first condition reflects the fact that each universe in $\mathfrak{A}^*$ is non-empty, and the second one is related to the fact that composite terms in $\mathcal{L}^*$ denote objects of specific sorts. A *many-sorted theory* T^* is a theory defined over a many-sorted language $\mathcal{L}^*$, i.e. $T^* \subseteq Fml_{\mathcal{L}^*}$ and $\phi \in T^*$ iff $T^* \models \phi$. The translation mechanism for many-sorted languages also allows the reduction of many-sorted into standard first-order theories.

[10] See [Monk 76], p. 483-5.

This translation mechanism is used in the Boyer and Moore theorem prover for the implicit representation of sorts, as discussed in section 3.1. The predicate symbols p_s, are the *recognisers* of the prover, thus reflecting their role in establishing a partition for the universe of a structure.

D.8 Sequents

Sequents provide an alternative representation for conditional formulae in (first-order) languages. Given two finite sets of formulae, $\Gamma_1 = \{\gamma_{1,1}, \ldots, \gamma_{1,n}\}$ and $\Gamma_2 = \{\gamma_{2,1}, \ldots, \gamma_{2,m}\}$, in a language $\mathcal{L}$, the *sequent*

$$\Gamma_1 \rightarrow \Gamma_2$$

is a metatheoretical abbreviation for

$$\neg\gamma_{1,1} \vee \ldots \vee \neg\gamma_{1,n} \vee \gamma_{2,1} \vee \ldots \vee \gamma_{2,m}$$

if $\Gamma_1 \cup \Gamma_2 \neq \emptyset$, or for $\perp$, otherwise. When both Γ_1 and Γ_2 are non-empty, the above representation can be converted to conditional form,

$$(\gamma_{1,1} \wedge \cdots \wedge \gamma_{1,n}) \supset (\gamma_{2,1} \vee \cdots \vee \gamma_{2,m})$$

Sequents are notationally convenient, amongst other cases, when it is not relevant to

(i) explicitly include connectives in either the antecedent or the consequent of conditional formulae,
(ii) indicate an order for conjuncts and disjuncts, or
(iii) indicate the number of occurrences of each formula $\gamma_{1,i}$ or $\gamma_{2,j}$ in either Γ_1 or Γ_2 (since they are both sets, the number of occurrences of an element is immaterial).

Whenever Γ_2 has a single element, it is replaced in the sequent by this element, hence

$$\Gamma \rightarrow \psi,$$

where ψ is a formula, is also a well-formed sequent.

Appendix E

Additional Theorems and Lemmas

The proofs for some of the auxiliary results required in previous chapters have been gathered in this appendix. Given their elementary nature, none of them should be regarded as original and, as a result, they are not to be taken as part of the contribution of this dissertation.

E.1 Chapter 2

Lemma E.1.1 *Let T be a recursively axiomatisable undecidable theory in $\mathcal{L}$.*

i. If Σ is a decidable subclass for T, then there is a subclass Σ' in $Fml_{\mathcal{L}}$ such that $\Sigma \subset \Sigma'$ and Σ' is also decidable for T.

ii. $Fml_{\mathcal{L}}$ is the least recursive upper bound (with respect to $\subset$) for the process of extension of any decidable subclass for T.

PROOF.

i. Since T is recursively axiomatisable, there is a recursive function f that effectively enumerates its elements. Since T is undecidable, there is a formula $\phi \in T$ such that $\phi \notin \Sigma$. Formulae of $T - \Sigma$ can be effectively exhibited through the enumeration of all the elements of T by f until $n \in \mathbb{N}$ is found such that $f(n) = \ulcorner \phi \urcorner$ and $\phi \notin \Sigma$. As Σ is a decidable class for T, there is an effective procedure for determining whether ϕ is its element or not; as a result, the identification of such formula consumes a finite amount of computation. If Σ' is defined as $\Sigma \cup \{\phi\}$, then it is also a decidable subclass for T.

ii. Since T is undecidable, $Fml_{\mathcal{L}}$ is a recursive upper bound for the process of extension of any decidable subclass Σ for T. Let Σ' be a proper recursive class of $Fml_{\mathcal{L}}$ that contains Σ as proper subset.

If $T \not\subset \Sigma'$, there is a formula $\phi \in T$ such that $\phi \in Fml_{\mathcal{L}} - \Sigma'$. Since T is axiomatisable and Σ' is recursive, the same procedure described in the above lemma can be used to effectively exhibit such formula. Hence $\Sigma \cup \{\phi\}$ is also a decidable subclass for T, and $\Sigma \cup \{\phi\} \not\subset \Sigma'$.

If $T \subset \Sigma'$, since Σ' is recursive, $Fml_{\mathcal{L}} - \Sigma'$ is a decidable subclass of type III for T (since the complement of a recursive class is also recursive). Consequently $\Sigma \cup (Fml_{\mathcal{L}} - \Sigma')$ is a decidable subclass for T that is not a subset of Σ'.

Since no recursive subclass of $Fml_{\mathcal{L}}$ is an upper bound for the extension of Σ, $Fml_{\mathcal{L}}$ is the least upper bound for this process. ∎

Lemma E.1.2 *Let T be a recursively axiomatisable and undecidable theory T in $\mathcal{L}$, and let Δ be a subclass of $Fml_{\mathcal{L}}$ that is decidable for T in $Fml_{\mathcal{L}}$. Then there exists a formula $\phi \in Fml_{\mathcal{L}} - \Delta$ such that, for every formula $\psi \in \Delta$,*

$$T \not\models \phi \equiv \psi$$

PROOF. Assume otherwise that, for every $\phi \in Fml_{\mathcal{L}} - \Delta$, there is a formula $\psi \in \Delta$ such that $T \models \phi \equiv \psi$. Considering that T is recursively axiomatisable, there is an effective procedure for the exhibition of ψ, for each $\phi \in Fml_{\mathcal{L}} - \Delta$, consisting of the recursive enumeration of all the elements of T, until a formula of the form $\phi \equiv \psi$, with $\psi \in \Delta$, is found. As any T-theorem, according to such enumeration, has only finitely many predecessors, and as the existence of such a theorem is guaranteed by hypothesis, the search is finite.

Let h be a recursive function that represents the procedure for the recognition of ψ, and let f be a decision procedure for T w.r.t. Δ in $Fml_{\mathcal{L}}$. Then $h \circ f$ is a decision procedure for T w.r.t. $Fml_{\mathcal{L}}$, which contradicts the undecidability of T. ∎

Lemma E.1.3 *Let*

(i) *T' be a theory in $\mathcal{L}$,*
(ii) *T be a decidable subtheory of T',*
(iii) *$\Phi = \{\phi_1, \ldots, \phi_n\}$ be a subset of T', and*
(iv) *Σ be the class of formulae defined as*

$$\Sigma = \{\phi \in Fml_{\mathcal{L}} \mid h(\ulcorner \phi \urcorner) = 1 \vee h(g_\Phi(\ulcorner \phi \urcorner)) = 1\}$$

where h is a decision procedure for T and g_Φ is defined as

$$g_\Phi(\ulcorner \phi \urcorner) = \ulcorner (\bigwedge_{i=1}^{n} \phi_i) \supset \phi \urcorner$$

Then Σ is an extended recursive class for T w.r.t. T'.

PROOF. From the definition of Σ, it follows that Σ is recursive. Let f then be the recursive function defined as

$$f(\ulcorner\phi\urcorner) = \begin{cases} \ulcorner\phi\urcorner & \text{if } h(\ulcorner\phi\urcorner) = 1 \\ g_\Phi(\ulcorner\phi\urcorner) & \text{if } h(\ulcorner\phi\urcorner) + h(g_\Phi(\ulcorner\phi\urcorner)) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Given that

i. $T \subseteq \Sigma$, and

ii. f is a reduction function for Σ w.r.t. T', since

   (a) $f(\Sigma) \subseteq T$, and

   (b) If ϕ is a formula of $\Sigma \cap T'$, then $f(\ulcorner\phi\urcorner) \in \ulcorner T\urcorner \subset \ulcorner T'\urcorner$. Also, if $f(\ulcorner\phi\urcorner) \in \ulcorner T'\urcorner$, then $\phi \in \Sigma \subset T'$. Hence $\phi \in T'$ iff $f(\ulcorner\phi\urcorner) \in \ulcorner T'\urcorner$.

Σ then is an extended class for T w.r.t. T'. ∎

Lemma E.1.4 *If ϕ is a formula of $\mathcal{L}$, the following statements are equivalent.*

i. ϕ is satisfiable.

ii. Every formula entailed by ϕ is satisfiable.

iii. Each of the formulae entailed by ϕ that does not contain ϕ as subexpression is satisfiable.

PROOF. $(i \rightarrow ii \rightarrow iii \rightarrow i)$

i. If ϕ is satisfiable, then ϕ cannot entail an unsatisfiable formula, otherwise $\models \phi \supset \bot$, i.e. $\models \neg\phi$, and therefore ϕ is unsatisfiable.

ii. Trivial.

iii. Case (a) ϕ does not entail any unsatisfiable formula. Then $\not\models \phi \supset \bot$, i.e. $\not\models \neg\phi$, hence ϕ is satisfiable.
Case (b) ϕ entails an unsatisfiable formula ψ. Then, according to the third clause above, ψ must contain ϕ as subexpression. Since $\psi[\![\phi]\!]$ is unsatisfiable and $\models \phi \supset \psi[\![\phi]\!]$, given that $\models \psi[\![\phi]\!] \equiv \bot$, then $\models \phi \supset \bot$, thus contradicting the hypothesis. Therefore, ϕ does not entail any unsatisfiable formula, and, from case (a), ϕ is satisfiable. ∎

E.2 Chapter 3

Lemma E.2.1 *Let T be a theory in a language $\mathcal{L}$, and let $\Phi \rightarrow \phi$ be a sequent of $\mathcal{L}$, where $\Phi = \{\phi_1, \ldots, \phi_n\}$. Then $T \models \phi$ iff*

$$T \models \Phi \rightarrow \phi \quad \& \quad T \models \neg\phi_1 \rightarrow \phi \quad \& \quad \ldots \quad \& \quad T \models \neg\phi_n \rightarrow \phi$$

PROOF.

($\rightarrow$)
Trivial, since, whenever $T \models \phi$, then $T \models \Phi \rightarrow \phi$, for any finite set of formulae Φ.

($\leftarrow$)
Given the standard syntactic representation for the above formulae,

$$\bigwedge_{i=1}^{n} \phi_i \supset \phi, \quad \neg\phi_1 \supset \phi, \quad \ldots, \quad \neg\phi_n \supset \phi$$

it follows that

$$T \models \left(\bigwedge_{i=1}^{n} \phi_i \supset \phi \right) \wedge (\neg\phi_1 \supset \phi) \wedge \cdots \wedge (\neg\phi_n \supset \phi)$$

which can be simplified to

$$\begin{array}{lll} T & \models & \left(\phi \vee \bigvee_{i=1}^{n} \neg\phi_i \right) \wedge (\phi_1 \vee \phi) \wedge \cdots \wedge (\phi_n \vee \phi) \\ T & \models & \phi \end{array}$$

∎

Lemma E.2.2 *Let T be a first-order theory. Then $T \models \Gamma \cup \{\neg\phi\} \rightarrow \phi$ if and only if $\models \Gamma \rightarrow \phi$.*

PROOF. Let $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$. The sequent $\Gamma \cup \{\neg\phi\} \rightarrow \phi$ corresponds to the formula $(\gamma_1 \wedge \cdots \wedge \gamma_n \wedge \neg\phi) \supset \phi$, from which the following list of logically equivalent formulae can be derived,

$$\begin{array}{c} \neg\gamma_1 \vee \cdots \vee \neg\gamma_n \vee \phi \vee \phi \\ \neg\gamma_1 \vee \cdots \vee \neg\gamma_n \vee \phi \\ (\gamma_1 \wedge \cdots \wedge \gamma_n) \supset \phi \end{array}$$

Since the last formula above corresponds to the sequent $\Gamma \rightarrow \phi$, it follows that, for any theory T, $T \models \Gamma \cup \{\neg\phi\} \rightarrow \phi$ iff $T \models \Gamma \rightarrow \phi$. ∎

E.3 Chapter 4

Conjecture E.3.1 *The class of terms $\mathcal{E}$, recursively defined as*

i. Any atomic term of $\mathcal{L}$ is an element of $\mathcal{E}$

ii. If $t \in \mathcal{E}$ and $S \in Sym_{\mathcal{L}}$, then $S(t,\ldots,t) \in \mathcal{E}$, provided that $S(t,\ldots,t)$ is a well-formed term of $\mathcal{L}$

iii. Only the terms defined above are elements of $\mathcal{E}$

cannot be generated by a context-free grammar.

EVIDENCE. Assume otherwise that there is a context-free grammar that generates the above mentioned class. Then there must be a production of the form

$$exp \;:=\; S(atm_1,\ldots,atm_n)$$

where variables $atm_1,\ldots,atm_n$ denote atomic terms of $\mathcal{L}$. Given that, for each expression of $\mathcal{E}$, if it is dominated by S, then each of its arguments is syntactically identical to the other arguments, it is necessary to guarantee that (i) a single variable *atm* has n occurrences in the scope of S in the above production (otherwise terms in the scope of S would belong to distinct subclasses and, as a result, would be syntactically distinct) and (ii) the set of expressions generated by the productions associated with *atm* is unitary (otherwise *atm* would denote more than a particular term, and the above production would generate S-dominated terms with syntactically distinct arguments). Hence, it would be necessary to have productions of the form

$$\begin{array}{rcl} atm^1 & := & t_1 \\ & \vdots & \\ atm^m & := & t_m \\ & \vdots & \end{array}$$

where $t_1,\ldots,t_m,\ldots$ are atomic terms of $\mathcal{L}$. Since $\mathcal{L}$ has infinite many atomic terms, there would be infinite many of such productions, thus contradicting definition C.2.2, which states that the number of rules of a grammar is finite. Therefore $\mathcal{E}$ cannot be defined by a context-free grammar.

E.4 Chapter 6

Lemma E.4.1 *Let $R.\ S(v_1,\ldots,v_p) \Rightarrow \delta$ be a total* remove *rule and let m denote the length of the longest symbolic chain of δ. Then, for any substitution σ, the maximum difference between the longest symbolic chain of $\sigma S(v_1,\ldots,v_p)$ and $\sigma\delta$ is $m-2$.*

PROOF. Let $\sigma = \{^{\epsilon_1}/_{v_1}, \ldots, ^{\epsilon_p}/_{v_p}\}$ be a substitution such that ϵ_i has the longest symbolic chain, for all $i, 1 \leq i \leq p$. Let n denote the length of the longest symbolic chain of ϵ_i. If v_i has an innermost occurrence[1] in δ, then the longest symbolic chain of $\sigma\delta$ has length $m + n - 1$ (since an occurrence of v_i is deleted before ϵ_i is introduced in its place). Otherwise the longest symbolic chain has length at least equal to m (since variable instantiation does not reduce chain length), and less than $m + n - 1$. Considering that the length of the longest chain of $S(\epsilon_1, \ldots, \epsilon_p)$ is $n + 1$, the difference between the lengths of the longest chains of $S(\epsilon_1, \ldots, \epsilon_p)$ and $\sigma\delta$ has a maximum value of $(m + n - 1) - (n + 1) = (m - 2)$. ∎

Lemma E.4.2 *Let ϕ be a formula that has n occurrences of a symbol S, and let R. $S(v_1, \ldots, v_p) \Rightarrow \delta$ be a total* remove *rule such that the longest symbolic chain of δ has length m. If ϕ' is obtained from ϕ after the exhaustive application of R, then the difference between the longest symbolic chains of ϕ' and ϕ is not greater than $n \times (m - 2)$.*

PROOF. (By induction on the number of occurrences of S)

Base case. If S has a single occurrence in ϕ, let $S(\epsilon_1, \ldots, \epsilon_p)$ be the S-dominated subexpression of ϕ. According to lemma E.4.1, the maximum difference between the longest symbolic chains of $S(\epsilon_1, \ldots, \epsilon_p)$ and $\delta(^{\epsilon_1}/_{v_1}, \ldots, ^{\epsilon_p}/_{v_p})$ is $m - 2$. Hence, if $S(\epsilon_1, \ldots, \epsilon_p)$ has an innermost occurrence in ϕ, the replacement of $S(\epsilon_1, \ldots, \epsilon_p)$ with $\delta(^{\epsilon_1}/_{v_1}, \ldots, ^{\epsilon_p}/_{v_p})$ causes an increase in the length of the longest chain of the rewritten formula ϕ' of $m - 2$ as well. Otherwise, the variation in the length of the longest chain of ϕ' w.r.t. ϕ lies between 0 and $m - 2$.

Step case. Assume that, for any formula that has n occurrences of S, the maximum expansion of the longest symbolic chain, after the exhaustive application of R, is $n \times (m - 2)$. Let ϕ then be a formula where S has $n + 1$ occurrences, and let ϵ be a S-dominated subexpression of ϕ where S occurs only once. If ϵ has the form $S(\epsilon_1, \ldots, \epsilon_p)$, then $\delta(^{\epsilon_1}/_{v_1}, \ldots, ^{\epsilon_p}/_{v_p})$ has no occurrences of S.

As a result, the replacement of ϵ with $\delta(^{\epsilon_1}/_{v_1}, \ldots, ^{\epsilon_p}/_{v_p})$ in ϕ reduces the number of occurrences of S in the rewritten formula ϕ' to n. Also, the length of the longest chain of ϕ' suffers a maximum expansion of $m - 2$ w.r.t. ϕ. Since ϕ' has n occurrences of S, according to the induction hypothesis, the exhaustive removal of S causes an maximum expansion of the rewritten formula of $n \times (m - 2)$ w.r.t. ϕ', or $(n + 1) \times (m - 2)$ w.r.t. ϕ. ∎

Lemma E.4.3 *If $\mathcal{R}$ is a normalised set of total* remove *rules, $\mathcal{R}$ is locally confluent.*

[1] A symbol S has an *innermost occurrence* in an expression ϵ iff S occurs in the lowest layer of ϵ. Clearly, the only symbols that can have innermost occurrences in a well-formed expression are variables, individual constants and propositional constants. An expression ϵ' has an *innermost occurrence* in ϵ iff the dominant symbol of ϵ' has an innermost occurrence in ϵ.

PROOF. Let $\mathcal{R}$ be a set of normalised total *remove* rules that includes the rules

$$\begin{array}{lll} R_i & S_i(v_1, \ldots, v_{n_i}) & \Rightarrow \quad \delta_i\{v_1, \ldots, v_{n_i}\} \\ R_j & S_j(u_1, \ldots, u_{n_j}) & \Rightarrow \quad \delta_j\{u_1, \ldots, u_{n_j}\} \end{array}$$

where v_k, u_l, $1 \leq k \leq n_i, 1 \leq l \leq n_j$, are variables. All critical pairs have the form

$$\left\langle \delta_i \left\{ v_1, \ldots, v_k, {}^{S_j(u_1, \ldots, u_{n_j})}/_{v_{k+1}, \ldots, v_{n_i}} \right\}, S_i \left(v_1, \ldots, v_k, {}^{\delta_j\{u_1, \ldots, u_{n_j}\}}/_{v_{k+1}, \ldots, v_{n_i}} \right) \right\rangle$$

The first element of the pair can be rewritten to

$$\delta_i\{v_1, \ldots, v_k, \delta_j\{u_1, \ldots, u_{n_j}\}, \ldots, v_{n_i}\}$$

by the application of R_j to its subexpression $S_j(u_1, \ldots, u_{n_j})$. The second element of the pair can be transformed into the same expression by an application of R_i. Since this result applies to all possible critical pairs for $\mathcal{R}$, $\mathcal{R}$ is locally confluent. ∎

E.5 Chapter 9

Lemma E.5.1 *Let*

i. ϵ_0 *be an expression whose longest* S*-chain has length* m*.*

ii. $\mathcal{R}$ *be a set of* remove *rules for* S*.*

iii. the rewriting sequence

$$\epsilon_0 \overset{R_1}{\Rightarrow} \epsilon_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_n}{\Rightarrow} \epsilon_n$$

be such that no S*-chain of length* m*, or fragment of such chain, occurs in the scope of the subexpression to which* R_i *is applied, where* $R_i \in \mathcal{R}$*,* $1 \leq i \leq n$*.*

iv. ϵ_i' *be the expression in which the longest* S*-chain has maximum length* $m - 1$ *generated from* ϵ_i*,* $1 \leq i \leq n$*, as follows: if* $S(u_1, \ldots, u_n)$ *is the terminal expression of a* S*-chain of length* m *in* ϵ_i*, it is replaced with an expression* ϵ'' *which is free from occurrences of* S*.*

Then

$$\epsilon_0' \overset{R_1}{\Rightarrow} \epsilon_1' \overset{R_2}{\Rightarrow} \ldots \overset{R_n}{\Rightarrow} \epsilon_n'$$

PROOF. (By strong induction on the length of the rewriting sequence)

Assume by hypothesis that, for any rewriting sequence of length n, $\epsilon_0 \overset{R_1}{\Rightarrow} \epsilon_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_n}{\Rightarrow} \epsilon_n$, it follows that

$$\epsilon'_0 \overset{R_1}{\Rightarrow} \epsilon'_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_n}{\Rightarrow} \epsilon'_n \quad (*)$$

Let

$$\epsilon \overset{R_1}{\Rightarrow} \epsilon_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_n}{\Rightarrow} \epsilon_n \overset{R_{n+1}}{\Rightarrow} \epsilon_{n+1}$$

be a rewriting sequence for ϵ_0 with length $n+1$, and let $\overline{\epsilon}_n$ be the subexpression of ϵ_n to which R_{n+1} is applied. According to the lemma's hypothesis, R_{n+1} is not applied to a subexpression of ϵ_n that contains either a S-chain of length m, or a fragment of such chain: $\overline{\epsilon}_n$ cannot therefore contain any terminal subexpression $S(u_1, \ldots, u_n)$ of a S-chain of length m. Also, $\overline{\epsilon}_n$ cannot be a proper subexpression of $S(u_1, \ldots, u_n)$, since R_{n+1} is a *remove* rule for S and each u_i is free from occurrences of S. Therefore, $\overline{\epsilon}_n$ must also occur in ϵ'_n, and, as a result, R_{n+1} is applicable to ϵ'_n as well.

Let ϵ denote the expression that results from ϵ'_n after the application of R_{n+1}. Considering that

i. ϵ'_n is obtained from ϵ_n by the replacement of terminal subexpressions of S-chains of length n with a S-free subexpression.

ii. ϵ'_{n+1} is obtained from ϵ_n by the application of R_{n+1}, which generates ϵ_{n+1}, followed by the operation of replacement of terminal subexpression of S-chains of length n.

iii. The application of R_{n+1} and the above operation of terminal subexpression replacement target distint non-overlapping subexpressions of ϵ'_n, being therefore interchangeable.

it follows that $\epsilon \overset{s}{=} \epsilon'_{n+1}$. Therefore

$$\epsilon'_n \overset{R_{n+1}}{\Rightarrow} \epsilon'_{n+1}$$

can be appended to sequence $(*)$, thus generating

$$\epsilon' \overset{R_1}{\Rightarrow} \epsilon'_1 \overset{R_2}{\Rightarrow} \ldots \overset{R_n}{\Rightarrow} \epsilon'_n \overset{R_{n+1}}{\Rightarrow} \epsilon'_{n+1}$$

∎

Lemma E.5.2 *Let T be an undecidable and recursively axiomatisable theory in $\mathcal{L}$. Let Π be the set of sentences of $\mathcal{L}$ which are undecidable in T, and Σ be a decidable subclass for T. Then*

i. $\ulcorner \Pi \urcorner$ *is not recursive*

ii. $\Sigma \cap \Pi \neq \Pi$

iii. If T *is negation complete w.r.t.* Σ *(i.e. if the sentences* τ *and* $\neg\tau$ *belong to* Σ*, then* $\tau \in T$ *or* $\neg\tau \in T$*), and* Σ' *is a recursive extension of* Σ *such that*

$$\Sigma' = \{\tau' \in Stn_{\mathcal{L}} \mid (\exists\tau)(\tau \in \Sigma \,\&\, T \models \tau' \equiv \tau)\}$$

then $\Sigma' \cap \Pi = \emptyset$

PROOF.

i. Assume otherwise that $\ulcorner \Pi \urcorner$ is recursive. Then there is a recursive function f_Π such that

$$f_\Pi(\tau) = \begin{cases} 0 & \text{if } \tau \notin \Pi \\ 1 & \text{otherwise} \end{cases}$$

Also, $\ulcorner Stn_{\mathcal{L}} - \Pi \urcorner$ must be recursive, and T must be negation complete w.r.t. this class (otherwise there would be an undecidable sentence in $Stn_{\mathcal{L}} - \Pi$). Since T is recursively axiomatisable, according to theorem B.3.1, there is a recursive function g_T that recursively enumerates all its elements. Given $\tau \in Stn_{\mathcal{L}}$, a decision procedure for T could then be built as follows.

(a) if $f_\Pi(\tau) = 1$ (i.e. $\tau \in \Pi$), then $\tau \notin T$.

(b) if $f_\Pi(\tau) = 0$, then $\tau \in Stn_{\mathcal{L}} - \Pi$. As a result, there is $n \in \mathbb{N}$ such that $g_T(n) = \ulcorner \tau \urcorner$ or $g_T(n) = \ulcorner \neg\tau \urcorner$. In the first case, it follows that $\tau \in T$, and in the second one, that $\tau \notin T$.

Hence T would be decidable, in contradiction with the hypothesis.

ii. Since Π is the set of all T-undecidable sentences of $\mathcal{L}$, and $\Pi \subset \Sigma$, then any sentence τ that does not belong to Σ is T-decidable, i.e. either $\tau \in T$ or $\neg\tau \in T$. Given the recursive axiomatisability of T and the recursiveness of Σ, for any sentence $\tau \in Stn_{\mathcal{L}}$, it is always possible to determine whether $\tau \in \Sigma$ or $\tau \in Stn_{\mathcal{L}} - \Sigma$. In the second case, since T is negation complete w.r.t. this subclass, there is an effective mechanism that establishes, for any sentence $\tau \in Stn_{\mathcal{L}} - \Sigma$, whether it is a T-theorem or not, based on the recursive enumeration of all formulae of T until either τ or its negation is obtained. As a result, T would be decidable in both Σ and $Stn_{\mathcal{L}} - \Sigma$, i.e. T would be a decidable theory.

iii. If $\tau' \in \Sigma' \cap \Pi$, then, according to the definition of Σ', there is a sentence $\tau \in \Sigma$ such that $T \models (\tau' \equiv \tau)$. Clearly, neither τ nor $\neg\tau$ can be an element of T, for otherwise either τ' or $\neg\tau'$ would be a T-theorem as well, in conflict with the undecidability of the elements of Π. Hence T is not negation complete w.r.t. Σ.

∎

Lemma E.5.3 *Let Σ be the class of all formulae of $\mathcal{L}_{PA}$ of the form $t_1 = 0 \vee t_2 = 1$ which can be rewritten into a formula of $\mathcal{L}_{SMA}$ by the strict application of the rules*

$$\begin{array}{lll} R_a & v_1 + v_2 = 0 & \Rightarrow \quad v_1 = 0 \wedge v_2 = 0 \\ R_b & v_1 + v_2 = 1 & \Rightarrow \quad (v_1 = 1 \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = 1) \end{array}$$

Then t_1 and t_2 must be members of the class of terms Θ defined as

$$\begin{array}{rcl} pol & := & sum | pol + pol \\ sum & := & 0|1|var|sum \times sum \\ var & := & x_1|y_1|z_1|x_2|\ldots \end{array}$$

PROOF. (By induction on the length of the rewriting sequence)

Base case. If ϕ' belongs to $\mathcal{L}_{SMA}$ (empty rewriting sequence), it already satisfies the conditions of the definition of Σ. As a result, t_1 and t_2 are terms of $\mathcal{L}_{SMA}$ and, therefore, belong to Θ, since $Trm_{\mathcal{L}_{SMA}} \subset \Theta$.

Step case. Assume that, for every formula $(t_1 = 0 \vee t_2 = 1)$ in $\mathcal{L}_{PA}$, if there is a rewriting sequence of the form

$$(t_1 = 0 \vee t_2 = 1) \overset{R_1}{\Rightarrow} \ldots \overset{R_n}{\Rightarrow} \psi$$

where ψ is a formula of $\mathcal{L}_{SMA}$ and $R_i \in \{R_a, R_b\}$, then t_1 and t_2 belong to Θ. Let $(t'_1 = 0 \vee t'_2 = 1)$ be a formula for which there is a rewriting sequence of length $n + 1$,

$$(t'_1 = 0 \vee t'_2 = 1) \overset{R'_1}{\Rightarrow} \ldots \overset{R'_n}{\Rightarrow} \psi' \overset{R'_{n+1}}{\Rightarrow} \psi'' \quad (*)$$

such that ψ'' is a formula of $\mathcal{L}_{SMA}$ and $R'_j \in \{R_a, R_b\}$. Then ψ' does not belong to $\mathcal{L}_{SMA}$, for otherwise the above sequence could be reduced to length n. Two cases must be taken into account.

If $\psi' \overset{R_a}{\Rightarrow} \psi''$, there must be an occurrence of an atom of the form $u_1 + u_2 = 0$ in ψ' such that u_1, u_2 are terms of $\mathcal{L}_{SMA}$. Given that, for any element of the sequence $(*)$, all the atoms have the form $\bar{t} = 0$ or $\bar{t} = 1$, where $\bar{t}$ is a summand contained in either t'_1 or t'_2, $u_1 + u_2$ must be a subexpression of either t'_1 or t'_2.

Without loss of generality, let $u_1 + u_2$ be a summand of t'_1. Let a new rewriting sequence be built from $(*)$ by the replacement of the chosen occurrence of $u_1 + u_2$ in t'_1 with a term u of $\mathcal{L}_{SMA}$. The resulting sequence,

$$(t'_1[\![^u/_{(u_1+u_2)}]\!] = 0 \vee t'_2 = 1) \overset{R'_1}{\Rightarrow} \ldots \overset{R'_n}{\Rightarrow} \psi'[\![^u/_{(u_1+u_2)}]\!]$$

has length n and a formula of $\mathcal{L}_{SMA}$ as final expression. According to the induction hypothesis, $t_1'[\![^u/_{(u_1+u_2)}]\!]$ and t_2' belong to Θ. Since u is a summand of $t_1'[\![^u/_{(u_1+u_2)}]\!]$, t_1' is also an element of Θ.

A similar proof applies to the case where $\psi' \stackrel{R_b}{\Rightarrow} \psi''$. ∎

Lemma E.5.4 *Let $\mathcal{L}$ be a first-order language, and $\mathcal{L}'$ be one of its sublanguages. Let $\mathcal{R}$ be a set of non-conditional* remove *rules for the deviant symbols of $\mathcal{L}$ w.r.t. $\mathcal{L}'$, such that the lhs expressions of all its rules are either terms or atomic formulae, and Σ be the $\mathcal{R}$-extension of $\mathcal{L}'$. If ϕ is a formula of $\mathcal{L}$, then*

$$\phi \in \Sigma \quad \textit{iff} \quad \phi^N \in \Sigma$$

where ϕ^N is the prenex disjunctive normal form of ϕ.

PROOF. Let Φ be the set of atomic subformulae of ϕ. Since the rules of $\mathcal{R}$ are applicable only to atoms or terms, ϕ is $\mathcal{R}$-reducible to $\mathcal{L}'$ iff each of the elements of Φ is also $\mathcal{R}$-reducible to $\mathcal{L}'$. Given that the process of prenex disjunctive normal forming does not alter any of the atomic components of a formula, Φ is also the set of atomic components of ϕ^N. As a result, ϕ is $\mathcal{R}$-reducible to $\mathcal{L}'$ iff ϕ^N is $\mathcal{R}$-reducible to $\mathcal{L}'$. ∎

E.6 Chapter 10

Lemma E.6.1 *Let ψ_1 and ψ_2 respectively be the formulae*

$$\begin{aligned} (\textstyle\bigwedge_{i=1}^{n} numberp(v_i)) &\supset \phi(v_1, \ldots, v_n) \\ (\textstyle\bigwedge_{i=1}^{n} numberp(v_i)) &\supset \phi^*(v_1, \ldots, v_n) \end{aligned}$$

where ϕ^* is the complement of ϕ. If ψ_1 is LA-valid, then ψ_2 is LA-invalid.

PROOF. Since the antecedent of ψ_1 is LA-satisfiable, there is an assignment α such that $\bigwedge_{i=1}^{n} numberp(v_i)[\alpha]$ is valid in LA. If ψ_1 is LA-valid, then ϕ is LA-valid for the same assingment. ϕ^* then is LA-invalid for α and, $\psi_2[\alpha]$ is invalid in LA as a result. Therefore ψ_2 is LA-invalid. ∎

Appendix F

Some General Purpose Plans

Decide1/1 is a general-purpose proof plan that incorporates the strategy for the extension of decidable sublanguages described in the main text. Whenever more than one decidable sublanguage is known, the function m_c is called to organise them into decreasing order of preference. The subplan *red_dec_cla*/2 then tries to perform the reduction. If it succeeds, the rewritten conjecture is eventually supplied to the corresponding decision procedure. Additional subplans are responsible for calling the function m_d and ordering deviant symbols with respect to the chosen sublanguage. The subplans *rem_dev_sym*/3-5 in particular control the removal of symbols and the elimination of disagreements, employing for this purpose the rule generation mechanism.

Decide2/1 has the same search control as *decide1*/1 for the selection of decidable sublanguages, the hierarchisation of the deviant symbols and the choice of *remove* rules. It is however interfaced to a stronger version of RGM that explores conjecture subexpressions other than the particular one directly involved in semantic matching.

Simplify/1 explores the fact that, even if a formula is not reducible to a decidable sublanguage, the rewriting process simplifies it, in the sense that the number of occurrences of deviant symbols usually decreases. When the reduction fails, the output of *simplify*/1 consists of the initial formula and one simplified formula for each sublanguage, corresponding to the failed reduction attempts. Since these expressions are all equivalent to each other, they can be supplied to an alternative proving strategy, e.g. induction.

An implementation for all three plans and their component subplans follows.

```
/*
 * Plan decide1/1
 */

plan(decide1(Thr),
if_then_else[mem_dec_cla(DecCla,Thr),
             dec_pro(DecCla,Thr),
             if_then[(dec_cla_lst(DecClaLst,Thr),
                      ord_dec_cla(OrdDecClaLst,DecClaLst,Thr)),
                     red_dec_cla(OrdDecClaLst,Thr)
                     then dec_pro(DecCla,Thr)]]).

plan(red_dec_cla(OrdDecClaLst,Thr),
if_then_else[OrdDecClaLst == [],
             fail,
             if_then[([DecCla|DecClaLst] = OrdDecClaLst,
                      psb_ord_dev_sym(PsbOrdDevSymPaiLst,DecCla,Thr)),
                     red_dec_cla(DecCla,Thr,PsbOrdDevSymPaiLst)
                     or red_dec_cla(DecClaLst,Thr)]]).

plan(red_dec_cla(DecCla,Thr,PsbOrdDevSymPaiLst),
if_then_else[PsbOrdDevSymPaiLst == [],
             fail,
             if_then[[OrdDevSymPai|OrdDevSymPaiLst] = PsbOrdDevSymPaiLst,
                     rem_dev_sym(OrdDevSymPai,DecCla,Thr)
                     or red_dec_cla(DecCla,Thr,OrdDevSymPaiLst)]]).

plan(rem_dev_sym(OrdDevSymPai,DecCla,Thr),
if_then_else[OrdDevSymPai == [],
             idmethod,
             if_then[([[DevSym,Ari]|DevSymPaiLst] = OrdDevSymPai,
                      pos_sym_exp(DevSym,Ari,PosDevSym),
                      ord_rem_rul(DevSym,Ari,PosDevSym,DecCla,
                                  Thr,RemRulLst)),
                     rem_dev_sym(DevSym,Ari,PosDevSym,RemRulLst,Thr)
                     then
                     if_then_else[mem_dec_cla(DecCla,Thr),
                                  dec_pro(DecCla,Thr),
                                  if_then_else[
                                        pos_sym_exp(DevSym,Ari,_),
                                        rem_dev_sym(OrdDevSymPai,DecCla,Thr),
                                        rem_dev_sym(DevSymPaiLst,
                                                    DecCla,Thr)]]]]).
```

```
plan(rem_dev_sym(DevSym,Ari,PosDevSym,RemRulLst,Thr),
if_then_else[RemRulLst == [],
             fail,
             if_then[[RemRul|RulLst] = RemRulLst,
                     partial_remove(PosDevSym,RemRul) orelse
                     (gen_rem_rul(RemRul,DevSym,Ari,PosDevSym)
                      orelse rem_dev_sym(DevSym,Ari,PosDevSym,
                                         RulLst,Thr))]]).

plan(gen_rem_rul(RemRul,DevSym,Ari,PosDevSym),
if_then[spc_trv_mat_cbl_rul(DevSym,Ari,PosDevSym,RemRul,NewRemRul),
        partial_remove(PosDevSym,NewRemRul)]).




/*
 * Plan decide2/1
 */

plan(decide2(Thr),
if_then_else[mem_dec_cla(DecCla,Thr),
             dec_pro(DecCla,Thr),
             if_then[(dec_cla_lst(DecClaLst,Thr),
                      ord_dec_cla(OrdDecClaLst,DecClaLst,Thr)),
                     red_dec_cla2(OrdDecClaLst,Thr)
                     then dec_pro(DecCla,Thr)]]).

plan(red_dec_cla2(OrdDecClaLst,Thr),
if_then_else[(OrdDecClaLst == []),
             fail,
             if_then[([DecCla|DecClaLst] = OrdDecClaLst,
                      psb_ord_dev_sym(PsbOrdDevSymPaiLst,DecCla,Thr)),
                     red_dec_cla2(DecCla,Thr,PsbOrdDevSymPaiLst)
                     or red_dec_cla2(DecClaLst,Thr)]]).

plan(red_dec_cla2(DecCla,Thr,PsbOrdDevSymPaiLst),
if_then_else[(PsbOrdDevSymPaiLst == []),
             fail,
             if_then[[OrdDevSymPai|OrdDevSymPaiLst] = PsbOrdDevSymPaiLst,
                     rem_dev_sym2(OrdDevSymPai,DecCla,Thr)
                     or red_dec_cla2(DecCla,Thr,OrdDevSymPaiLst)]]).
```

```
plan(rem_dev_sym2(OrdDevSymPai,DecCla,Thr),
if_then_else[(OrdDevSymPai == []),
             idmethod,
             if_then[([[DevSym,Ari]|DevSymPaiLst] = OrdDevSymPai,
                      pos_sym_exp(DevSym,Ari,PosDevSym),
                      ord_rem_rul(DevSym,Ari,PosDevSym,DecCla,
                                  Thr,RemRulLst)),
                     rem_dev_sym2(DevSym,Ari,PosDevSym,RemRulLst,Thr)
                     then
                     if_then_else[mem_dec_cla(DecCla,Thr),
                                  dec_pro(DecCla,Thr),
                                  if_then_else[
                                      pos_sym_exp(DevSym,Ari,_),
                                      rem_dev_sym2(OrdDevSymPai,DecCla,Thr),
                                      rem_dev_sym2(DevSymPaiLst,
                                                   DecCla,Thr)]]]]).

plan(rem_dev_sym2(DevSym,Ari,PosDevSym,RemRulLst,Thr),
if_then_else[(RemRulLst == []),
             fail,
             if_then[[RemRul|RulLst] = RemRulLst,
                     partial_remove(PosDevSym,RemRul) orelse
                     (if_then[can_sub_exp_lst2(RemRul,PosDevSym,
                                OrdPosSubExpLst),
                              gen_rem_rul2(RemRul,PosDevSym,OrdPosSubExpLst)
                              orelse rem_dev_sym2(DevSym,Ari,PosDevSym,
                                                  RulLst,Thr)])]]).

plan(gen_rem_rul2(RemRul,PosDevSym,OrdPosSubExpLst),
if_then_else[(OrdPosSubExpLst == []),
             fail,
             if_then[[[PosSubExp,_]|PosSubExpLst] = OrdPosSubExpLst,
                     if_then_else[trv_mat_cbl_rul2(PosSubExp,RemRul,
                                                   NewRemRul),
                                  partial_remove(PosDevSym,NewRemRul),
                                  gen_rem_rul2(RemRul,PosDevSym,
                                               PosSubExpLst)]]]).
```

```
/*
 * Plan simplify/1
 */

plan(simplify(Thr),
if[mem_dec_cla_lst(_,DecCla,Thr),
   dec_pro(Pos,DecCla,Thr),
   if[(dec_cla_lst(DecClaLst,Thr),
       ord_dec_cla_lst(_,OrdDecClaLst,DecClaLst,Thr)),
      simplify1(OrdDecClaLst,Thr)
      thn (if[mem_dec_cla_lst(_,DecCla,Thr),
              dec_pro(Pos,DecCla,Thr),
              simplify2(OrdDecClaLst,Thr)
              thn (if[mem_dec_cla_lst(_,DecCla,Thr),
                      dec_pro(Pos,DecCla,Thr),
                      idmthlst])])]]).

plan(simplify1(OrdDecClaLst,Thr),
if[OrdDecClaLst == [],
   synsmp,
   if[([DecCla|DecClaLst] = OrdDecClaLst,
       dec_cla(DecCla,Thr,DevSymLst),
       ord_exp_lst_cla(DecCla,Thr,[Pos|_])),
      duplicate(Pos)
      thn (((rpeat random_remove2(Pos,DevSymLst,Thr))
            orels idmthlst)
           thn (if[mem_dec_cla_lst(Pos,DecCla,Thr),
                   synsmp,
              simplify1(DecClaLst,Thr)]))]]).

plan(simplify2(OrdDecClaLst,Thr),
synsmp
thn (if[OrdDecClaLst == [],
        idmthlst,
        if[mem_dec_cla_lst(_,DecCla,Thr),
           idmthlst,
           if[([DecCla|_] = OrdDecClaLst,
               ord_exp_lst_cla(DecCla,Thr,[Pos|_])),
              duplicate(Pos)
      thn spf_dec_cla(Pos,OrdDecClaLst,Thr)]]])).

plan(spf_dec_cla(Pos,OrdDecClaLst,Thr),
if[([DecCla|DecClaLst] = OrdDecClaLst,
    dec_cla(DecCla,Thr,DevSymLst)),
   spf_dec_cla(Pos,DecCla,Thr,DevSymLst)
   thn simplify2(DecClaLst,Thr)]).
```

```
plan(spf_dec_cla(Pos,DecCla,Thr,DevSymLst),
if[DevSymLst == [],
   idmthlst,
   if[(DevSymLst = [[DevSym,Ari]|SymLst],
       occ_sym_exp_lst(Pos,DevSym,Ari,[PosDevSym|_])),
      (rem_dev_sym(Pos,DevSym,Ari,PosDevSym,DecCla,Thr)
       thn (if[mem_dec_cla_lst(Pos,_,Thr),
               idmthlst,
               spf_dec_cla(Pos,DecCla,Thr,DevSymLst)
               orels spf_dec_cla(Pos,DecCla,Thr,SymLst)]))
       orels idmthlst,
       if[DevSymLst = [_|SymLst],
          spf_dec_cla(Pos,DecCla,Thr,SymLst)]]]).

plan(rem_dev_sym(Pos,DevSym,Ari,PosDevSym,DecCla,Thr),
if[ord_rem_rul_lst(Pos,DevSym,Ari,PosDevSym,DecCla,Thr,RemRulLst),
   rem_dev_sym2(Pos,DevSym,Ari,PosDevSym,RemRulLst,Thr)]).

plan(rem_dev_sym2(Pos,DevSym,Ari,PosDevSym,RemRulLst,Thr),
if[RemRulLst == [],
   faillst,
   if[[RemRul|RulLst] = RemRulLst,
      prt_remove(Pos,PosDevSym,RemRul)
      orels (gen_rem_rul_lst(Pos,RemRul,DevSym,Ari,PosDevSym)
             orels rem_dev_sym2(Pos,DevSym,Ari,PosDevSym,RulLst,Thr))]]).

plan(gen_rem_rul_lst(Pos,RemRul,DevSym,Ari,PosDevSym),
if[trv_mat_cbl_rul_lst(Pos,DevSym,Ari,PosDevSym,RemRul,NewRemRul),
   prt_remove(Pos,PosDevSym,NewRemRul)]).

plan(random_remove2(Pos,[[Sym,Ari]|SymLst],Thr),
if[(cjt_lst(CjtLst),
    mem_at(Cjt,CjtLst,Pos),
    occ_sym_exp(Cjt,Sym,Ari,[_|_])),
   remove_(Pos,Sym,Ari,Thr)
   orels random_remove2(Pos,SymLst,Thr),
   random_remove2(Pos,SymLst,Thr)]).
```

Appendix G

Arithmetical Conjectures

Four main experiments have been conducted in the course of the development and testing of the general-purpose plans described in chapter 9. Section G.1 lists the conjectures employed in the measurement of the effect on efficiency of several control features available in proof plans such as *decide*/1. Section G.2 presents the complete set of arithmetical lemmas described as representative of the domain of verification conditions by Boyer and Moore. Section G.3 has some of the randomly generated conjectures used to assess the performance of the proof plans for simplification in subsets of formulae of various degrees of syntactic complexity. Finally, section G.4 presents the complete set of quantifier-free formulae employed in the comparative assessment of the performances of *Nqthm*'s simplifier, *simplify*/1 and *weak_simplify*/1.

G.1 Development Sample

The formulae listed in table G.1 have been supplied to the general-purpose proof plans and the rewriters described in chapter 9 with the purpose of comparing their performances. All formulae belong to the language

$$\{0, 1, \mathrm{s}, +, \times, \exp, <, \leq\}$$

Natural numbers (other than 0 and 1) are employed in the mentioned table as abbreviations. The sample has been partitioned in three groups: series 100 contains conditional equations, series 200, conditional inequalities, and series 300, conditional systems of equations and/or inequalities. In the first series, formula ϕ_{102} has been derived from a problem proposed by Boyer and Moore[1]. Formulae ϕ_{109} and ϕ_{114} are variations of this problem, obtained by the application of the associativity of multiplication.

Experimental results concerning the rewriters and *decide*/1 are shown in tables G.2, G.3, G.4 and G.5. Additional results for the deciders and simplifiers are presented in tables G.6, G.7 and G.8.

[1] See [Boyer & Moore 88], p. 102.

No.	Conjecture
ϕ_{101}	$(x)(y)(z)(x \neq 0 \supset x \times (y+z) = x)$
ϕ_{102}	$(x)(y)(x \neq 0 \supset x^2 \times x^2 + 2x^2 \times y = x^2)$
ϕ_{103}	$(x)(y)(z)(w)(x \neq 0 \supset x \times (x \times y \times z) + x \times (w^2 \times y^2 \times x^2) = x)$
ϕ_{104}	$(x)(y)(x \neq 0 \supset x^2 = x \times 1 + 1 \times x + y \times x)$
ϕ_{105}	$(x)(y)(z)(x \neq 0 \supset z \neq 0 \supset x \times y \times y \times z + x \times y \times z = z \times x)$
ϕ_{106}	$(x)(y)(z)(w)(s(x) \times y = x \times y + z \times z \times w)$
ϕ_{107}	$(x)(y)(x \neq 0 \supset x^2 + 2x^2 \times y = x^2)$
ϕ_{108}	$(x)(y)(z)(w)(x + y < 1 \wedge z \times w = 0 \supset x^2 + y^2 + z^2 + w^2 = 1)$
ϕ_{109}	$(x)(y)(x \neq 0 \supset x^4 + 2x^2 \times y = x^2)$
ϕ_{110}	$(x)(y)(z)(x \neq 0 \supset (y \neq 0) \supset (x \times y) \times (x + y) + (x \times y) \times (y + z) = x \times y)$
ϕ_{111}	$(x)(y)(x \neq 0 \supset x^2 \times x^2 + x^2 \times y = x^2)$
ϕ_{112}	$(x)(y)(z)(x \neq 0 \supset z \neq 0 \supset x \times y \times y \times z + x \times y \times z = x \times z)$
ϕ_{113}	$(x)(y)(z)(w)(x + y < 1 \wedge z \times w = x^2 \supset x^2 + y^2 + z^2 + w^2 = 1)$
ϕ_{114}	$(x)(y)(x \neq 0 \supset x \times (x^2 \times x) + 2x^2 \times y = x^2)$
ϕ_{115}	$(x)(y)(z)(x \neq 0 \supset y \neq 0 \supset x \times y \times y + x \times y \times z = x \times y)$
ϕ_{116}	$(y)(y \neq 0 \supset y^2 + 3 = 3 + 2y)$
ϕ_{117}	$(x)(y)(z)(w)(x \neq 0 \supset y \neq 0 \supset z \neq 0 \supset y \times z + y \times z \times z = y \times (z \times (w + x)))$
ϕ_{118}	$(x)(y)(z)(w)(y \neq 0 \supset x + y^2 \times w = y^2 \times z + x)$
ϕ_{119}	$(x)(y)(z)(y \neq 0 \supset x + y^2 = y^2 + x)$
ϕ_{120}	$(x)(y)(z)(x \neq 0 \supset y \neq 0 \supset x \times y \times y + x \times z \times y = x \times y)$
ϕ_{121}	$(x)(y)(z)(w)(x \neq 0 \supset x \times y + x \times w = x \times z + x^2)$
ϕ_{122}	$(x)(z)(x \neq 0 \supset z \neq 0 \supset x^2 + (1 + z) = x \times (x + 1))$
ϕ_{123}	$(x)(y)(x \neq 0 \supset x^2 = (x + x) + y \times x)$
ϕ_{124}	$(x)(y)(z)(w)(x \neq 0 \supset x \times (x \times y \times z) + x \times (w^2 \times y \times y \times x \times x) = x)$
ϕ_{125}	$(x)(y)(x \neq 0 \supset x^2 = (x \times 1 + x \times 1) + y \times x)$
ϕ_{126}	$(x)(y)(z)(x \neq 0 \supset x \times (y \times (x + y)) + x \times (z^2 \times (x + y)) = x)$
ϕ_{127}	$(x)(y)(z)(x \neq 0 \supset y \neq 0 \supset x \times (y + z) \times y = x \times y)$
ϕ_{128}	$(x)(y)(z)(w)(z \neq 0 \supset z^2 + (y \times x^2 + z \times x^2) = x^2 \times (y + z) + w \times z)$
ϕ_{201}	$(x)(y)(z)(x \neq 0 \supset (x + y) \neq 0 \supset x \times (y \times (x + y)) + x \times (z^2 \times (x + y)) < x \times (x + y))$
ϕ_{202}	$(x)(y)(z)(x + y \leq x)$
ϕ_{203}	$(x)(y)(x \neq 0 \supset x \times (x^2 \times x) + 2x^2 \times y \leq x^2)$
ϕ_{204}	$(x)(y)(z)((x \neq 0) \supset (x \times y^{x^2+2x} = x))$
ϕ_{205}	$(x)(y)(w)(w \neq 0 \supset x^y \times w = 1 \times w)$
ϕ_{301}	$(x)(y)(z)(w)(y \neq 0 \supset z \neq 0 \supset (x = 2y \ \wedge \ y^2 + 3 = 3 + 2y \ \wedge \ z \leq 1 + 2w) \supset$ $\supset (2x + 3y = z \ \wedge \ z^3 \leq z^2))$
ϕ_{302}	$(x)(y)(z)((x \neq 0) \supset (y \neq 0) \supset (x \times (y + z) = x \times y \ \wedge \ y^2 \times w = z^3))$
ϕ_{303}	$(x)(y)(z)((x \neq 0) \supset (x \times (y + z) = x \times y \ \wedge \ y + w = 3z))$
ϕ_{304}	$(x)(y)(z)(0 = x \ \wedge \ y^2 + z \times w^2 = x)$
ϕ_{305}	$(x)(y)(z)(x = 0 \ \wedge \ y^2 + z = x)$

Series 100 – (Conditional) equations
Series 200 – (Conditional) inequalities
Series 300 – (Conditional) systems of equations and/or inequalities

Table G.1: Arithmetical conjectures - Development stage

Conjecture	Execution Time (in s)						
	rewrite1/4	rewrite2/5	rewrite3/5	rewrite4/5	rewrite5/5	rewrite6/5	decide1/1
ϕ_{101}	2.5	2.5	2.5	2.5	2.5	2.5	77.1
ϕ_{102}	□	728.3	7,764.0	5,645.8	652.9	684.6	425.6
ϕ_{103}	□	883.0	4,322.8	3,598.7	406.7	426.0	288.1
ϕ_{104}	□	1,271.4	358.6	280.5	827.3	873.9	472.8
ϕ_{105}	□	3,514.4	1,323.8	961.5	2,113.4	2,201.5	1,218.3
ϕ_{106}	□	969.2	1,234.0	1,822.9	1,131.1	565.2	38.0
ϕ_{107}	5.7	5.7	5.7	5.7	5.7	5.7	10.6
ϕ_{108}	45.1	45.1	45.1	45.1	45.1	45.1	90.9
ϕ_{109}	□	747.9	7,916.5	5,788.9	656.4	681.2	430.6
ϕ_{110}	□	1,168.0	426.7	305.2	1,562.9	1,624.1	506.4
ϕ_{111}	□	1,162.9	6,284.3	4,661.2	812.0	844.9	819.9
ϕ_{112}	□	3,560.3	1,318.2	959.9	2,129.3	2,205.7	1,186.1
ϕ_{113}	46.0	46.0	46.0	46.0	46.0	46.0	90.3
ϕ_{114}	□	753.8	11,762.7	8,428.7	671.0	700.9	428.8
ϕ_{115}	□	956.5	310.3	259.4	578.9	598.8	527.3
ϕ_{116}	□	2,547.1	1,477.9	1,399.6	1,401.9	1,461.9	117.7
ϕ_{117}	□	6,793.1	2,575.7	2,026.7	3,496.3	3,630.8	1,259.8
ϕ_{118}	□	989.7	672.6	585.2	522.3	539.3	16.8
ϕ_{119}	□	796.6	296.0	234.5	424.8	436.9	14.4
ϕ_{120}	□	3,117.5	1,013.7	724.8	1,628.9	1,688.0	775.6
ϕ_{121}	□	2,403.3	704.3	619.5	1,876.5	1,957.5	1,113.8
ϕ_{122}	□	2,969.3	1,447.0	1,254.3	1,253.2	1,302.9	1,059.5
ϕ_{123}	□	1,236.4	358.1	277.1	816.7	859.6	481.7
ϕ_{124}	□	878.3	3,959.0	3,253.7	410.9	429.3	270.4
ϕ_{125}	□	1,273.2	353.6	276.6	831.4	861.8	471.2
ϕ_{126}	□	761.9	9,264.2	4,801.6	1,450.7	1,509.3	527.0
ϕ_{127}	□	2,848.4	988.5	728.2	2,141.9	2,232.8	1,210.4
ϕ_{128}	□	8,313.9	5,879.7	5,796.7	7,687.1	8,019.4	1,820.3
ϕ_{201}	□	605.7	5,801.4	3,833.0	213.7	222.8	77.0
ϕ_{202}	0.4	0.4	0.4	0.4	0.4	0.4	0.8
ϕ_{203}	□	3,576.0	262,849.3	190,698.7	1,569.4	1,634.7	612.2
ϕ_{204}	15.2	15.2	15.2	15.2	15.2	15.2	306.6
ϕ_{205}	□	1,801.4	177.9	615.2	155.3	161.2	89.0
ϕ_{301}	□	22,186.3	9,731.3	18,231.6	18,206.1	18,965.3	521.9
ϕ_{302}	11.2	11.2	11.2	11.2	11.2	11.2	345.3
ϕ_{303}	5.1	5.1	5.1	5.1	5.1	5.1	9.7
ϕ_{304}	□	□	□	□	□	□	□
ϕ_{305}	□	□	□	□	□	□	□

Series 100 – (Conditional) equations
Series 200 – (Conditional) inequalities
Series 300 – (Conditional) systems of equations and/or inequalities

□ – failure

Table G.2: Time Performances – Rewriters & *decide1*/1

	Time Performance (sample size - 38)						
	rewrite1/4	rewrite2/5	rewrite3/5	rewrite4/5	rewrite5/5	rewrite6/5	decide1/1
MT (s)	16.4	2,192.9	9,741.7	7,450.0	1,548.9	1,596.0	492.0
SD (s)	18.9	3,889.4	43,515.7	31,608.6	3,176.7	3,313.4	456.3
SR (%)	21.1	94.7	94.7	94.7	94.7	94.7	94.7
BP (%)	21.1	21.1	21.1	47.4	23.7	21.1	44.7
SP (%)	0.0	2.6	21.1	10.5	26.3	2.6	31.6
WP (%)	0.0	44.7	23.7	2.6	0.0	2.6	0.0

MT mean time
SD sample standard deviation
SR success rate
BP best time performance
SP second best time performance
WP worst time performance

Table G.3: Success Rates – Rewriters & *decide1*/1

From	To	Sample Size	Time Variation (% of sample)		
			Reduction	Increase	Unchanged
rewrite2/5	rewrite3/5	36	50.0	27.8	22.2
rewrite2/5	rewrite4/5	36	50.0	27.8	22.2
rewrite2/5	rewrite5/5	36	69.4	8.3	22.2
rewrite2/5	rewrite6/5	36	72.2	5.6	22.2
rewrite2/5	decide1/5	36	77.8	22.2	0.0
rewrite3/5	rewrite4/5	36	69.4	8.3	22.2
rewrite3/5	rewrite5/5	36	38.9	38.9	22.2
rewrite3/5	rewrite6/5	36	38.9	38.9	22.2
rewrite3/5	decide1/5	36	58.3	41.7	0.0
rewrite4/5	rewrite5/5	36	38.9	38.9	22.2
rewrite4/5	rewrite6/5	36	33.3	44.4	22.2
rewrite4/5	decide1/1	36	50.0	50.0	0.0
rewrite5/5	rewrite6/5	36	2.8	75.0	22.2
rewrite5/5	decide1/1	36	75.0	25.0	0.0
rewrite6/5	decide1/1	36	77.8	22.2	0.0

Table G.4: Cumulative Effect – Rewriters & *decide1*/1

Conjecture	Time Fraction (%)	Conjecture	Tim Fraction (%)
ϕ_{102}	65.2	ϕ_{122}	84.5
ϕ_{103}	70.8	ϕ_{124}	65.8
ϕ_{106}	6.7	ϕ_{126}	69.2
ϕ_{109}	65.6	ϕ_{128}	31.4
ϕ_{114}	63.9	ϕ_{201}	36.0
ϕ_{116}	8.4	ϕ_{203}	39.0
ϕ_{117}	62.2	ϕ_{205}	57.3
ϕ_{118}	3.2	ϕ_{301}	5.4
ϕ_{119}	6.1		

$$\text{Time fraction} = \frac{\text{time consumed by } \textit{decide1}/1}{\text{time consumed by the 2nd fastest system}} \times 100\%$$

The above conjectures are all those for which *decide1*/1 exhibited the best time performance. The percentages indicate the fraction of the running time consumed by the second fastest system (which may vary from one conjecture to another) that was required by *decide1*/1. The average fraction for this subsample (43.6%) shows that the decider provided a substantial time reduction in the average.

Table G.5: *Decide1*/1 × Second best performance

Conjecture	Execution Time (in s)			
	decide1/1	decide2/1	simplify/1	weak_ simplify/1
ϕ_{101}	77.1	222.2	3.1	3.1
ϕ_{102}	425.6	1,482.4	435.5	334.1
ϕ_{103}	288.1	14,372.5	298.1	63.7
ϕ_{104}	472.8	2,851.2	487.1	□
ϕ_{105}	1,218.3	4,464.4	1,180.4	□
ϕ_{106}	38.0	36.6	21.2	20.6
ϕ_{107}	10.6	10.9	3.1	3.1
ϕ_{108}	90.9	88.0	20.7	21.5
ϕ_{109}	430.6	1,272.0	432.2	340.7
ϕ_{110}	506.4	1,912.2	478.9	292.3
ϕ_{111}	819.9	16,930.8	816.1	345.1
ϕ_{112}	1,186.1	4,482.3	1,190.9	□
ϕ_{113}	90.3	91.8	21.9	22.3
ϕ_{114}	428.8	1,320.3	444.7	350.0
ϕ_{115}	527.3	1,808.0	535.8	292.9
ϕ_{116}	117.7	363.0	121.8	□
ϕ_{117}	1,259.8	13,596.2	1,273.6	□
ϕ_{118}	16.8	18.2	23.1	22.2
ϕ_{119}	14.4	14.8	19.0	19.0
ϕ_{120}	775.6	2,535.0	785.4	532.1
ϕ_{121}	1,113.8	21,078.9	1,120.8	513.6
ϕ_{122}	1,059.5	10,502.3	1,077.0	□
ϕ_{123}	481.7	2,841.7	485.7	□
ϕ_{124}	270.4	7,917.6	275.7	64.2
ϕ_{125}	471.2	2,897.5	490.7	□
ϕ_{126}	527.0	4,316.6	535.0	348.4
ϕ_{127}	1,210.4	8,346.0	1,210.3	661.6
ϕ_{128}	1,820.3	21,651.2	1,830.8	870.4
ϕ_{201}	77.0	113.1	80.4	73.4
ϕ_{202}	0.8	0.8	0.7	0.7
ϕ_{203}	612.2	1,856.6	576.7	466.7
ϕ_{204}	306.6	854.7	21.7	22.3
ϕ_{205}	89.0	568.2	57.7	50.4
ϕ_{301}	521.9	2,156.5	272.1	□
ϕ_{302}	345.3	2,277.3	19.5	18.7
ϕ_{303}	9.7	9.8	2.8	2.8
ϕ_{304}	□	1,173.5	□	□
ϕ_{305}	□	3,352.6	□	□

Series 100 – (Conditional) equations
Series 200 – (Conditional) inequalities
Series 300 – (Conditional) systems of equations and/or inequalities
□ – failure

Table G.6: Time Performances – Deciders & Simplifiers

	Time Performance (sample size - 38)			
	decide1/5	decide2/5	simplify/1	weak_ simplify/1
MT (s)	492.0	4,204.9	456.5	213.2
SD (s)	456.3	5,907.5	463.9	241.3
SR (%)	94.7	100.0	94.7	71.1
BP (%)	23.7	5.3	23.7	57.9
SP (%)	44.7	7.9	36.8	7.9
WP (%)	5.3	81.6	5.3	2.6

MT mean time
SD sample standard deviation
SR success rate
BP best time performance
SP second best time performance
WP worst time performance

Table G.7: Success Rates – Deciders & Simplifiers

From	To	Sample Size	Time Variation (% of sample)		
			Reduction	Increase	Unchanged
decide1/5	decide2/5	36	5.6	91.7	2.8
decide1/5	simplify/1	36	44.4	55.6	0.0
decide1/5	weak_simplify/1	27	92.6	7.4	0.0
decide2/5	simplify/1	36	94.4	5.6	0.0
decide2/5	weak_simplify/1	27	92.6	7.4	0.0
simplify/1	weak_simplify/1	27	70.4	11.1	18.5

Table G.8: Cumulative Effect – Deciders & Simplifiers

G.2 Verification Conditions

The arithmetical lemmas listed in tables G.9 and G.10 have been taken from appendix A of [Boyer & Moore 79]. Their original enumeration has been preserved in the above mentioned tables. Lemmas 317 and 368 have been broken into respectively two and three new lemmas, due to the fact that they were originally conjunctions of formulae[2]. Table G.11 describes the role of each of the two main modules of *Nqthm* in the process of proving these lemmas. Tables G.12 and G.13 describe the time performances of the plan *weak_simplify*/1 and the subplan *simplify1*/2 for the above set of lemmas.

G.3 Randomly Generated Conjectures

One randomly generated formula for each of the depths examined in the experiment has been included in tables G.15 and G.16. All formulae belong to the first-order language described in section G.1, and have been represented according to the syntax detailed in table G.14. The experiments have been conducted with the general-purpose proof plan *simplify*/1. In the entry for each depth, `Cjt` indicates the randomly generated conjecture, whereas `RewCjt` is the resulting simplified formula. `Strict_sum` and `strict_multiplication` respectively denote the decidable sublanguages $\mathcal{L}_{PrA^-}$ and $\mathcal{L}_{SMA}$. Each new *remove* rule generated in the process is represented as a list of three elements, the lhs expression, the rhs expression and a condition. The results for the whole random sample are described in tables G.17 and G.18.

G.4 Quantifier-free Conjectures

Ten quantifier-free conjectures for each depth ranging from 3 to 10 have been randomly generated and supplied to *Nqthm* and to two proof plans for simplification. They belong to the same first-order language defined in section G.1, as it can be observed below.

Depth 3

qtf0301 $x_6 \leq 1 \wedge x_9 \leq 0$
qtf0302 $(x_9 \leq 1 \equiv 0 \leq x_6)$
qtf0303 $(x_3 \leq 1 \equiv x_5 \leq 0)$
qtf0304 $0 \leq 1 \supset \bot$
qtf0305 $1 \leq 1 \vee 1 < x_8$
qtf0306 $x_5 < 0 \vee 1 \leq 1$
qtf0307 $1 \leq 1 \supset \bot$
qtf0308 $x_8 \leq x_7 \supset \bot$
qtf0309 $1 \exp 1 < x_8$
qtf0310 $(x_9 \leq 0 \equiv 1 \leq x_5)$

---

[2] It has to be taken into account that

$$T \models \phi_1 \wedge \phi_2 \quad \text{iff} \quad T \models \phi_1 \text{ and } T \models \phi_2$$

No.	Theorem
37	$x \neq 0 \supset x - 1 < x$
100	$\text{even}(\text{double}(x))$
102	$\text{half}(\text{double}(x)) = x$
103	$\text{even}(x) \supset \text{double}(\text{half}(x)) = x$
109	$x^{y+z} = x^y \times x^z$
110	$x^{y \times z} = (x^y)^z$
113	$(x \leq y) \vee (y \leq x)$
120	$\text{half}(x) \leq x$
168	$x < y \supset y - \text{s}(x) < y - x$
169	$(x - y < x) \equiv (x \neq 0 \wedge y \neq 0)$
170	$x \neq 0 \wedge y \neq 0 \supset x - y < x$
179	$x \not< (x - y)$
180	$\text{rmdr}(y, 1) = 0$
182	$(\text{rmdr}(x, y) < y) \equiv (y \neq 0)$
183	$(y \not< x) \supset x - y = 0$
184	$\text{rmdr}(x, x) = 0$
185	$y \neq 0 \supset \text{rmdr}(x, y) + (y \times (x/y)) = x$
186	$(x + y = 0) \equiv (x = 0 \wedge y = 0)$
187	$(x + y = x) \equiv (y = 0)$
188	$z \neq 0 \wedge y \neq 0 \supset x < z + (y \times x)$
189	$(x/y < x) \equiv (x \neq 0 \wedge (y = 0 \vee y \neq 1))$
190	$(\text{rmdr}(x, y) < x) \equiv (x \neq 0 \wedge y \neq 0 \wedge x \not< y)$
191	$x \neq 0 \wedge y \neq 0 \wedge y \neq 1 \supset x/y < x$
197	$y \leq x \supset \neg(x \leq y)$
198	$\text{gcd}(x, y) = \text{gcd}(y, x)$
202	$y \neq 0 \wedge y < x \supset \text{rmdr}(x, y) < x$
208	$(x + y) - x = y$
209	$(x + y) - (x + z) = y - z$
210	$(y \times x) - (w \times x) = (x \times (y - w))$
212	$\text{rmdr}(x \times z, z) = 0$
213	$(y + (x + z)) - x = (y + z)$
215	$\text{s}(y + z) - z = \text{s}(y)$
216	$y \neq 0 \wedge y \neq 1 \supset \text{rmdr}(\text{s}(x \times y), y) \neq 0$
217	$\text{rmdr}(x, z) = 0 \wedge \text{rmdr}(y, z) = 0 \supset \text{rmdr}(x + y, z) = 0$
218	$\text{rmdr}(x, z) = 0 \wedge \text{rmdr}(y, z) \neq 0 \supset \text{rmdr}(x + y, z) \neq 0$
219	$\text{rmdr}(x, z) = 0 \supset (\text{rmdr}(x + y, z) = 0) \equiv (\text{rmdr}(y, z) = 0)$
220	$\text{rmdr}(x, z) = 0 \supset (\text{rmdr}(y + x, z) = 0) \equiv (\text{rmdr}(y, z) = 0)$
221	$(x - y = 0) \equiv (y \not< x)$
222	$x \leq y \supset (x + (y - x) = y)$
225	$\text{rmdr}(x, z) = 0 \supset (\text{rmdr}(y - x, z) = 0) \equiv (x < y \supset \text{rmdr}(y, z) = 0)$
226	$(x \times z < y \times z) \equiv (z \neq 0 \wedge x < y)$
227	$\text{gcd}(x \times z, y \times z) = z \times \text{gcd}(x, y)$
228	$(\text{rmdr}(x, \text{gcd}(x, y)) = 0) \equiv (\text{rmdr}(y, \text{gcd}(x, y)) = 0)$

Table G.9: Arithmetical lemmas proved by *Nqthm* (I)

No.	Theorem
229	$(x \neq 0 \wedge y \neq 0 \wedge z \mid x \wedge z \mid y) \supset (z \leq \gcd(x, y))$
237	$(x \neq 0 \wedge y \neq 0 \wedge z \neq 0) \supset (y < (x \times y) + (x \times z))$
279	$(y < x \wedge \neg \mathrm{prime}_1(x, y) \wedge x \neq 0 \wedge x \neq 1 \wedge y \neq 0) \supset (\mathrm{gfc}(x, y) < x)$
280	$(y < x \wedge \neg \mathrm{prime}_1(x, y) \wedge x \neq 0 \wedge x \neq 1 \wedge y \neq 0) \supset$
	$\supset (\mathrm{rmdr}(x, \mathrm{gfc}(x, y)) = 0)$
283	$(\mathrm{gfc}(x, y) = 0) \equiv ((y = 0 \vee y = 1) \wedge x = 0)$
284	$(\mathrm{gfc}(x, y) = 1) \equiv (x = 1)$
292	$(x \neq 0 \wedge x \mid y) \supset (x \times (y/x) = y)$
293	$(x \neq 0 \wedge x < y) \supset (y/x \neq 0)$
299	$(\mathrm{rmdr}(\mathrm{s}(x), z) = 0 \wedge z \neq \mathrm{s}(x) \wedge \neg \mathrm{prime}_1(\mathrm{s}(x), (x - z) + z)) \supset$
	$\supset (\neg \mathrm{prime}_1(\mathrm{s}(x), x))$
300	$(z \neq 1 \wedge z \neq \mathrm{s}(x) \wedge \mathrm{rmdr}(\mathrm{s}(x), z) = 0) \supset (\neg \mathrm{prime}_1(\mathrm{s}(x), z + w))$
301	$(z \neq 1 \wedge z \neq x \wedge z \mid x) \supset (\neg \mathrm{prime}_1(x, \mathrm{pr}(x)))$
302	$(\gcd(z, x) = y) \supset (\mathrm{rmdr}(z, y) = 0)$
303	$(\mathrm{rmdr}(z, x) \neq 0) \supset (\gcd(z, x) \neq x)$
304	$(x = z \times y) \supset (\mathrm{rmdr}(x, z) = 0)$
305	$y = 1 \supset x = x \times y$
306	$(y = z \times x) \supset (\gcd(y, w \times x) = x \times \gcd(w, z))$
307	$(x \nmid y \wedge y = \gcd(x \times y, z \times y)) \supset (w \times x \neq z \times x)$
308	$(x \nmid y \wedge \mathrm{prime}_1(x, \mathrm{pr}(x))) \supset (\gcd(y, x) = 1)$
309	$(\mathrm{prime}(x) \wedge x \nmid z \wedge x \nmid y) \supset (x \times w \neq y \times z)$
310	$(x \times y/x \neq y) \supset (\mathrm{rmdr}(y, x) \neq 0)$
311	$(\mathrm{prime}(x) \wedge y \neq 1 \wedge x \neq y) \supset (\mathrm{rmdr}(x, y) \neq 0)$
314	$\mathrm{rmdr}(y \times x, y) = 0$
317*a*	$(y = 0) \supset ((y \times x)/y = 0)$
317*b*	$(y \neq 0) \supset ((y \times x)/y = x)$
318	$(x \neq 0 \wedge x \mid w) \supset (y \times w/x = (y \times w)/x)$
321	$(z \neq 0 \wedge z \nmid x) \supset (z \times y \neq x)$
323	$(x \neq 0 \wedge x \times z = y) \supset (z = y/x)$
324	$(x \times y = 1) \equiv (x \neq 0 \wedge y \neq 0 \wedge \mathrm{pr}(x) = 0 \wedge \mathrm{pr}(y) = 0)$
345	$y + z \not< \mathrm{pr}(z)$
368*a*	$((x = 0) \supset (x + (y - \mathrm{pr}(x)) = y))$
368*b*	$((x \neq 0 \wedge y < \mathrm{pr}(x)) \supset (x + (y - \mathrm{pr}(x)) = x))$
368*c*	$((x \neq 0 \wedge y \not< \mathrm{pr}(x)) \supset (x + (y - \mathrm{pr}(x)) = \mathrm{s}(y)))$
372	$(x < y) \supset (\mathrm{pr}(y) \not< x)$
375	$(((y = 0 \wedge x = 0) \supset (z \neq 0)) \wedge ((y \neq 0) \supset (\mathrm{pr}(y) < z))) \equiv$
	$\equiv (\mathrm{pr}(y + x) < x + z)$
397	$(x < y) \supset (x - y = 0)$
398	$(\mathrm{pr}(\mathrm{pr}(x)) < x) \equiv (x \neq 0)$
400	$(\mathrm{pr}(\mathrm{pr}(y)) < z + y) \equiv (z \neq 0 \vee y \neq 0)$
401	$(((x < y) \supset (\mathrm{pr}(z) < x)) \wedge ((x \not< y \wedge z = 0) \supset (x \neq 0)) \wedge$
	$((x \not< y \wedge z \neq 0) \supset (y \not< z))) \equiv (\mathrm{pr}(z + (x - y)) < x)$
402	$x + y \not< \mathrm{pr}(x)$
403	$(x \not< y \wedge x \neq y) \supset (\mathrm{pr}(x) \not< y)$

Table G.10: Arithmetical lemmas proved by *Nqthm* (II)

Lemma	Simplifier		IP	Lemma	Simplifier		IP
	core	LP			core	LP	
37		○		229		○	
100			○	237		○	
102			○	279			○
103			○	280			○
113		○		283			○
120			○	284			○
168		○		292		○	
169		○		293		○	
170		○		299		○	
179		○		300			○
180			○	301	○		
182			○	302	○		
183			○	303	○		
184		○		304	○		
185	○			305			○
186		○		306	○		
187	○			307			○
188		○		308		○	
189			○	309		○	
190		○		310		○	
191	○			311	○		
197		○		314	○		
198			○	317*a*			○
202		○		317*b*			○
208		○		318	○		
209	○			321	○		
210			○	323	○		
212			○	324		○	
213		○		345		○	
215		○		368*a*	○		
216			○	368*b*		○	
217			○	368*c*		○	
218			○	372		○	
219	○			375		○	
220	○			397		○	
221	○			398		○	
222		○		400		○	
225		○		401		○	
226			○	402		○	
227			○	403		○	
228			○				

IP inductive prover
LP linear arithmetic procedure

Table G.11: Arithmetical lemmas - *Nqthm*'s performance

Lemma	*simplify1*/2		*weak_simplify*/1	
	Time (s)	Sublanguage	Time (s)	Sublanguage
37	0.7	$\mathcal{L}_{PrA^\bullet}$	2.1	$\star$
100	0.3	$\mathcal{L}_{PrA^\bullet}$	0.4	$\star$
102	0.5	$\mathcal{L}_{PrA^\bullet}$	0.7	$\star$
103	0.7	$\mathcal{L}_{PrA^\bullet}$	1.0	$\star$
113	0.5	$\mathcal{L}_{PrA^\bullet}$	0.8	$\star$
120	0.7	$\mathcal{L}_{PrA^\bullet}$	1.0	$\star$
168	3.9	$\mathcal{L}_{PrA^\bullet}$	4.5	$\star$
169	0.8	$\mathcal{L}_{PrA^\bullet}$	1.3	$\star$
170	1.2	$\mathcal{L}_{PrA^\bullet}$	1.7	$\star$
179	0.6	$\mathcal{L}_{PrA^\bullet}$	0.9	$\star$
180	1.8	$\mathcal{L}_{PrA^\bullet}$	2.1	$\star$
182	5.6	$\square$	138.5	$\square$
183	0.9	$\mathcal{L}_{PrA^\bullet}$	1.4	$\star$
184	4.3	$\mathcal{L}_{SMA}$	4.6	$\star$
185	3.5	$\square$	478.1	$\square$
186†	0.0	$\mathcal{L}_{PrA^\bullet}$	0.0	$\star$
187†	0.0	$\mathcal{L}_{PrA^\bullet}$	0.0	$\star$
188	3.0	$\square$	14.1	$\mathcal{L}_{PrA^\bullet}$
189	10.7	$\square$	271.8	$\square$
190	6.8	$\square$	146.0	$\square$
191	11.4	$\square$	272.2	$\square$
197	0.5	$\mathcal{L}_{PrA^\bullet}$	0.8	$\star$
198	29.2	$\square$	513.0	$\mathcal{L}_{SMA}$
202	6.1	$\square$	142.3	$\square$
208	0.7	$\mathcal{L}_{PrA^\bullet}$	1.1	$\star$
209	4.1	$\mathcal{L}_{PrA^\bullet}$	4.9	$\star$
210	16.1	$\square$	420.1	$\square$
212	4.7	$\mathcal{L}_{SMA}$	2.9	$\star$
213	0.9	$\mathcal{L}_{PrA^\bullet}$	1.5	$\star$
215	0.8	$\mathcal{L}_{PrA^\bullet}$	1.3	$\star$
216	10.4	$\square$	317.3	$\square$
217	22.0	$\square$	396.3	$\square$
218	22.0	$\square$	392.9	$\square$
219	21.7	$\square$	391.6	$\square$
220	21.5	$\square$	391.5	$\square$
221	0.7	$\mathcal{L}_{PrA^\bullet}$	1.2	$\star$
222	1.2	$\mathcal{L}_{PrA^\bullet}$	1.6	$\star$
225	53.9	$\square$	339.2	$\square$
226	2.3	$\square$	25.8	$\mathcal{L}_{PrA^\bullet}$
227	31.1	$\square$	13,039.4	$\mathcal{L}_{SMA}$

† member of decidable sublanguage

$\star$ same sublanguage as for *simplify1*/2

$\square$ failure

Table G.12: Time Performances – Simplifiers (I)

Lemma	*simplify1/2*		*weak_simplify/1*	
	time (s)	sublanguage	time (s)	sublanguage
228	76.7	□	512.4	$\mathcal{L}_{SMA}$
229	6.5	□	72.0	□
237	8.6	□	149.0	□
279	14.8	□	267.8	□
280	61.7	□	508.5	□
283	11.7	□	146.4	□
284	10.8	□	147.2	□
292	3.5	□	88.3	□
293	0.7	$\mathcal{L}_{PrA^\bullet}$	1.1	⋆
299	75.2	□	669.6	□
300	24.2	□	195.9	□
301	25.6	□	149.9	□
302	13.3	$\mathcal{L}_{SMA}$	8.3	⋆
303	14.5	$\mathcal{L}_{SMA}$	5.9	⋆
304	5.2	$\mathcal{L}_{SMA}$	3.9	⋆
305†	0.0	$\mathcal{L}_{SMA}$	0.0	⋆
306	32.9	□	12,455.4	$\mathcal{L}_{SMA}$
307	4.1	□	16.7	$\mathcal{L}_{SMA}$
308	44.0	□	295.2	□
309	6.1	$\mathcal{L}_{SMA}$	2.7	⋆
310	18.5	□	322.8	□
311	9.2	$\mathcal{L}_{SMA}$	6.3	⋆
314	4.7	$\mathcal{L}_{SMA}$	2.8	⋆
317*a*	1.2	$\mathcal{L}_{PrA^\bullet}$	2.2	⋆
317*b*	9.0	□	24.7	$\mathcal{L}_{PrA^\bullet}$
318	13.7	□	106.4	□
321	2.8	$\mathcal{L}_{SMA}$	1.3	⋆
323	2.5	□	92.4	□
324	4.3	$\mathcal{L}_{PrA^\bullet}$	5.0	⋆
345	0.7	$\mathcal{L}_{PrA^\bullet}$	0.9	⋆
368*a*	3.5	$\mathcal{L}_{PrA^\bullet}$	4.2	⋆
368*b*	5.5	$\mathcal{L}_{PrA^\bullet}$	6.3	⋆
368*c*	5.8	$\mathcal{L}_{PrA^\bullet}$	6.6	⋆
372	1.0	$\mathcal{L}_{PrA^\bullet}$	1.4	⋆
375	4.1	$\mathcal{L}_{PrA^\bullet}$	5.2	⋆
397	0.8	$\mathcal{L}_{PrA^\bullet}$	1.2	⋆
398	2.6	$\mathcal{L}_{PrA^\bullet}$	3.0	⋆
400	2.9	$\mathcal{L}_{PrA^\bullet}$	4.5	⋆
401	12.2	$\mathcal{L}_{PrA^\bullet}$	12.8	⋆
402	0.7	$\mathcal{L}_{PrA^\bullet}$	0.9	⋆
403	1.3	$\mathcal{L}_{PrA^\bullet}$	1.8	⋆

† member of decidable sublanguage

⋆ same sublanguage as for *simplify1/2*

□ failure

Table G.13: Time Performances – Simplifiers (II)

Symbol	Meaning
#	conjunction
\	disjunction
=>	conditional
<=>	biconditional
void	contradiction
:pnat=>	universal quantifier (in $\mathbb{N}$)
:pnat#	existential quantifier (in $\mathbb{N}$)
_ = _ in pnat	equality (in $\mathbb{N}$)
<	less than relation
=<	less than or equal to relation
*	multiplication
^	exponentiation
s	successor
+	sum

Table G.14: Object-level syntax

Depth 4

qtf0401 $(0 \leq 0 \vee 0 \leq 1) \wedge 0 \leq 0 \supset \bot$
qtf0402 $(x_6 \leq 0 \supset \bot \equiv 1 = 1 \wedge 0 \leq x_5)$
qtf0403 $1 + 1 < s0 + x_{10}$
qtf0404 $((1 \leq x_5 \equiv 0 \leq x_5) \supset \bot)$
qtf0405 $(x_3 \leq 1 \wedge (0 \leq x_3 \equiv 0 < 1))$
qtf0406 $((0 \leq x_4 \equiv 1 \leq 1) \supset x_9 \leq x_5 \supset x_2 = x_1)$
qtf0407 $(x_2 < 0 \equiv 1 \leq 0 \supset 1 \leq 1)$
qtf0408 $0 + x_4 = 1 \wedge x_{10} = x_8 \wedge 1 \leq 0$
qtf0409 $((1 \leq 0 \equiv 0 \leq x_5) \supset 0 \leq 1 \supset \bot)$
qtf0410 $1 \times (1 \times 0) \leq 0$

Depth 5

qtf0501 $((1 \leq x_{10} \supset (1 \leq x_{10} \equiv 1 \leq 1)) \vee (0 \leq 0 \supset x_2 < 1) \vee x_{10} = 0 \wedge 1 = 1)$
qtf0502 $(((0 \leq 0 \supset 1 < 1) \vee 0 = x_6 \wedge 1 \leq 1) \supset (x_1 \leq 0 \supset \bot \equiv 0 \leq 1 \supset \bot))$
qtf0503 $((x_2 \leq 1 \vee 0 = 0) \supset \bot \equiv (0 \leq x_9 \supset x_3 \leq 0) \vee 1 \leq 0 \supset \bot)$
qtf0504 $1 \times x_7 \times 1 \leq 0 + 1 \exp 1 \wedge 0 \times 1 = s0 \wedge 1 \leq x_6 \supset x_7 \leq 0$
qtf0505 $((0 \leq 0 \supset 1 \leq 1) \vee x_4 < x_5 \exp 1) \wedge s1 = 0 \times 1 \wedge x_8 \times 1 = 0 + 0$
qtf0506 $(x_3 \exp 0) \exp(1 + 0) \leq 0 \wedge (x_{10} \leq 1 \vee x_{10} < 0) \supset \bot$
qtf0507 $(x_3 \exp 0) \exp(1 + 0) \leq 0 \wedge (x_{10} \leq 1 \vee x_{10} < 0) \supset \bot$
qtf0508 $((1 \leq 0 \vee x_5 \leq 1) \supset \bot \equiv 1 \leq x_6 \supset (1 \leq 1 \equiv 1 \leq 0))$
qtf0509 $((x_2 \leq 0 \supset x_5 \leq x_5) \vee x_9 \leq 1 \supset \bot) \supset \bot$
qtf0510 $((x_2 \leq 0 \supset x_5 \leq x_5) \vee x_9 \leq 1 \supset \bot) \supset \bot$

```
Depth of conjecture. 3
Cjt. 0=0^x10 in pnat
********** Successfully reduced to strict_sum
RewCjt. 0=0 in pnat#x10=0 in pnat=>void
New remove rules.
  [0=v1^v2 in pnat,v1=0 in pnat#v2=0 in pnat=>void,void=>void]
Time = 13650


Depth of conjecture. 4
Cjt. 0=<x1* (1+x4)
********** Successfully reduced to strict_multiplication
RewCjt. ((x1*1=0 in pnat#x1*x4=0 in pnat)=>void)\x1*1=0 in
pnat#x1*x4=0 in pnat
New remove rules.
  [v4* (v5+v3)=0 in pnat,v4*v5=0 in pnat#v4*v3=0 in pnat,void=>void]
  [0=v4* (v5+v3)in pnat,v4*v5=0 in pnat#v4*v3=0 in pnat,void=>void]
Time = 69033


Depth of conjecture. 5
Cjt. s (x8*x7)+1^1*x8=<1
********** Successfully reduced to strict_multiplication
RewCjt. ((x8*x7=0 in pnat#1=0 in pnat)#1*x8=0 in pnat)\ (((x8*x7=1 in
pnat#1=0 in pnat)\x8*x7=0 in pnat#1=1 in pnat)#1*x8=0 in pnat)\
(x8*x7=0 in pnat#1=0 in pnat)#1*x8=1 in pnat
Time = 4250


Depth of conjecture. 6
Cjt. s (x4+1^0)+x4* (0+1)*x7=0 in pnat
********** Successfully reduced to strict_multiplication
RewCjt. ((x4=0 in pnat#1=0 in pnat)#1=0 in pnat)#x4*1*x7=0 in pnat
Time = 2117


Depth of conjecture. 7
Cjt. ((x1^1*1^x10=< (x6^0)^s 0# (0=1 in pnat#1=0 in pnat)\0=0^x10 in
pnat)\x5:pnat=>x3=1^x4 in pnat=>0+x8=<x2)=>void
********** Successfully reduced to strict_sum
RewCjt. (((x1<1\x1=1 in pnat)# (0=1 in pnat#1=0 in pnat)\0=0 in
pnat#x10=0 in pnat=>void)\x5:pnat=>x3=1 in pnat=>x8<x2\x8=x2 in
pnat)=>void
New remove rules.
  [0=v1^v2 in pnat,v1=0 in pnat#v2=0 in pnat=>void,void=>void]
Time = 12233
```

Table G.15: Sample of Randomly generated formulae (I)

```
Depth of conjecture. 8
Cjt. ((x3:pnat=>x10:pnat#1^1=<x9)#x6* (1+x7)=<s (0*1)=>void<=>0^ (1+
(1+0+0))=1 in pnat)=>x8:pnat#s x7^0=1 in pnat
********** Successfully reduced to strict_multiplication
RewCjt. ((x3:pnat=>x10:pnat# ((x9=0 in pnat=>void)#x9=1 in
pnat=>void)\1=x9 in pnat)# ((x6*1=0 in pnat#x6*x7=0 in pnat)\ (x6*1=1
in pnat#x6*x7=0 in pnat)\x6*1=0 in pnat#x6*x7=1 in pnat)=>void<=>0=1
in pnat\1=0 in pnat# (1=0 in pnat#0=0 in pnat)#0=0 in pnat)=>x8:pnat#
((x7=1 in pnat#1=0 in pnat)\x7=0 in pnat#1=1 in pnat)\0=0 in pnat
New remove rules.
  [v4* (v5+v3)<1,v4*v5=0 in pnat#v4*v3=0 in pnat,void=>void]
  [v4* (v5+v3)=1 in pnat,(v4*v5=1 in pnat#v4*v3=0 in pnat)\v4*v5=0 in
   pnat#v4*v3=1 in pnat,void=>void]
Time = 107667


Depth of conjecture. 9
Cjt. x2:pnat=>x7:pnat#0=<x1* (1^ (x4^x6)*1+ (s (x4+1)+0))
********** Successfully reduced to strict_multiplication
RewCjt. x2:pnat=>x7:pnat# ((x1* (1*1)=0 in pnat# (x1*x4=0 in
pnat#x1*1=0 in pnat)#x1*1=0 in pnat)=>void)\x1* (1*1)=0 in pnat#
(x1*x4=0 in pnat#x1*1=0 in pnat)#x1*1=0 in pnat
New remove rules.
  [0=v4* (v5+v3)in pnat,v4*v5=0 in pnat#v4*v3=0 in pnat,void=>void]
  [v4* (v5+v3)=0 in pnat,v4*v5=0 in pnat#v4*v3=0 in pnat,void=>void]
Time = 162967


Depth of conjecture. 10
Cjt. ((x2:pnat=>0=1 in pnat=>void)=>void)\ ((((0=0 in pnat#x3=x1 in
pnat=>1<0<=>x9:pnat=>1=<x4=>void)=> (x9*1<1+x9<=>x1=<x3=>0=<1)=>void)\
(0=0+ (1+x7)in pnat=>x9^1<1*x4)\ (x2<s (0^0)<=>1=x6 in pnat))# (s
0*1=1*x5 in pnat=> (x6:pnat=>0=x6 in pnat=>1=<1)=>void<=>
(x4:pnat#1=<0<=> (0^1+1)^1=1 in pnat))<=>0=<0*0^ (s x8*1^0))
********** Successfully reduced to strict_sum
RewCjt. ((x2:pnat=>0=1 in pnat=>void)=>void)\ ((((0=0 in pnat#x3=x1
in pnat=>1<0<=>x9:pnat=> (1<x4\1=x4 in pnat)=>void)=> (x9<1+x9<=>
(x1<x3\x1=x3 in pnat)=>0<1\0=1 in pnat)=>void)\ (0=0+ (1+x7)in
pnat=>x9<x4)\ (x2<s 1<=>1=x6 in pnat))# (s 0=x5 in pnat=>
(x6:pnat=>0=x6 in pnat=>1<1\1=1 in pnat)=>void<=> (x4:pnat#1<0\1=0 in
pnat<=>0+1=1 in pnat\1=0 in pnat))<=>0<0\0=0 in pnat)
Time = 35900
```

Table G.16: Sample of Randomly generated formulae (II)

Formula Depth	Sample Size	Distribution (%)				
		$\mathcal{L}_{PrA^\bullet}$	$\mathcal{L}_{SMA}$	$\Sigma'_{PrA^\bullet}$	Σ'_{SMA}	UF
3	300	44	2	50	2	1
4	300	33	0	57	5	5
5	300	12	0	73	3	12
6	300	7	0	70	5	18
7	300	1	0	64	3	32
8	300	4	0	59	3	34
9	300	0	0	45	7	48
10	300	0	0	33	0	67

$\Sigma_{PrA^\bullet}$ extension of $\mathcal{L}_{PrA^\bullet}$
Σ_{SMA} extension of $\mathcal{L}_{SMA}$
$\Sigma'_{PrA^\bullet}$ $\Sigma_{PrA^\bullet} - \mathcal{L}_{PrA^\bullet}$
Σ'_{SMA} $\Sigma_{SMA} - \mathcal{L}_{SMA}$
UF undecided formulae

Table G.17: Distribution - Randomly generated formulae

Formula Depth	Sample Size	Subclass					
		$\Sigma_{PrA^\bullet} - \mathcal{L}_{PrA^\bullet}$		$\Sigma_{SMA} - \mathcal{L}_{SMA}$		Undecided Formulae	
		Mean (s)	Deviation (s)	Mean (s)	Deviation (s)	Mean (s)	Deviation (s)
3	300	0.5	1.6	0.3	0.0	78.9	26.6
4	300	1.9	12.1	5.6	18.3	84.9	35.3
5	300	0.9	1.0	14.2	23.1	118.3	132.5
6	300	4.1	12.7	21.2	41.3	118.3	146.0
7	300	7.5	12.5	42.7	42.8	148.0	103.3
8	300	9.1	10.8	53.7	52.8	375.3	458.6
9	300	12.3	18.5	1,441.7	2,071.8	229.0	160.8
10	300	14.9	20.4	—	—	381.2	273.1

$\Sigma_{PrA^\bullet}$ extension of $\mathcal{L}_{PrA^\bullet}$
Σ_{SMA} extension of $\mathcal{L}_{SMA}$

Table G.18: Time Performance - Randomly generated formulae

Depth 6

qtf0601 $(sx_7 + (x_8 + 0) + 0 < (x_2 + 0)\exp(0 + 0) \equiv (x_3 \leq sx_8 \supset$
$\supset 1 \leq 0 \vee 1 \leq 1) \vee s1 \leq 0 \supset (1 \leq 0 \equiv x_7 = 1))$

qtf0602 $(1 \exp 1 \leq s0 \supset (x_3 \leq x_6 \supset 0 = x_{10}) \vee s1 \leq 1 \exp 0 \equiv$
$\equiv 0 \leq (x_5 + 0)\exp(1 + 0) + x_7)$

qtf0603 $(((x_6 \leq 1 \vee x_7 \leq 1) \supset \bot \equiv (x_9 \leq 0 \supset$
$\supset 0 \leq 0) \vee x_7 \leq x_8 \vee 0 \leq 1) \supset s(s(0 + 1)) = 1 \exp 1)$

qtf0604 $0 \times x_3 = 0 \times (0 + 0 \times x_3) \supset 1 \leq 1 \supset x_6 < sx_8 \exp s1$

qtf0605 $((((1 \leq 1 \supset x_1 \leq 1) \vee 0 \leq 1 \supset 0 \leq x_{10}) \vee \supset \bot) \vee$
$\vee((1 \leq 0 \vee 0 \leq x_8) \supset \bot \equiv (0 \leq 1 \supset x_{10} \leq x_3) \vee 0 \leq x_3 \supset 1 \leq 0))$

qtf0606 $(((x_1 \leq s1 \supset 1 \leq x_2 \supset \bot) \vee x_5 \leq 1) \wedge ((0 \leq x_3 \vee 0 \leq 1) \supset \bot \equiv$
$\equiv s0 \times (x_{10} + 0) < s(x_4 \exp 0)))$

qtf0607 $(x_8 \times x_{10} \exp 1 \leq 1 \times (0 \times x_4) \supset 0 = s1) \vee (0 \times x_{10} \leq 0 \supset$
$\supset 0 = 0 \wedge x_6 \leq 0)\ (0 \leq 0 \vee 0 = x_{10}) \supset \bot$

qtf0608 $1 \times ((x_8 + 1 + x_7 \times 0) \times 0) = x_6$

qtf0609 $((1 + (1 + 1) \leq 1 \supset x_2 < s1 \exp 1) \vee ((0 \leq 1 \vee 0 \leq 0) \supset \bot \equiv$
$\equiv (x_2 \leq x_6 \vee 1 \leq 0) \supset \bot))$

qtf0610 $0 \times (x_9 \exp 1 + x_{10}) + 0 < 1 + s(sx_9 + x_4 \exp 0)$

Depth 7

qtf0701 $((((x_{10} \leq 0 \supset 1 \leq x_7) \vee 1 < x_9 + 0) \supset \bot \equiv ((x_1 \leq 0 \supset 0 \leq x_8) \vee$
$\vee 1 \leq 0) \supset \bot) \supset \bot)$

qtf0702 $(((1 \leq 0 \exp 1 \supset 1 \leq x_1 \supset 1 \leq 1) \vee 1 \exp 0 < 1 \exp(x_6 \exp 1)) \supset \bot) \vee$
$\vee 0 \times s(x_7 \times x_9) = 1 \exp(x_3 \exp 0 + sx_3)$

qtf0703 $((s(0 \times 1) \leq 0 \supset 1 + (x_9 + x_6) \leq x_{10}) \vee 0 \leq 1 \times x_7 \times 1 + 0 \times (0 + 0)) \supset \bot$

qtf0704 $0 \leq s(0 \times 1 \exp x_9)\exp((0 \exp(0 + x_2))\exp s(1 \exp 1)) \wedge ((sx_5 \leq 0 \exp 0 \supset$
$\supset x_9 = x_7 \wedge x_3 = x_{10}) \vee 1 = 1 \exp(0 \times x_4)) \supset \bot$

qtf0705 $(((0 \leq sx_2 \times x_2 \exp 0 \supset x_3 = 1 \wedge 0 \leq 1 \vee 1 \leq x_4) \vee ((1 \leq x_4 \vee 1 \leq x_5) \supset \bot \equiv$
$\equiv 0 \exp x_9 \leq 0 \times x_7 \supset 1 = x_3 \wedge x_7 \leq x_5)) \supset 0 \times (0 \times 0) < 1 + x_7)$

qtf0706 $x_5 = (1 \exp(x_2 \times 0 + s1))\exp(1 \times (1 \times x_8 \exp x_2)) + (s0 \exp(x_3 + s0) + x_3)$

qtf0707 $s1 \times 0 < s(0 \exp s((1 \exp 0)\exp s0))$

qtf0708 $x_1 \exp sx_{10} + 1 \exp(x_7 \exp(1 + 0)) + s(1 \exp 1 \times (1 \times x_6) + 0) < x_7$

qtf0709 $(0 = s(0 + x_4 + sx_5) \wedge x_9 \leq x_3 \exp x_1 \times x_7 \supset (0 \leq x_{10} \vee 1 \leq 0) \supset \bot \equiv$
$\equiv (s(sx_7) \leq x_3 \supset 1 \leq 1 \exp 1 \supset x_6 \leq x_{10} \supset 0 \leq 1) \vee ((0 \leq 1 \supset 1 \leq 1) \vee s1 < 1) \supset \bot)$

qtf0710 $((sx_4 = 0 \wedge x_1 \leq (0 + x_4)\exp(x_7 + 0) \supset s(sx_{10}) = 0 \times x_8) \vee (((0 \leq 0 \supset 0 \leq 1) \vee$
$\vee(0 \leq x_{10} \equiv 1 \leq x_1)) \supset \bot \equiv sx_5 + 1 = 1 + x_6 \wedge (1 \leq 1 \supset \bot \equiv 1 \leq x_6 \supset \bot)))$

Depth 8

qtf0801 $((s(s(1 \times x_7)) + s(0 \times (1+1)))\exp(0\exp(1+(1+0+0))))\exp 1 =$
$= ((sx_7 \exp 0)\exp 1)\exp x_7$

qtf0802 $(((1 \times 0\exp 0 \leq 0 \supset (x_1 \leq x_4 \supset \bot \equiv 0 \times 1 = x_6))\vee$
$\vee sx_7 \times x_9 \leq (s0+1)\times 1) \supset \bot \equiv (((1+0)\exp(1+1) \leq x_7 \supset$
$\supset (1 \leq 0 \vee 1 < x_3) \supset \bot) \vee x_8 < 0) \supset \bot)$

qtf0803 $((0 \times (1 \times (1+x_3))\exp(sx_8 + x_4) \leq 0 \supset 1+0 < 0\exp s(s1\exp 1))\vee$
$\vee(1 \leq 1 \times 1\exp 1 \times (x_1+1) \supset sx_4 \leq s(s0))\vee$
$\vee 1 \leq (x_1\exp x_1 + (1+0))\exp(0 \times 1 \times s1) \supset ((x_9 \leq 0 \vee 0 = x_4) \supset \bot \equiv$
$\equiv sx_7 \times sx_{10} \leq sx_5))$

qtf0804 $((((s1 \leq x_7 \supset x_7 \leq x_1 \supset 0 \leq 0) \vee (x_3 \leq 1 \supset \bot \equiv$
$\equiv 1 \leq 0 \supset \bot)) \supset \bot \equiv (1+1\times 1)\exp s(1+0) \leq 1 \supset 1 \leq 0) \supset \bot)$

qtf0805 $0 < s(((0 \times x_9)\exp x_5)\exp 0 + 1) \times 0$

qtf0806 $(0 < 0 \vee (1 \times 0 + (1+0+x_3\exp x_4) \leq s(1 \times 0 \times s0) \supset ((1 \leq x_7 \vee 1 \leq 0) \supset \bot \equiv$
$\equiv (x_9 \leq 1 \vee 0 \leq x_1) \supset \bot)) \vee ((x_4\exp x_6 \leq 1 \supset$
$\supset x_4 = 1 \wedge x_4 < 0) \vee (0 \leq 1 \vee 1 < 0) \supset \bot) \supset \bot)$

qtf0807 $0 \leq x_5\exp 1 \times (1\exp 0 + (1 + x_9\exp x_9)) \times (0 \times (1 \times s1) + 0 + 0)$

qtf0808 $((x_3 = 0\exp 1 \wedge x_{10} \times s(s(0\exp 0)) \leq s(x_2 \times (1 \times (x_3 \times x_2))))\vee$
$\vee((x_2 \times 0 + 1\exp x_2) \times x_4 \leq 0 \supset ((0 \leq 0 \supset 1 \leq x_8)\vee$
$\vee x_3 \leq 0 \vee 0 \leq 0) \supset \bot) \vee (((1 \leq 0 \supset 0 \leq 1) \vee 0 \leq 1 \supset \bot) \supset \bot \equiv$
$\equiv ((1 \leq 1 \vee 0 \leq 0) \supset \bot \equiv 1 \times x_2 < s1 \times 0\exp x_4)))$

qtf0809 $s(1 \times 0) + 0 = 1 + s(s((0 \times x_7)\exp 0)) + x_4$

qtf0810 $((x_5 = 0 \wedge (0 \leq 1 \supset x_1\exp 0 + 0 < 1) \vee (x_3 \leq x_1 + 1 \supset 1 = 1 \wedge 1 \leq 0)\vee$
$\vee sx_{10} \leq 1) \vee ((x_7 \leq 0 \supset (1 \leq 1 \supset \bot \equiv x_9 \leq x_1 \vee x_3 = x_3))\vee$
$\vee 1 \leq 0 \supset 0 \times (0+1) < sx_7 \times 1) \supset \bot)$

Depth 9

qtf0901 $x_6\exp 1 \leq 1 + x_9 \times 0 \times (1+0) \wedge x_9 + s((1\exp 0)\exp(x_6 \times x_3)) + s(s(s1)) \leq sx_4 \supset$
$\supset s0 = s(x_1 \times s(0 \times 1)\exp x_{10})$

qtf0902 $((((1 \leq (x_8 \times 1)\exp(x_5\exp 0) \supset (1 \leq 1 \supset 1 \leq 1)\vee$
$\vee x_8 < x_8) \vee s(x_8 \times x_7) + 1\exp 1 \times x_8 \leq 1) \supset \bot \equiv ((x_9 \leq 0\exp(x_6+0) \supset$
$\supset x_5 = 0 + 1 \wedge 0 \times 0 \leq x_9) \vee 0 \leq 1\exp 0 + s0 \supset (x_6 \leq 0 \supset \bot \equiv 1 = 1 \wedge 0 \leq x_5)) \supset \bot) \supset \bot)$

qtf0903 $((((1 \leq sx_5 \supset (0 \leq x_5 \vee x_3 \leq 1) \supset \bot) \vee ((0 \leq x_3 \vee 0 < 1) \supset \bot \equiv$
$\equiv (0+x_4) \times (0+1) < s1 \times (x_9 + x_5))) \supset \bot \equiv sx_2 = sx_1 \wedge x_2 \times 0 = x_1\wedge$
$\wedge 1 = 0 \times s0 \wedge 1 \times 1 = x_4 \wedge 0 + x_4 = 1) \supset \bot)$

qtf0904 $s(x_{10} + x_8 + (0+1)\exp s0) + 0\exp 0 + 1 = 0 + 1 + x_7 \supset \bot$

qtf0905 $(s0 = x_1 \supset 0 \leq x_5 \times 1 \supset (0 + s(s1) \leq (1 \times 1)\exp 0 + 0 \supset$
$\supset x_6 + s0 \leq 1 \times 1\exp x_{10} \supset (1 \leq x_{10} \supset \bot \equiv 1 \leq 1 \vee 0 \leq 0))\vee$
$\vee(x_2\exp 1)\exp x_1 < x_{10} + (0 + s1)\exp 1)$

qtf0906 $(((0 \leq 0 \supset s1 < 1 \times s(0 \times x_6) + (s1+1)\exp s(1\exp 0)) \vee 0 + x_6 \times 1 = 0 \times 1\wedge$
$\wedge(((x_2 \leq 1 \supset 0 = 0) \vee 0 \leq x_9 \vee x_3 \leq 0) \supset \bot \equiv ((1 \leq 0 \supset 1 \leq 0)\vee$
$\vee x_1\exp x_5 < 1) \supset \bot)) \supset x_7 = x_5)$

qtf0907 $((s1 + x_7 \times 1) \times (0 + s1\exp(1+1)) \times x_2 \leq (0 + s((1\exp 0)\exp(x_2 \times x_6)) + sx_7) \times (0 \times 0)\wedge$
$\wedge(0 \leq s0 \supset (((1 \leq 1 \supset x_4 < x_5) \vee 0 \leq 1 \supset \bot) \supset \bot \equiv$
$\wedge(1 \leq x_8\exp 1 \supset (0 \leq x_4 \equiv x_3 \leq 0)) \vee (0 \leq 0 \vee x_{10} \leq 1) \supset \bot)) \vee x_{10} < 0)$

qtf0908 $0 \times ((x_2 + s(x_2\exp 1)) \times (s(0+1) + (x_5 + (x_{10} + x_4)))) + x_6 + 0 < x_2 \times (0 \times x_6)$

qtf0909 $(x_5 \leq ((1\exp(1\exp x_6 \times (1 \times x_6)))\exp(1\exp s1 + s1))\exp((1 \times 0)\exp(0 \times (sx_2 + 0 \times 1))) \supset$
$\supset x_5 = s(x_6 + x_9 \times 1 + x_5) \wedge x_7 = 1\exp 0 \times 1 \wedge x_7 + x_8 + 0 + 0 = (x_2 + 0)\exp(0+0) \equiv$
$\equiv (x_3 \leq sx_8 \supset (1 + s0 \leq x_3 \supset 1 \times 1 \leq sx_2 \supset 0 = x_7\exp 1)\vee$
$\vee 0 \leq x_7\exp 1) \vee (1\exp 1 \leq s0 \supset (sx_3 \leq x_6 \times 0 \supset 1 \leq 1 \vee 0 \leq x_5) \vee 0 + (1+0) < x_7) \vee 1 < 0)$

qtf0910 $((x_7 \leq s(1 \times 1) \supset x_9 \leq 0 \supset (((0 \leq 0 \supset x_7 \leq x_8) \vee s0 \leq 1\exp 0) \supset \bot \equiv$
$\equiv s(0 \times 1 + 1\exp 1) < 0 \times x_3 \times (0 \times 0) \times (0 \times x_3))) \vee 1 \leq 1 \supset x_6 < sx_8\exp s1)$

Depth 10

qtf1001 $((((1 \leq s(s1 + s(x_8 \exp x_1)) \supset ((0 \leq s1 \supset (0 \leq x_{10} \equiv$
$\equiv 1 \leq 0)) \vee (0 \leq x_8 \supset 0 \leq 1) \vee x_{10} \leq x_3 \supset \bot) \supset \bot) \vee$
$\vee 0 \leq x_3 \exp((s1 + s0) \times (0 \times (x_1 \times 1))) \supset ((s1 \leq x_2 + 1 \supset$
$\supset 1 = 1 \wedge 0 \leq x_3) \vee (0 \leq 1 \vee 0 < x_{10} \supset \bot) \supset \bot) \supset \bot) \vee$
$\vee(((sx_4 \leq 0 \times (sx_8 + x_{10} \exp 1) \supset s0 \times x_4 = 0 \wedge s1 < s1) \vee$
$\vee s(0 \exp x_{10}) \times 0 \times s1 \leq 0 \exp s((0 + 0) \times s0)) \supset \bot \equiv 0 = 0 \wedge$
$\wedge((0 \times 0 + 1 \leq (0 \times x_{10}) \exp s1 \supset (0 \leq 1 \vee x_7 \leq 0) \supset \bot) \vee x_6 < x_8) \supset \bot))$

qtf1002 $(1 + (1 + 1) \exp x_1 + x_1 + s(s1 \times 1) = s(1 + x_4 + 0) \wedge 0 < 1 \times 0 \equiv$
$\equiv 0 \leq s0 \exp s(1 \times (s(x_2 \exp x_6 + x_2) \times (0 \exp 1 \times 0))))$

qtf1003 $0 \times (s(x_9 + 1) \exp x_{10} + 0) + (1 + s(s(s(x_9 \times x_4) + (0 + x_8) \times 0 \exp x_2) + x_{10})) <$
$< 0 \exp((x_{10} + (1 + (sx_7 \exp 1) \exp 1 \times (x_9 \exp s0 + (x_1 \times 0) \exp 1))) \exp$
$\exp(s((0 \exp(x_8 \times 1) + 0) \exp(0 + (1 \exp 1) \exp(0 \exp 1))) \exp 0))$

qtf1004 $(x_1 = 0 \wedge (1 \leq 1 \supset 1 \exp 0 < 1 \exp(((x_6 \times 1) \exp s0) \exp s(x_7 \times x_9))) \vee$
$\vee 1 < x_3 \exp 0 + sx_3) \vee ((s((0 \exp 1) \exp 0 \times s(1 + x_9)) \leq s(s(sx_6)) + x_{10} \supset$
$\supset 0 \leq 1 \times x_7 \times 1 \times (0 \times (0 + 0)) + s0) \vee s(0 \times (1 \exp x_9) \exp(0 \exp 0)) = x_2 \wedge$
$\wedge(1 \times 1) \exp 1 \leq 0 \exp s(s(x_5 + 0))) \supset \bot$

qtf1005 $x_2 \exp(x_9 + x_7) \leq 0 \exp s(x_3 + x_{10} \times (1 \times 1 \exp(s0 \times x_4 \exp 0)))$

qtf1006 $x_{10} < s(x_2 \exp((x_2 \times s(0 \times x_1)) \exp x_3) \times 1) \times (0 + (s0 + 1))$

qtf1007 $((((s((0 + x_7) \times (1 + x_3)) \leq 0 \supset x_7 + x_5 + 0 \exp 0 + 0 \leq 1 + x_7) \vee$
$\vee((1 + x_2 \leq s0 \supset 1 \leq 1 \supset 0 \leq x_3) \vee x_3 = 1 \exp 0 \wedge 0 \exp 1 = 0 \exp 0) \supset \bot) \supset \bot \equiv$
$\equiv x_1 \exp sx_{10} + 1 \exp(x_7 \exp(s1 + (0 + 1))) + (1 + 1 \times x_6) < 0) \wedge 0 = s0 \exp 0 \wedge x_5 \leq 0)$

qtf1008 $((((x_1 \leq x_7 \supset ((0 \leq x_{10} \supset 1 \leq 0) \vee x_7 \leq x_3 \vee 1 \leq 1) \supset \bot) \vee$
$\vee(x_6 \times (x_{10} \times 0) \leq 1 \supset (0 \leq 1 \vee 1 \leq 1) \supset \bot) \vee s(s1) < 1) \supset \bot \equiv$
$\equiv x_9 \leq sx_4) \wedge (x_1 \leq ((0 \times x_4) \exp(x_7 + 0) \times x_8 + s(x_{10} \times (0 \times x_8))) \exp 0 \supset$
$\supset ((0 \times (0 \times 0) \leq x_2 \supset 1 = s(0 \exp x_5)) \vee x_{10} \exp s1 = x_1 + 1 + x_5 \times 1 \wedge$
$\wedge 1 \leq x_6) \supset \bot) \vee ((((1 \times 1) \exp x_9 \leq sx_8 \supset x_6 \times (1 + x_7) = s(0 \times 1)) \vee$
$\wedge 0 \exp(1 + (1 + 0)) \leq 0) \supset \bot \equiv ((sx_7 \exp 0) \exp 1) \exp x_7 = 1 \exp(1 \times (1 + 0))))$

qtf1009 $(((1 \times 0 \exp((0 + 0) \exp(0 + x_4)) \leq s(x_1 \exp(x_4 \times 0)) \exp 1 \supset$
$\supset (sx_7 \times x_9 \leq (0 + 1) \times 1 \supset (1 \leq 0 \vee 1 \leq x_7) \supset \bot) \vee$
$\vee((1 \leq 0 \supset 1 < x_3) \vee x_8 < 0) \supset \bot) \vee 0 \leq x_4 \exp 0 + 1 \supset$
$\supset 0 \exp(1 \exp 1 \times (0 \times x_1)) + 1 \leq x_5 \times (0 \times (1 \times 1) \exp x_3)) \supset \bot \equiv$
$\equiv s(x_8 \times x_4) < 0)$

qtf1010 $s(s1 \exp 1) \leq 0 \exp 1 + 1 \times 1 \exp(s(s1) \exp(x_1 + 1)) \times x_4 \supset$
$\supset s(s0) \times (1 \times (x_1 \exp x_1 + (s1 + 0 \times 0)) \exp 1 + s(s(s1))) =$
$= s(1 \times s(((x_9 \times 0) \exp 0) \exp 1)))$

The performance of *Nqthm* for this sample is described in table G.19. The results for the simplifiers follow in table G.20.

Nqthm

Formula Depth	Sample Size	Success rate (%)		Role of LP (%)	
		Original sample	Extended sample	Original sample	Extended sample
3	10	40	40	20	20
4	10	30	50	10	20
5	10	10	80	0	40
6	10	20	30	20	20
7	10	20	50	10	20
8	10	20	40	10	20
9	10	20	20	0	0
10	10	10	40	0	10
Average results		21.3	43.8	8.8	18.8

Nqthm with arithmetical lemmas

Formula Depth	Sample Size	Success rate (%)		Role of LP (%)	
		Original sample	Extended sample	Original sample	Extended sample
3	10	40	40	20	20
4	10	30	50	10	20
5	10	10	80	0	40
6	10	20	30	20	20
7	10	20	60	10	20
8	10	30	50	10	10
9	10	20	30	10	10
10	10	10	40	0	10
Average results		22.5	47.5	10.0	18.8

LP linear arithmetic procedure

Table G.19: Success Rates - Randomly generated quantifier-free formulae (I)

Weak_simplify/1

Formula Depth	Sample Size	Distribution (%)				
		$\mathcal{L}_{PrA^*}$	$\mathcal{L}_{SMA}$	Σ'_{PrA^*}	Σ'_{SMA}	UF
3	10	0	0	100	0	0
4	10	10	0	90	0	0
5	10	0	0	100	0	0
6	10	0	0	70	0	30
7	10	0	0	50	20	30
8	10	0	0	40	0	60
9	10	0	0	30	0	70
10	10	0	0	10	0	90

Simplify/1

Formula Depth	Sample Size	Distribution (%)				
		$\mathcal{L}_{PrA^*}$	$\mathcal{L}_{SMA}$	Σ'_{PrA^*}	Σ'_{SMA}	UF
3	10	0	0	100	0	0
4	10	10	0	90	0	0
5	10	0	0	100	0	0
6	10	0	0	70	0	30
7	10	0	0	60	10	30
8	10	0	0	40	0	60
9	10	0	0	30	0	70
10	10	0	0	10	0	90

Σ'_{PrA^*}	$\Sigma_{PrA^*} - \mathcal{L}_{PrA^*}$
Σ'_{SMA}	$\Sigma_{SMA} - \mathcal{L}_{SMA}$
Σ_{PrA^*}	extension of $\mathcal{L}_{PrA^*}$
Σ_{SMA}	extension of $\mathcal{L}_{SMA}$
UF	undecided formulae

Table G.20: Success Rates - Randomly generated quantifier-free formulae (II)

Appendix H

Arithmetical *Remove* Rules

The arithmetical rule base $\mathcal{R}_{PA^*}$ contains *remove* rules for all the symbols of PA^* that are deviant with respect to $\mathcal{L}_{PrA^*}$ and/or $\mathcal{L}_{SMA}$. Table H.1 and H.2 respectively have the total rules and the rule schemes, all of which derived from the definitions of the corresponding symbols. Partial rules are distributed amongst three tables, H.3 (quantifier-free rules), H.4 (rules applicable to terms) and H.5 (quantifier-introducing rules). A last table, H.6, exhibits the elements of the arithmetical equality base $\mathcal{E}_{PA^*}$, which provides equations for the elimination of disagreements.

Symbol	Rules		
$>$	$v_1 > v_2$	$\Rightarrow$	$v_2 < v_1$
$\leq$	$v_1 \leq v_2$	$\Rightarrow$	$(v_1 < v_2) \vee (v_1 = v_2)$
$\geq$	$v_1 \geq v_2$	$\Rightarrow$	$(v_2 < v_1) \vee (v_1 = v_2)$
even	$\text{even}(v)$	$\Rightarrow$	$(\exists u)(v = u + u)$
prime	$\text{prime}(v_1)$	$\Rightarrow$	$(v_2)(v_2 \mid v_1 \supset (v_2 = 1 \vee v_2 = v_1))$
prime_1	$\text{prime}_1(v_1, v_2)$	$\Rightarrow$	$v_2 \neq 0 \wedge (v_3)((v_3 \mid v_1 \wedge 1 < v_3) \supset (v_2 < v_3))$
s	$\text{s}(v)$	$\Rightarrow$	$v + 1$
double	$\text{double}(v)$	$\Rightarrow$	$v + v$
Sum_n	$\text{Sum}_n(v_1, \ldots, v_n) = v_{n+1}$	$\Rightarrow$	$v_{n+1} = v_1 + \cdots + v_n$
$\mid$	$v_1 \mid v_2$	$\Rightarrow$	$v_1 \neq 0 \wedge (\exists v_3)(v_1 \times v_3 = v_2)$
pr	$\text{pr}(v)$	$\Rightarrow$	$v - 1$

Intermediate Sublanguage

$$\mathcal{L}' = \{0, 1, +, \times, \exp, -, \text{half}, /, \text{gfc}, \text{rmdr}, \gcd, \min_n, \max_n, <\}$$

Table H.1: Total *remove* rules

Symbol	Rules		
$-$	$\phi[v_1 - v_2]$	$\Rightarrow$	$(\exists v_3)(((v_2 < v_1 \wedge v_1 = v_3 + v_2) \vee (v_2 \not< v_1 \wedge v_3 = 0)) \wedge$
	$\wedge\ \phi[v_3/(v_1 - v_2)])$		
half	$\phi[\text{half}(v_1)]$	$\Rightarrow$	$(\exists v_2)((v_1 = v_2 + v_2 \vee v_1 = v_2 + v_2 + 1) \wedge$
	$\wedge\ \phi[v_2/\text{half}(v_1)])$		

Table H.2: *Remove* rules schemes

Symbol	Rules		
$<$	$0 < 0$	$\Rightarrow$	$\bot$
$<$	$0 < v$	$\Rightarrow$	$v \neq 0$
$<$	$1 < v$	$\Rightarrow$	$(v \neq 0) \wedge (v \neq 1)$
$+$	$v_1 + v_2 < 0$	$\Rightarrow$	$\bot$
$+$	$v_1 + v_2 = 0$	$\Rightarrow$	$(v_1 = 0 \wedge v_2 = 0)$
$+$	$v_1 + v_2 < 1$	$\Rightarrow$	$(v_1 = 0 \wedge v_2 = 0)$
$+$	$v_1 + v_2 = 1$	$\Rightarrow$	$(v_1 = 1 \wedge v_2 = 0) \vee (v_1 = 0 \wedge v_2 = 1)$
$+$	$v_1 + v_2 = v_1$	$\Rightarrow$	$v_2 = 0$
$+$	$v_1 + v_2 = v_1 + v_3$	$\Rightarrow$	$v_2 = v_3$
$+$	$v_1 < 1 + v_2$	$\Rightarrow$	$v_1 = v_2 \vee v_1 < v_2$
$\times$	$v_1 \times v_2 = 0$	$\Rightarrow$	$(v_1 = 0 \vee v_2 = 0)$
$\times$	$v_1 \times v_2 = 1$	$\Rightarrow$	$(v_1 = 1 \wedge v_2 = 1)$
$\times$	$v_1 \times v_2 = v_1$	$\Rightarrow$	$v_1 = 0 \vee v_2 = 1$
$\times$	$v_1 \times v_2 = v_1 \times v_3$	$\Rightarrow$	$v_1 = 0 \vee v_2 = v_3$
$\times$	$v_1 \times v_2 = v_3 \times v_2$	$\Rightarrow$	$v_2 = 0 \vee v_1 = v_3$
$\times$	$v_1 \times v_2 < v_1$	$\Rightarrow$	$v_1 \neq 0 \wedge v_2 = 0$
$\times$	$v_1 \times v_2 < v_3 \times v_2$	$\Rightarrow$	$v_2 \neq 0 \wedge v_1 < v_3$
$\times$	$v_1 \times v_2 + 1 = v_3 \times v_2$	$\Rightarrow$	$v_2 = 1 \wedge v_1 + 1 = v_3$
$\times$	$v_1 < v_2 \times v_1 + v_3$	$\Rightarrow$	$(v_1 = 0 \wedge v_3 \neq 0) \vee$
			$\vee (v_1 \neq 0 \wedge (v_1 < v_3 \vee (1 < v_2 + v_3 \wedge v_2 \neq 0)))$
$\times$	$v_1 < v_2 \times (v_1 + v_3)$	$\Rightarrow$	$(v_1 < v_2 \wedge v_1 + v_3 \neq 0) \vee$
			$\vee (v_1 = v_2 \wedge v_1 \neq 0 \wedge 1 < v_1 + v_3) \vee$
			$\vee (v_2 < v_1 \wedge (v_2 = 1 \wedge (v_3 \neq 0 \vee 1 < v_2)))$
exp	$v_1^{v_2} = 0$	$\Rightarrow$	$v_1 = 0 \wedge v_2 \neq 0$
exp	$v_1^{v_2} = 1$	$\Rightarrow$	$v_1 = 1 \vee v_2 = 0$
exp	$v_1^{v_2} = v_1$	$\Rightarrow$	$v_1 = 1 \vee v_2 = 1 \vee (v_1 = 0 \wedge v_2 \neq 0 \wedge v_2 \neq 1)$
exp	$v_1^{v_2} = v_2$	$\Rightarrow$	$v_1 = 1 \wedge v_2 = 1$
exp	$v_1^{v_2} < v_1$	$\Rightarrow$	$v_2 = 0 \wedge v_1 \neq 0 \wedge v_1 \neq 1$
exp	$v_1 < v_1^{v_2}$	$\Rightarrow$	$(v_1 = 0 \wedge v_2 = 0) \vee (v_1 \neq 0 \wedge v_1 \neq 1 \wedge v_2 \neq 0 \wedge v_2 \neq 1)$
exp	$v_1^{v_2} < v_2$	$\Rightarrow$	$(v_1 = 0 \wedge v_2 \neq 0) \vee (v_1 = 1 \wedge v_2 \neq 0 \wedge v_2 \neq 1)$
exp	$v_2 < v_1^{v_2}$	$\Rightarrow$	$v_2 = 0 \vee (v_1 \neq 0 \wedge v_1 \neq 1)$
$-$	$v_1 - v_2 = v_3$	$\Rightarrow$	$(v_2 < v_1 \wedge v_1 = v_2 + v_3) \vee (v_2 \not< v_1 \wedge v_3 = 0)$
$-$	$v_1 - v_2 < v_3$	$\Rightarrow$	$(v_2 < v_1 \wedge v_1 < v_2 + v_3) \vee (v_2 \not< v_1 \wedge v_3 \neq 0)$
$-$	$v_1 < v_2 - v_3$	$\Rightarrow$	$v_3 < v_2 \wedge v_1 + v_3 < v_2$
$-$	$v_1 + (v_2 - v_3) < v_4$	$\Rightarrow$	$(v_3 < v_2 \wedge v_1 + v_2 < v_3 + v_4) \vee (v_3 \not< v_2 \wedge v_1 < v_4)$
$-$	$v_1 < v_2 + (v_3 - v_4)$	$\Rightarrow$	$(v_4 < v_3 \wedge v_1 + v_4 < v_2 + v_3) \vee (v_4 \not< v_3 \wedge v_1 < v_2)$
$-$	$v_1 + (v_2 - v_3) = v_4$	$\Rightarrow$	$(v_3 < v_2 \wedge v_1 + v_2 = v_3 + v_4) \vee (v_3 \not< v_2 \wedge v_1 = v_4)$
$-$	$v_1 = v_2 + (v_3 - v_4)$	$\Rightarrow$	$(v_4 < v_3 \wedge v_1 + v_4 = v_2 + v_3) \vee (v_4 \not< v_3 \wedge v_1 = v_2)$
$-$	$v_1 = (v_2 - v_3) + v_4$	$\Rightarrow$	$(v_3 < v_2 \wedge v_1 + v_3 = v_2 + v_4) \vee (v_3 \not< v_2 \wedge v_1 = v_4)$
$/$	$v_1 / v_2 = 0$	$\Rightarrow$	$v_2 = 0 \vee v_1 < v_2$
half	$\mathrm{half}(v_1) + \mathrm{half}(v_1) = v_2$	$\Rightarrow$	$(v_2 = v_1 + v_1) \vee (v_2 = \mathrm{s}(v_1) + \mathrm{s}(v_1))$

Table H.3: Partial quantifier-free *remove* rules for atoms

Symbol	Rules		
s	$s(0)$	$\Rightarrow$	1
$+$	$v+0$	$\Rightarrow$	v
$+$	$0+v$	$\Rightarrow$	v
$\times$	$v \times 0$	$\Rightarrow$	0
$\times$	$1 \times v$	$\Rightarrow$	v
$\times$	$v \times 1$	$\Rightarrow$	v
$\times$	$0 \times v$	$\Rightarrow$	0
$\times$	$v \times 0$	$\Rightarrow$	0
exp	1^v	$\Rightarrow$	1
exp	v^0	$\Rightarrow$	1
exp	v^1	$\Rightarrow$	v
$-$	$v - v$	$\Rightarrow$	0

Table H.4: Partial *remove* rules for terms

Symbol	Rules		
$\min_n$	$\min_n(v_1, \ldots, v_n) = v_{n+1}$	$\Rightarrow$	$\bigwedge_{i=1}^{n}(v_i \not< v_{n+1}) \wedge \bigvee_{j=1}^{n}(v_{n+1} = v_j)$
$\max_n$	$\max_n(v_1, \ldots, v_n) = v_{n+1}$	$\Rightarrow$	$\bigwedge_{i=1}^{n}(v_i \not< v_{n+1}) \wedge \bigvee_{j=1}^{n}(v_{n+1} = v_j)$
/	$v_1/v_2 = v_3$	$\Rightarrow$	$((v_2 = 0 \vee v_1 < v_2) \wedge v_3 = 0) \vee (v_2 \neq 0 \wedge v_1 \not< v_2 \wedge$
			$\wedge(\exists v_4)(v_4 < v_2 \wedge v_3 \times v_2 + v_4 = v_1))$
/	$v_1/v_2 < v_3$	$\Rightarrow$	$((v_2 = 0 \vee v_1 < v_2) \wedge 0 < v_3) \vee (v_2 \neq 0 \wedge v_1 \not< v_2 \wedge$
			$\wedge(\exists v_4)(\exists v_5)(v_4 < v_2 \wedge v_5 \times v_2 + v_4 = v_1 \wedge v_5 < v_3))$
gfc	$\text{gfc}(v_1, v_2) = v_3$	$\Rightarrow$	$((1 \not< v_1 \vee \text{prime}(v_1)) \wedge v_3 = v_1) \vee$
			$\vee(v_3 \mid v_1 \wedge (v_4)((v_4 \mid v_1 \wedge v_2 \not< v_4) \supset v_3 \not< v_4))$
gfc	$\text{gfc}(v_1, v_2) < v_3$	$\Rightarrow$	$(\exists v_4)(v_4 < v_3 \wedge (1 \not< v_1 \vee \text{prime}(v_1)) \wedge v_4 = v_1) \vee$
			$\vee(v_4 \mid v_1 \wedge (v_5)((v_5 \mid v_1 \wedge v_2 \not< v_5) \supset v_4 \not< v_5))$
half	$\text{half}(v) = u$	$\Rightarrow$	$v = u + u \vee v = u + u + 1$
half	$\text{half}(v) < u$	$\Rightarrow$	$v < u + u$
rmdr	$\text{rmdr}(v_1, v_2) = v_3$	$\Rightarrow$	$(v_2 = 0 \wedge v_3 = v_1) \vee$
			$\vee(v_2 \neq 0 \wedge (\exists v_4)(v_1 = v_4 \times v_2 + v_3 \wedge v_3 < v_2))$
rmdr	$\text{rmdr}(v_1, v_2) < v_3$	$\Rightarrow$	$(v_2 = 0 \wedge v_1 < v_3) \vee (v_2 \neq 0 \wedge$
			$\wedge(\exists v_4)(\exists v_5)(v_1 = v_4 \times v_2 + v_5 \wedge v_5 < v_2 \wedge v_5 < v_3))$
gcd	$\text{gcd}(v_1, v_2) = v_3$	$\Rightarrow$	$v_3 \mid v_1 \wedge v_3 \mid v_2 \wedge (v_4)((v_4 \mid v_1 \wedge v_4 \mid v_2) \supset v_4 \mid v_3)$
gcd	$\text{gcd}(v_1, v_2) < v_3$	$\Rightarrow$	$(\exists v_4)(v_4 < v_3 \wedge v_4 \mid v_1 \wedge v_4 \mid v_2 \wedge (v_5)((v_5 \mid v_1 \wedge v_5 \mid v_2) \supset v_5 \mid v_4))$

Table H.5: Partial quantified *remove* rules for atoms

Equation			Condition
$v_1 + v_2$	$=$	$v_2 + v_1$	–
$(v_1 + v_2) + v_3$	$=$	$v_1 + (v_2 + v_3)$	–
$v_1 \times (v_2 \times v_3)$	$=$	$(v_1 \times v_2) \times v_3$	–
$v_1 \times v_2$	$=$	$v_2 \times v_1$	–
$v \times 1$	$=$	v	–
v^0	$=$	1	–
v^1	$=$	v	–
$v_1 \times (v_2 + v_3)$	$=$	$(v_1 \times v_2) + (v_1 \times v_3)$	–
$(v_1 + v_2) \times v_3$	$=$	$(v_1 \times v_3) + (v_2 \times v_3)$	–
$v_1 \times (v_2 - v_3)$	$=$	$(v_1 \times v_2) - (v_1 \times v_3)$	–
$(v_1 - v_2) \times v_3$	$=$	$(v_1 \times v_3) - (v_2 \times v_3)$	–
$v_1 \times \gcd(v_2, v_3)$	$=$	$\gcd(v_1 \times v_2, v_1 \times v_3)$	–
$\gcd(v_1, v_2) \times v_3$	$=$	$\gcd(v_1 \times v_3, v_2 \times v_3)$	–
$(v_1 \times v_2)^{v_3}$	$=$	$v_1^{v_3} \times v_2^{v_3}$	–
$v_1 = v_2$	$\equiv$	$v_2 = v_1$	–
$v_1 + v_2 = v_1$	$\equiv$	$v_2 = 0$	–
$v_1 + v_2 = v_1 + v_3$	$\equiv$	$v_2 = v_3$	–
$v_1 \times v_2 = v_1$	$\equiv$	$v_2 = 1$	$v_1 \neq 0$
$v_1 \times v_2 < v_1$	$\equiv$	$v_2 < 1$	$v_1 \neq 0$
$v_1 \times v_2 = v_1 \times v_3$	$\equiv$	$v_2 = v_3$	$v_1 \neq 0$
$v_1 \times v_2 = v_3 \times v_2$	$\equiv$	$v_1 = v_3$	$v_2 \neq 0$

Table H.6: Elimination equations