Imperial College of Science, Technology and Medicine
(University of London)
Department of Computing

A Generic Logic Environment

by

William Mark Grant Dawson

A thesis presented to the University of London
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy (Ph.D.)
and the
Diploma of Imperial College (D.I.C.)
in
Computing Science

Abstract

The thesis presents a framework for the development and manipulation of a wide variety of logical systems. The approach is original in its aim of providing tools to assist the construction of logics, and the framework has been realised on a computer workstation as a logic development environment. Both the framework and the design and implementation of the prototype environment are described.

The definition of a logical system is taken to have two components: the language from which the syntactic elements—including judgements—of the system are constructed; and a deductive part, in which the judgements can be interpreted as relations, expressed using inference rules. A method is given for transforming different syntactic presentations of logical systems into the all-introduction style required. This method is applied to Axiomatic and Natural Deduction presentations of several systems, including intuitionistic, relevance, linear, classical, modal and many-valued logics. It is shown that the framework is sufficiently expressive to capture presentations of these systems.

A number of tools are provided for the analysis of systems presented within the environment. One of these checks whether a judgement holds within a given system, by constructing a derivation using inference rules. As it is intended that the environment be used during the development of logical systems, a simple strategy language is provided to select between inference rules.

A second tool allows a user of the environment to conduct case studies which reveal relationships between connectives. In particular, it is shown how the distinct modalities of some classical and intuitionistic modal systems can be found, and their inter-relationships discovered.

A third, and more fundamental, type of analysis is also discussed. When a new inference rule is added to a system, it may not allow anything new to be proved. When this is the case, the new rule is said to be derived with respect to the other rules. If a property of a judgement is expressed as an inference rule, and this rule is found to be derived with respect to the other—more computationally useful—rules, then the property was previously implicitly present. Such knowledge can be used to give results about the presentation of the system as a whole, for example whether the system is consistent.

To my parents

Tom and Jane

Contribution of the Thesis

The thesis makes several contributions to the study of proof theoretic presentations of logical systems made in the spirit of Gentzen's sequent calculi.

A new, simple, but expressive framework is presented that supports the presentations of such systems in a natural way. This is done in Chapter 1, with aspects concerned with quantification treated in Chapter 4. The framework can also be readily implemented on a computer as is shown in Chapter 7.

The next aspect of our contribution is a systematic study which shows how a variety of other, proof theoretic, presentations of logical systems can be transformed into a form suitable for the framework. This forms Chapter 2 and the first part of Chapter 3. Of particular interest is the way in which our approach reveals the nature and manipulation of assumptions in a system, and thereby which structural properties are required in the framework. The methodology is applied Classical and Intuitionistic logic in Chapter 2, where Linear and Relevant systems are also briefly discussed. The analysis is extended in Chapter 3 to a number of modal systems. A way of Gentzenising many-valued systems is also shown, as well as examples that illustrate how the framework treats hypersequents and defeasible systems.

The final aspect of our contribution is an environment implemented on a computer workstation which supports the framework. The environment provides its users with access to a number of activities concerned with the manipulation and analysis of systems. The environment's interface has novel aspects that are described in Chapter 6. The environment supports a user's preferred syntactic presentation directly, and derivations made in a system are shown graphically as proof trees.

One of the activities that is supported by the environment is the construction of a derivation in a system. The environment translates the user's representation of a system into an efficient internal form that is used to construct such derivations. Chapter 7 gives details of this translation.

The activity of making many such derivations in a structured way is illustrated in the second part of Chapter 3 for the case of Modal systems, where the relationship between strings of modalities is often of interest. We illustrate this analysis by showing how the environment was used to discover modalities in Modal systems based on Classical and Intuitionistic logics.

Finally, Chapter 5 presents a novel discussion on the role of derived rules in a system presented using the framework. The argument made there yields a systematic search for schematic proofs that can replace derived rules. Examples are given of the application of this approach.

Acknowledgements

First and foremost I would like to extend my heartfelt thanks to my supervisor Krysia Broda. Krysia has shown great patience, and given me much advice and assistance throughout my studies. I am greatly in her debt.

It is a pleasure to have worked with Martin Sadler when he was at Imperial College. Martin's influence was pervasive here, and he was always a source of inspiration and encouragement throughout the GENESIS project. I hope his influence can be seen in this thesis as well.

I would like to thank Tom Maibaum and Samson Abramsky for their continuing support and encouragement. Friends, colleagues and students of the Department of Computing have been a constant source of stimulation. Paul Taylor deserves particular thanks for writing his excellent TEX macros that have greatly improved the presentation of the Thesis, as has his dedication to the task of providing a modern TEX environment at Imperial College. Special thanks must go to Jan-Simon Pendry for his work converting *ω-prolog* to C++, writing its module language and introducing many enhancements. David James' TEXpertise fathomed major difficulties with the technology used to print the figures without scissors and glue. Mark Ryan, Krysia, Tom, Martin, Paul and David have kindly read drafts of this thesis and made many thoughtful comments and corrections; needless to say, any errors that remain are mine. My thanks go to them all, and to the many other people at the College who have helped me in one way or another.

Contents

List of Figures

Chapter 1

Introduction

It is a truth universally acknowledged, that a Computer Scientist in possession of a good problem, must be in want of a logic.

(with apologies to Jane Austen)

This thesis describes the motivation, design and implementation of an environment which supports the manipulation of a wide variety of logical and formal systems on a computer workstation.

This work has been motivated by the desire to bring together a large number of logical systems within a single *framework* for study and analysis. By providing a clear account of the framework's relationship with other presentations of logical systems, it is hoped that the environment will be a useful tool for devising and developing new logics. The objective of the environment is to use a computer: to explore the immediate consequences of a logic, to expose combinatorial relationships that may exist between logical operations in a system and to verify that properties hold of the system as a whole. The last two aspects can be used to elicit meta-mathematical properties of the system.

The framework described here allows a logical system to be formulated in a natural way. Logical systems are presented as a collection of inference rules in an all-introduction style. These rules can be given a computational interpretation such that theorem proving within a system can be carried out largely automatically from its definition.

The environment is designed to be usable by people who may not be completely familiar with computer systems. To achieve this "ease of use," there is a graphical interface that provides a controlled dialogue between the user and the environment.

The logical systems themselves are easily formulated, and the environment supports a concrete syntax which allows users to describe systems in a natural way. To this end, Roman, Greek and mathematical symbols are made directly available.

A second class of user - the "meta-user" - can make use of the *ω-prolog* meta-language to increase the functionality of the environment. Such new facilities are then available to users via the graphical interface. A meta-user can provide tools that allow ordinary users to conduct analyses of a system. There are two distinct types of tool considered here: one constructs a series of related object-level judgements and uses their derivability to guide its search; the other puts the rules of the system together at the meta-level.

An example of the first type of analysis is the procedure used to determine the interderivability of a collection of formulas. Another example is the discovery of distinct modalities in a modal system. These tools assist a user by exposing relationships between the logical operators of a system.

The second type of tool can be used to verify that a logical system has its intended properties. An inference rule can be thought of as expressing a property, or relation, over the syntactic structures used in a system. For example, that the consequence relation is transitive or monotonic. However, these rules, unlike the all-introduction rules, are very general and have little computational content to a theorem prover. Rather, the interest in them arises from the properties they express: if the rules are (implicitly) present they may lead to meta-mathematical results about the system.

An inference rule is a *derived* rule with respect to the other rules of the system when it can be expressed in terms of them in some uniform way. The goal here is to show that these general inference rules are derived rules with respect to the computationally effective rules, and thereby that the system has the required properties. To show that a rule is derived requires an inductive argument over the structure of possible derivation trees in the system. Such an argument is very sensitive to the formulation of the logic, and may be upset by slight changes in its presentation which subtly change the ways in which the rules can interact. Frequently, a large number of cases must be considered, and these can be tedious to check manually. Such combinatorial theorems are good candidates for mechanisation when it is possible to identify the necessary sub-cases and there are heuristics available for solving them. The study of derived rules is the second type of tool present in the framework/environment described here.

The overall objective of this work is to provide an environment in which a user can model the logic underlying particular classes of reasoning, or problem solving.

The resulting system describes the user's intuitions about the problem, and the environment allows the user to refine the system until it has the required consequences and necessary overall properties.

The benefits of the environment can be summarised as:

1. it provides a mechanical theorem prover which allows the user to conduct derivations within a system;
2. the user may undertake 'case-studies' about a system using procedures that attempt many small object-level derivations within it;
3. it can be used to establish the status of derived rules, and thereby meta-mathematical properties of a system.

1.1 Logics in Computer Science

There is a tendency throughout Computer Science towards the adoption of logics that reflect the type of reasoning appropriate to a specific problem domain. This trend is fairly recent and perhaps grows partly from the long tradition of producing programming languages tailored to problem domains. Also influential is a move away from the monistic dogma of First Order Predicate Calculus, towards instrumentalism and pluralism[Haa78]. There are many examples of the application of different logics to different branches of Computer Science including:

Programming languages whose study has been given a logical basis through the work of Tony Hoare[Hoa89]. His program logics formalise the treatment of assertions about the behaviour of programming language constructs. Each programming language construct gives rise to one or more inference rules that state the construct's effect on the assertions surrounding it, *e.g.* the Hoare triples $\{\varphi\}p\{\psi\}$, read as "if the condition φ is satisfied before executing the program fragment p, then the condition ψ is satisfied afterwards". Dynamic Logics, developed by Pratt [Pra80] and Goldblatt [Gol82], are more general. They contain modalities indexed by program fragments, *e.g.* $[p]\psi$, read as "the condition ψ must hold after executing p". In a Dynamic Logic there is a correspondence between the possible states of the computer and the possible worlds of the modal system. A world contains assertions that characterises a possible configuration of the computer. Execution of the program then corresponds to transitions between worlds. In the context of Dynamic Logic a Hoare triple is $\varphi \rightarrow [p]\psi$.

Domain Theory which is traditionally given in model-theoretic terms such as Information Systems[Sco82] has been placed in a logical setting by Abramsky[Abr87]. His framework allows the production of logics tailored to particular domains, *e.g.* for concurrency, or the λ-calculus.

Concurrency where the notion of time and causality are important, *e.g.* temporal logics [Gol87], the transition systems of the Calculus of Communicating Systems[Mil89] and Hennessy-Milner's modal logic for characterising bisimulation in such systems. Another example is the calculus of Communicating Sequential Processes[Hoa78, Hoa81] in which Hoare provides a proof theory of process expressions.

Type Inference Systems have become an integral part of a number of programming languages to check or infer the type of programming language constructs. The functional programming language ML has a polymorphic type system which is used to infer the most general type for functions and expressions [DM82].

Specification languages arose from the need to specify the intended behaviour of programs. One influential approach is Cliff Jones' Vienna Development Methodology, VDM, which uses a special three-valued logic, the Logic for Partial Functions[Che86], to model the possibility of undefinedness of programs for some inputs.

Another type of specification is offered by Per Martin-Löf's Constructive Type Theory[Mar72]. Here, specifications are types in the type theory. A proof that a type is well-constructed gives rise to a functional program as its "witness" through the propositions-as-types paradigm (see below). This shows that there is a precisely defined relationship between the specification and a program satisfying the specification. The algorithms and structure of the program depend on how the specification is derived. An adaptation of this Type Theory aimed more towards the abstract data types used in Software Engineering is given by Querioz[dQ88].

Automated Theorem Proving has been steadily developed since Herbrand's results and the introduction of the Unification algorithm by Robinson in 1965[Rob65]. The programme uses a reduction of classical first-order logic to clausal form and the resolution method. Subsequent improvements in this technology have led to theorem proving systems capable of proving new results in some branches of mathematics[W+84].

Mechanical Theorem Proving also has a long tradition. Much effort sprang from a desire to mechanise Dana Scott's Logic of Computable Functions.

This led to Stanford then Edinburgh LCF[GMW79] and Cambridge LCF[Pau88]. As other logics were put forward, the ML-based technology of LCF theorem provers was deployed, *e.g.* [Pet82]. A generalisation of this approach is the Isabelle system[Pau89] which uses Higher-Order Logic as the meta-language for formalising many object-logics.

Logic programming characterised by the PROLOG language[Kow74, CKPR73] is the application of a special case of the resolution method used by the automated theorem proving community. Clauses are restricted to a having a single positive literal. When Horn-clauses are combined with a predictable search strategy the theorem prover behaves like a programming language.

Artificial intelligence has many applications of modal logics of knowledge and belief for modelling the reasoning of individuals. Non-monotonic systems are also used for handling belief revision or reasoning with defaults. A system is called non-monotonic when a conclusion that holds at one point may not hold once additional information is available. For an overview, see the works of Genesereth and Nilsson [GN87] and Turner [Tur84].

Several formal systems have been constructed for the purpose of formalising parts of mathematics. An early example is that of the AUTOMATH project [dB80] which arose from a desire to check parts of mathematics on a computer. This was done for Landau's Grundlagen[Jut77]. Other systems have been built with similar objectives in mind. Theirry Coquand and Gerard Heut's Calculus of Constructions[CH85] provides a more recent version of the AUTOMATH concept. In the propositions-as-types paradigm[How80], the types of a typed lambda-calculus are identified with the propositions of a logic and the terms of the calculus are identified with derivations that justify the propositions. This has provided the inspiration for a number of Constructive Type Theories. For example, Robert Constable's group at Cornell has integrated the Type Theory of Martin-Löf with their earlier Proof Refinement Logic (PRL) to produce a new system Nuprl[C$^+$86]. This has been used to formalise parts of constructive mathematics. In these examples, the effort is focused on supporting theories representing bodies of mathematics, rather than on the presentation of any underlying logical systems.

The Isabelle system [Pau89] is an example of a system designed for the formal representation of a variety of logics within a single meta-logic. The meta-logic used is Higher-Order Logic; a set of axioms in this defines an object-logic. Deductions in an object-logic are formalised as equivalent deductions in the meta-logic.

The approach adopted in this thesis is to represent object-logics in a less formal way, but a way that directly reflects the proof-theoretic nature of the presentation. If the meta-logic approach were followed for the framework described here, the representation of inference rules (and their structural properties) would indeed be theories in the meta-logic, but the relationship would not be as elegant as it is in Isabelle.

1.2 Thesis

This thesis presents an framework that is designed for the development and manipulation of a wide variety of logical systems. This approach is original in its aim of providing tools to assist the construction of logics: the framework has been implemented on a computer workstation as a logic development environment.

The definition of a logical system is taken to have two components: the language from which the syntactic elements, including judgements, of the system are constructed, and a deductive part, where relations over the judgements of the system are expressed as inference rules. A method is given for transforming different syntactic presentations of logical systems into the all-introduction style required. This is applied to Axiomatic and Natural Deduction presentations of several systems, including: intuitionistic, relevance, linear, classical, modal and many-valued logics.

It is shown that the framework provided by the environment is sufficiently expressive to capture presentations of these systems with the inference rules alone. The prototype environment has been used to construct presentations of Classical Logic; Intuitionistic Logic; Type Inference Systems; the Classical Modal Logics: K, T, K4, S4, S5 (in various styles of presentation); some of their Intuitionistic counterparts; the Deontic Systems: D, D4; some Modal logics of Knowledge and Belief; Temporal logics; Linear Logic; Three-valued logics; and some other finite many-valued logics.

A number of tools are provided for the analysis of systems presented within the environment. One of these checks whether a judgement holds within a given system by constructing a derivation using inference rules. As it is intended that the environment be used for the development of presentations of systems, a simple strategy language is provided to select between inference rules.

A second tool allows a user of the environment to conduct case studies which reveal the relationships between connectives. In particular, it is shown how the distinct modalities of some classical and intuitionistic modal systems can be found and their inter-relationships discovered.

A third, and more fundamental, type of analysis is also discussed. For example, when a logical system is extended by the addition of a new inference rule, it may not be possible to prove any new theorems. When this is the case, the rule is said to be *derived* with respect to the other rules of the system, and is not computationally useful. If a property of a judgement is expressed as an inference rule and the new rule is found to be derived with respect to the other, more computationally effective rules, then the property was previously implicitly present. This knowledge can be used to give results about the presentation of the system as a whole, for example whether the system is consistent.

This thesis has also been written to explain aspects of the environment to a potential user. This is part of the motivation for exploring the types of transformations required to produce suitable presentations of systems for the framework. A secondary objective is to provide a reasonably coherent explanation of the prototype environment's inner workings, should someone wish to extend or maintain it. In some respects, these two interests are unlikely to coincide in a single individual and so the reader's indulgence is sought in advance.

1.3 The Framework

The framework allows the specification of the two principal components of a logical system. The first of these is the set of linguistic structures that are required to construct the syntactic entities of the object-logic's language. The second part is a specification of the deductive properties of the object logic. The language of a system consists of a declaration of the *syntactic categories* that are involved. Of these categories one or more are designated as *judgements*. The judgement categories are the conveyors of meaning for a system and define the relations between other syntactic categories within it. For example, the relation may be regarded as holding when some formula is a consequence of some other formulas. However, other interpretations of judgements are possible. Nor is a system restricted to a single type judgement. The precise form of the judgement is left open; which is an important aspect of the flexibility of the framework.

The *meaning* of the judgements is given by their definition as inference rules in the deductive part of the definition of a system. Inference rules capture the meaning of the other syntactic entities in the language by showing how they relate to each other, and the precise conditions under which they can be *used*. Often, the judgement of a

system is a *consequence relation.* The inference rules take the form

$$\frac{J_1 \cdots J_n}{J}$$

where $n \geq 0$ and each of the J_i belongs to a judgement category. The judgements forming an inference rule are allowed to contain schematic variables that act as placeholders for actual elements of the categories they represent.

If the language of a system is denoted by $\mathcal{L}$ and its deductive part by R, then a system can be stated as the pair

$$S = \langle \mathcal{L}, R \rangle$$

Before launching into a detailed discussion of how a system is defined, it is appropriate to digress briefly to give a short review of consequence relations. Consequence relations are a typical example of the type of judgements the framework is designed to support.

1.3.1 Logical Consequence

The framework provides a way of formulating presentations of logical systems in terms of their consequence relations.

Suppose that the language of a system, $\mathcal{L}$, defines the syntactic category called 'formula' that consists of propositions and some logical operators such as: $\wedge$, $\vee$, $\neg$, $\rightarrow$; and that the set of all formulas that can be written down in this category is $\mathcal{U}$. Then, given a collection of assumptions taken from this set, $\{\varphi_1, \ldots, \varphi_n\} \subset \mathcal{U}$, the set of all conclusions that can be reached from the assumptions using the deductive power of the system is given by a function $\mathbf{Cn} : \mathbf{P}\mathcal{U} \rightarrow \mathbf{P}\mathcal{U}$. So $\mathbf{Cn}(\{\varphi_1, \ldots, \varphi_n\})$ stands for the set of inferences that are derivable from the assumptions. The relation $\mathbf{Cn}$ has a fixed point such that $\mathbf{Cn}(\mathbf{Cn}(S)) = \mathbf{Cn}(S)$; $\mathbf{Cn}$ can be defined inductively by:

$$\begin{aligned} \mathbf{Cn}_0(S) &\doteq S \\ \mathbf{Cn}_{k+1}(S) &= \mathbf{Cn}_k(S) \cup \{\varphi | \frac{\varphi_i \cdots \varphi_n}{\varphi} \in R \text{ and } \varphi_i \in \mathbf{Cn}_k(S)\} \end{aligned}$$

At about the same time as Tarski's $\mathbf{Cn}$ was proposed, Hertz introduced the relation '$\vdash$' between finite sets (or rather sequences) of formulas and their consequences in the following way:

$$\varphi_1, \ldots, \varphi_n \vdash \psi \text{ iff } \psi \in \mathbf{Cn}(\{\varphi_1, \ldots, \varphi_n\})$$

Hertz's notation was later extended by Gentzen to allow multiple conclusions:

$$\varphi_1, \ldots, \varphi_n \vdash \psi_1, \ldots, \psi_m$$

The interpretation given to such an expression is that the conjunction of φ_i has the disjunction of the ψ_j as a logical consequence. Or, put differently, that the truth of all of the assumptions entails the non-falsity of at least one of the conclusions.

A consequence relation over finite sets of formulas can be defined as '$\Gamma \vdash \Delta$' where $\Gamma, \Delta \subset \mathcal{U}$ and $\varphi \in \mathcal{U}$. A convenient shorthand is used to obviate the need for braces:

$$\begin{aligned} \Gamma, \Delta &= \Gamma \cup \Delta \\ \Gamma, \varphi &= \Gamma \cup \{\varphi\} \end{aligned}$$

Using this notation, some of the properties of $\vdash$ can be stated as: reflexivity (R), monotonicity (M), and transitivity (T); and presented as follows:

$$\text{(R)} \quad \Gamma \vdash \Delta \quad \text{if} \quad \Gamma \cap \Delta \neq \varnothing$$

$$\text{(M)} \quad \frac{\Gamma \vdash \Delta}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

$$\text{(T)} \quad \frac{\Gamma \vdash \varphi, \Delta \quad \Gamma, \varphi \vdash \Delta}{\Gamma \vdash \Delta}$$

The conditions R, M, and T, may or may not be present in a consequence relation of a system. If their presence was insisted upon, then this would exclude several families of logics that the framework should support. Certainly, monotonicity does not hold in some of the systems we wish to represent, for example Linear Logic, Relevant Logics, and non-monotonic logics to do with defaults and defeasible reasoning. Linear Logic restricts the use of assumptions in the proof; an assumption must be used in the proof precisely once whereas monotonicity would allow an assumption to be introduced without it being used. Relevance Logics similarly require that an assumption be used at least once in a proof. Non-monotonic logics are popular in Artificial Intelligence for reasoning about systems which model the accumulation of knowledge. Here, increasing knowledge can lead to the rejection of previously acceptable inferences. It is possible that neither reflexivity nor transitivity will hold where a relation is between different kinds of objects. Avron calls consequence relations with (R) and (T), *simple* in [Avr87].

What is termed a *logic* in the framework varies according to the nature of the consequence relation being defined. Sometimes the more neutral term *system* is preferred to logic. For many systems, it is enough to take (R) and (T) as desired properties of the relation. However, in order to be pluralistic, the framework does not take a rigid view of what constitutes a consequence relation in this sense.

When a system is presented over a (multiply conclusioned) consequence relation the inference rules in the deductive part of the system take the form:

$$\frac{\Gamma_1 \vdash \Delta_1 \quad \cdots \quad \Gamma_n \vdash \Delta_n}{\Gamma \vdash \Delta}$$

where $n \geq 0$.

If a judgement of the form '$\Gamma \vdash \Delta$' is interpreted as the argument "assuming that all the assumptions Γ hold then at least one of the conclusions Δ holds", then the inference rules can be seen as governing what is and is not admissible as a valid argument. The condition of reflexivity states the self-evident fact that an assumption entails itself. In this way an inference rule can, depending on the number of antecedents present:

$= 0$ introduce an obviously valid argument;

$= 1$ transform a valid argument into another valid argument; or else

≥ 2 combine several valid arguments into a single valid argument.

1.3.2 General Properties

There are however two important properties, *finiteness* and *uniform substitution*, that are built into the framework.

Finiteness or compactness. If some argument holds then it requires only a finite number of assumptions.

$$(\mathrm{F}) \quad \frac{\Gamma \vdash \Delta}{\Gamma_n \vdash \Delta_m} \; n,m \text{ finite}$$

Uniform substitution If some argument holds then so does the corresponding argument obtained by systematically replacing some the schematic variables in it for other instances of the categories they represent.

$$(\mathrm{U}) \quad \frac{\Gamma \vdash \Delta}{\Gamma\theta \vdash \Delta\theta} \; \theta \text{ is a substitution}$$

Finiteness is forced by the framework; rules cannot have more than a finite number of antecedents. Uniform substitution is likewise built into the way in which the framework performs substitution, and is always present in a system which does not use negative judgements[1].

We now return to the discussion of the framework.

1.3.3 Language

The language of a system is defined by its *syntactic categories.* Categories define classes of syntactic entities such as Formulas, Terms, Sequents. Corresponding to each category is at least one *formation rule.* Formation rules say how the elements of the category are constructed, either in terms of other categories, or in terms of primitive syntactic units. It is also possible to use operators to form collections of categories with particular properties, *e.g.* Sets, Bags, Lists. The definition of the language for propositional intuitionistic logic is shown in figure 1.1.

Categories: Sequent Formula[$\varphi\psi\theta$] SetOf(Formula)[$\Gamma\Delta\Theta$] ;
Judgements: Sequent;
a:SetOf(Formula) '$\vdash$' c:Formula $\rightarrow$ SEQ(a,c):Sequent
a:Formula '$\wedge$' b:Formula $\rightarrow$ AND(a,b):Formula
a:Formula '$\vee$' b:Formula $\rightarrow$ OR(a,b):Formula
a:Formula '$\rightarrow$' b:Formula $\rightarrow$ IMPLIES(a,b):Formula
a:Formula '$\leftrightarrow$' b:Formula $\rightarrow$ IFF(a,b):Formula
'$\neg$' a:Formula $\rightarrow$ NOT(a):Formula
'$\perp$' $\rightarrow$ BOT:Formula
t:'[abc]' $\rightarrow$ ID(t):Formula

Figure 1.1: Language of intuitionistic logic

1.3.3.1 Categories

There are just two categories required: Formula and Sequent.

[1] A negative judgement allows a system to be introspective, and enables the outcome of a proof to depend on the non-derivability of its components. Clearly, substitutions may change these components, and hence the outcome of the proof. The defeasible system in §3.7 illustrates the use of a negative judgement.

1.3.3.2 Metavariables

The names of schematic, or metavariables, are declared at the same time as the categories are introduced. A metavariable declaration gives the root names of variables that act as placeholders for elements of the category to which the metavariable belongs. They are used to make the definition of rules schematic and by declaring them as part of the system's language, avoid the need to declare schematic variables in the deductive part.

The declaration $[\varphi\psi\theta]$ indicates that the metavariables for the category of Formula have the form $\varphi, \psi, \theta, \varphi', \psi', \theta', \ldots, \varphi_0, \psi_0, \theta_0, \ldots$ Only the first character of the variable name is indicated.

It is not necessary to declare metavariables for categories that are not 'talked about' by inference rules.

1.3.3.3 Judgements

The judgement category, Sequent, is declared separately.

```
Judgements: Sequent;
```

Judgements are 'external' categories. Other categories are 'internal' to the language and are not permitted to form the basis of queries or appear by themselves in rules. Judgements are not restricted to the single form of consequence relation defined above and more elaborate relations than the usual binary relation '$\vdash$' can be used. Examples are ternary or n-ary relations which may be used to represent many-valued systems. In some cases 'hypersequents' (or sequences of sequents) can be used for the presentation of some modal, many-valued, and relevance logics. The issues of representing non-standard logics are discussed in chapter 3.

1.3.3.4 Formation Rules

Each category has at least one formation rule. The formation rule for the Sequent category is defined by

```
a:SetOf(Formula) '⊢' c:Formula → SEQ(a,c):Sequent
```

An element of the category of **Sequents** is '**SEQ(a,c)**' when '**a**' and '**c**' are elements of the categories '**SetOf(Formula)**' and '**Formula**' respectively. The declaration provides the abstract syntax of the element for internal use. It also gives a pattern that can be used to parse an element of the category and build the internal abstract representation. Similarly, the declaration is used to reconstruct the familiar concrete syntax of the element from the internal form.

The logical operations, or connectives, in the category of Formulas are defined in similar way. For example the following introduces the operation '∧' (and).

a:Formula '∧' b:Formula → AND(a,b):Formula

1.3.3.5 Constants

The propositional constants are defined to be words starting with the roman letters 'a', 'b', or 'c':

t:'[abc]' → ID(t):Formula

where '[abc]' is a regular expression in the sense of UNIX and formal language theory.

1.3.3.6 Collections of a Category

The expression '**a:SetOf(Formula)**' introduces a collection of Formulas. The collection forming operations are a convenient way of packaging the comma convention described in §1.3.1. For example, using the metavariable declarations given above, the following abbreviations are possible:

$$\begin{array}{rll} \varphi, \psi & \text{for} & \{\varphi, \psi\} \\ \Gamma, \varphi \wedge \psi, \Delta & \text{for} & \Gamma \cup \{\varphi \wedge \psi\} \cup \Delta \\ & \text{for} & \varnothing \end{array}$$

The effect of collections is to make writing inference rules less clumsy and more convenient. An extension of this notation is supported: It is permissible to write $\neg\Gamma$ and $\Gamma \wedge \Delta$; these stand for the set of formulas that have the form $\neg\varphi$ and $\varphi \wedge \psi$, *etc.* The collection variables Γ and Δ are used to refer to the appropriate subformulas of the matching formulas. If the pattern is "$\neg\Gamma, \Gamma'$", then the division of formulas is as follows:

$$\underbrace{\neg a, \neg(b \wedge c)}_{\Gamma=\{a, b\wedge c\}}, \quad \underbrace{d, \neg c \wedge e}_{\Gamma'=\{d, \neg c\wedge e\}}$$

Patterns are particularly convenient for presenting the rules of modal systems.

The second aspect of the definition of a system is its deductive part, presented as a collection of inference rules.

1.3.4 Inference Rules

Inference rules give meaning to the syntactic structures of a system by showing the ways in which the structures interact. The inference rules present in a system can be divided according to their intended use, by the structures they operate on, their suitability for backwards reasoning, and so forth. Rules should be given names that reflect these considerations. In the environment some of these characterisations are made by the framework, whereas others are given by a user defining a system.

The structure of a rule can be simply stated as

$$\frac{J_1 \quad J_2 \quad \cdots \quad J_n}{J}\ \textsf{side-condition}$$

where each Js is an instance of judgement categories, usually containing schematic variables. The **side-condition** is optional, and when present acts as an additional guard on the applicability of the rule. The name of a rule in the environment is a list of identifiers, separated by dashes (-). Each identifier indicates a property of the rule that is important to the user, and should by convention suggest the rule's use. Several rules can be selected at once as part of a simple strategy language by using the presence or absence of their constituent names. The following is an example of a rule governing the introduction of $\wedge$ on the right-hand side of the intuitionistic sequent:

$$\frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\ \textit{AND-right}$$

The label indicated to the right of the rule gives the rules name. (In the thesis this is often written in symbols, *e.g.* *AND-right* becomes $\vdash\wedge$.) The rule can be read as constructing an argument justifying $\varphi \wedge \psi$ from the two individual arguments justifying φ and ψ separately, but from the same set of assumptions.

The rules defining a consequence relation can be divided into three kinds: *basic*, *structural* and *logical.*

1.3.4.1 Basic rules

Basic rules are those that follow unconditionally and allow the introduction of arguments that are valid without further justification - axioms, *e.g.*

$$\frac{}{\varphi \vdash \varphi}$$

which admits a trivial argument.

1.3.4.2 Structural rules

Structural rules detail the manner in which assumptions can be manipulated. They can act to allow an individual assumption to be freely introduced into the argument, or permit several occurrences of the same assumption to be reduced to fewer occurrences of that assumption. Structural rules also act on groups of assumptions for example allowing them to be reordered. Depending on the nature of the logical system it may require any of these, or other structural properties.

The use of collection categories in the definition of language can avoid the need for explicit structural rules in many cases.

Structural rules can often be absorbed into definition of a system's language. In this framework there are essentially two ways of giving the structural rules for the logic.

a:ListOf(Formula) '⊢' c:ListOf(Formula) → SEQ(a,c):Sequent

$$\frac{\Gamma, \varphi, \psi, \Gamma' \vdash \Delta}{\Gamma, \psi, \varphi, \Gamma' \vdash \Delta} \quad \frac{\Gamma \vdash \Delta, \varphi, \psi, \Delta'}{\Gamma \vdash \Delta, \psi, \varphi, \Delta'}$$

$$\frac{\Gamma, \varphi, \varphi, \Gamma' \vdash \Delta}{\Gamma, \varphi, \Gamma' \vdash \Delta} \quad \frac{\Gamma \vdash \Delta, \varphi, \varphi, \Delta'}{\Gamma \vdash \Delta, \varphi, \Delta'}$$

Figure 1.2: Explicit structural rules

Perhaps the traditional way, certainly for the sequent calculus, is to treat the left and right-hand sides of the consequence relation as sequences. Sequences are here termed *lists* and rules are then written to manipulate these lists. These rules are referred to as *structural* rather than logical, as no logical operators are affected. Structural rules have a meaning in the sense that they specify the way in which assumptions can be considered (in sequence, or in any order), and indicate whether assumptions may, for example be duplicated (or, according to the reading, contracted). Figure 1.2 shows the declaration required for a sequent with explicit structural rules.

a:SetOf(Formula) '⊢' c:SetOf(Formula) → SEQ(a,c):Sequent

Figure 1.3: Implicit structural rules

In this framework structural rules may be given implicitly by using, as part of the language, collections of categories with properties that allow rearrangement or duplication (*e.g.* sets, bags). Figure 1.3 shows the sequent defined using *sets* rather than lists.

The advantages of the second approach over the first are that:

- the proofs are shorter, as structural steps are absorbed by the framework when conducting proofs in the logic;
- the theorem prover has greater control over the proof procedure, as properties of collections can be used when they are needed rather than in a disorderly fashion.

1.3.4.3 Logical rules

Logical rules give meaning to the logical operations. As an example of this consider the following two rules:

$$\frac{\varphi, \psi, \Gamma \vdash \Delta}{\varphi \wedge \psi, \Gamma \vdash \Delta} \wedge\vdash \quad \text{and} \quad \frac{\Gamma \vdash \Delta, \varphi \quad \Gamma \vdash \Delta, \psi}{\Gamma \vdash \Delta, \varphi \wedge \psi} \vdash\wedge$$

These two rules give the meaning of the '$\wedge$' operator when it occurs in an assumption ($\wedge\vdash$) and when it occurs in a conclusion ($\vdash\wedge$). The first rule allows us to form the 'and' of any two assumptions. The second rule allows us to conclude 'φ and ψ' only when we can conclude both 'φ' and 'ψ' independently. This form of definition of the behaviour of a connective is an *all-introduction* presentation - the rules show the conditions which must pertain before the connective can be introduced as an assumption or conclusion of the sequent. When a system is presented in this way, it is more suitable for backwards reasoning.

1.3.4.4 Side-conditions

Side-conditions are essentially *guards* on the applicability of the rule and are decided from the instantiated form of the judgements in the rule. In this sense they are *observable* properties; they can be determined 'at a glance'. Side-conditions provide additional syntactic capabilities for the rule and may, for instance perform a specialised pattern recognition, or check for the existence of free/bound variables. But, whatever the function a side-condition provides, it must be effectively decidable from the actual form of the judgements when the rule is used.

To illustrate the importance of the decidable aspect of a side-condition, consider the behaviour of the theorem prover using rules in a backwards or bottom-to-top sense. Before a rule may be used, all of its side-conditions must be satisfied using only the information available in the concluding judgement of the rule. If it is not possible to decide the side-condition, or if the antecedents of the rule are used before the side-condition is satisfied, the theorem prover may attempt to find a derivation for an ill-formed judgement; this can cause it to enter an internal loop.

In many cases where a side-condition could be used, it can usually be replaced by a more refined choice of syntactic categories, or else by a *pattern* or a *binder*. Experience with side-conditions prior to the introduction of patterns and binders was that unavoidable use of side-conditions pointed to useful extensions of the framework. In this sense they can have a positive role as design heuristics.

1.3.5 Example System

When taken together figures 1.1 and 1.4 give a presentation of Intuitionistic propositional logic.

All-introduction Rules for Intuitionistic Logic

$$\frac{}{\Gamma, \varphi \vdash \varphi}\ basic \qquad \frac{\Gamma \vdash \varphi \quad \Gamma, \psi \vdash \theta}{\Gamma, \varphi \to \psi \vdash \theta}\ \to\vdash \qquad \frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\ \vdash\to$$

$$\frac{\Gamma, \varphi, \psi \vdash \theta}{\Gamma, \varphi \wedge \psi \vdash \theta}\ \wedge\vdash_1 \qquad \frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\ \vdash_1\wedge$$

$$\frac{\Gamma, \varphi \vdash \theta \quad \Gamma, \psi \vdash \theta}{\Gamma, \varphi \vee \psi \vdash \theta}\ \vee\vdash_1 \qquad \frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}\ \vdash_1\vee a \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}\ \vdash_1\vee b$$

$$\frac{\Gamma \vdash \varphi}{\Gamma, \neg\varphi \vdash \psi}\ \neg\vdash_1 \qquad \frac{\Gamma, \varphi \vdash \bot}{\Gamma \vdash \neg\varphi}\ \vdash_1\neg$$

Figure 1.4: Intuitionistic Rules

1.4 Activities

Once a system has been presented in the framework, it can be put to a number of different uses. These are divided into two distinct types: proofs *within* a logic, and proofs *about* a logic.

1.4.1 Proofs within a Logic

Proofs are conducted by the environment with respect to a logic and a particular *strategy* which specifies which rules are involved and the order of their application.

The proof procedure makes use of the form of rules to decompose a judgement into successively smaller and smaller judgements. This is aided by the *sub-formula property* and an *all-introduction presentation* of the system. The sub-formula property requires the size of the antecedents of a rule to be no greater than the size of its conclusion. The all-introduction presentation ensures that that the definition of each connective is achieved through its occurrences in the conclusion of the rule alone. Derivations are conducted in this 'backwards' sense guided by a strategy. The strategy divides the derivation tree into regions according to the choice of rules available. This results in the 'layering' of the derivation tree illustrated by figure 1.5 where the arrowed line indicates a branch through the derivation tree from its root to a leaf. The nodes of the tree are applications of inference rules.

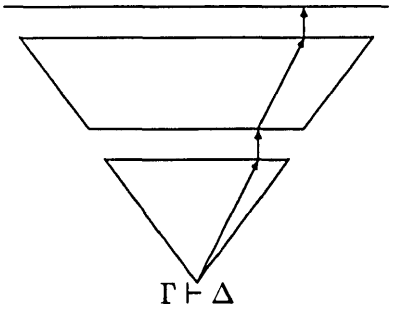


Figure 1.5: Derivation Tree

Strategies also convey information about the 'force' of a region. The force can affect the shape of the final proof. Two forces are provided: eager and lazy. Eager regions attempt to be as large as possible, whereas lazy ones grow only when they must - that is when it is not possible to complete the derivation beyond them. A third type of region inserts an occurrence of a single rule.

Strategies can also affect the choice of logic by specifying precisely which rules are used at each stage of the derivation. So a strategy can for instance ignore a number of rules. This is useful for developing a family of related systems as a single collection of rules. A global constraint can be given that places bounds on the depth of the derivation, or the number of times certain rules are used in a branch. Further details of strategies are given in Chapter 7.

The proof tree is constrained further during its construction so that every extension of it is irredundant. This simple technique acts as a loop checker.

Since strategies are free to ignore rules, they could be thought of as part of the specification of the logic. Proofs and the derivation of properties are usually done *relative to* strategies.

1.4.2 Proofs about a Logic

Given a presentation of a new system, one of the first questions that a user of an environment will ask is: "what properties does it have?" This might be refined further by asking how the connectives interact one with another. Are there normal forms? Is the system based on a consequence relation? Depending on the applications of the system, these may or may not be important questions to answer. An environment of the type described should provide tools that support activities like these.

1.4.2.1 Normal forms

The search for normal forms takes various forms depending on the system. For classical logic it is possible to place formulas in prenex-form, whereas this is not possible for an intuitionistic system. Similarly a system may allow all formulas to be written in disjunctive or conjunctive normal forms.

A type of analysis that is particularly useful for presentations of modal system is the identification and relationship between modalities. This involves an examination of the combinatorial relationship between the unary operators in the language, typically $\neg$, $\Box$ and $\Diamond$. This example is explored further in §3.3 for several interesting modal systems.

1.4.2.2 Derived rules

Inference rules state ways in which syntactic elements of a system can interact. It is sometimes convenient to formulate inference rules that state general properties of the system. Often, these rules are not useful to the theorem prover described here, as the rules are not suitable for backwards reasoning. However, it is often possible to show that the rules which *are* useful to the theorem prover capture the general properties which were required. In this case, the new rules are described as *derived* with respect to the other rules.

To show constructively that a rule is a derived rule, one must find a number of translations such that irrespective of where the derived rule might occur in a derivation, its occurrence can be replaced by an equivalent derivation that does not depend on the derived rule. If a rule can be replaced in this systematic way by other rules then the effect - or meaning - of the derived rule is implicitly present in the other rules.

By its nature, the structure of a derivation or proof tree gives rise to a large number of possible contexts in which the rules may occur. Each such case must be considered, and a transformation found. Consequently, the structure of possible proof trees is sensitive to the ways in which rules can interact with other rules and axioms. The construction must therefore be rechecked whenever any of the axioms or rules present in the system are changed. This type of activity is considered to be a candidate for tool support in the environment. The nature of derived rules is discussed in Chapter 5.

Consider the task of checking that the consequence relation of the example of figure 1.4 is a reflexive and transitive. For reflexivity, it is necessary just to inspect the rules to find one of the appropriate form.

$$\frac{}{\varphi \vdash \varphi}$$

is subsumed by the rule

$$\frac{}{\Gamma, \varphi \vdash \varphi}\ \textit{basic}$$

The case of transitivity is more complicated. The transitivity condition is expressed by a "cut" rule such as[2]

$$\frac{\Gamma \vdash \varphi \quad \Delta, \varphi \vdash \psi}{\Gamma, \Delta \vdash \psi}$$

which is not well behaved from the point of view of the inference engine described above. The reason for this is that the rule eliminates the formula 'φ' from the conclusions of the top two proofs $\Gamma \vdash \varphi$ and $\Delta, \varphi \vdash \psi$ to form a new proof of $\Gamma, \Delta \vdash \psi$ which contains fewer occurrences of the formula 'φ' (usually no occurrences). As the proof procedure works in a 'backwards' manner (using the rules from bottom-to-top), when it uses the cut rule, it has to guess a suitable formula to substitute for 'φ'. This guess may be inappropriate and the procedure may never find the correct choice. Fortunately Gentzen's great insight when formulating the sequent calculus was to observe that a system of rules may contain the effect of the cut rule *implicitly.* The cut rule may therefore be a *derived rule* with respect to the other *well-behaved* rules in the system. This result is related to Gentzen's Hauptsatz.

If the transitivity property can be verified, it can give rise to a number of important corollaries depending on the other connectives present. It can be verified that the proof procedure is complete, in the sense that it will work for all judgements that are derivable. In some cases, when there are clear bounds on the size of the search space, a decision procedure is obtained. For some logics, the Hauptsatz leads to consistency results, and for others it leads to the disjunction property and the interpolation theorem[3].

[2] although the precise formulation varies from logic to logic

[3] The Interpolation Theorem has been found to have important applications in the theory of

1.5 The environment

A prototype environment to support the framework has been constructed by the author. A number of illustrative graphics from this, and a discussion of its user interface are included in Chapter 6; also, graphics illustrating results for modalities are included in Chapter 3.

1.6 Structure of the thesis

The remainder of the thesis is structured as follows: In Chapter 2 a methodology for constructing systems of rules for the environment from other presentations is presented. Chapter 3 extends this methodology to treat a number of non-standard logics and gives the presentations used in the environment. Logics treated there include many modal logics and their applications together with some many-valued systems. The derivation of results about the distinct modalities in some modal systems is also shown. Chapter 4 discusses the issue of quantification. Chapter 5 describes the details of the technique for showing the status of derived rules and gives examples of how this leads to other useful meta-theorems. The environment's interface is discussed more fully in Chapter 6 and aspects of the implementation are presented in Chapter 7. Conclusions and a discussion of future work are given in the final chapter.

Specification where it is necessary to preserve conservative extensions of specification theories when they are *Implemented* in terms of another theory. It is the basis of the modularity result in [Sad84].

Chapter 2

Methodology

This chapter provides a user of the environment with guidance on formulating a logic within the environment, and provides techniques which may be used to relate different presentations of logical systems.

The chapter also justifies the use of all-introduction sequent presentations of systems in the environment. This is done by demonstrating how other proof theoretic presentations of systems can be transformed into an appropriate, equivalent, all-introduction presentation of the system.

2.1 Introduction

Three techniques for formalising the deductive process are examined: the Axiomatic, Natural Deduction and Sequent calculi. Each technique can be seen to capture the consequence relation of the particular logical system under consideration, but each uses a different syntactic presentation of the system. Consequently, each technique has advantages and disadvantages over the others in its ability to present different classes, or families, of logics. Similarly, each technique gives more or less computational insight into the deductive process of a system and arguably, therefore its suitability for automation on a computer.

As the deductive formalisms are surveyed, the progression from a presentation of a system in one formalism to a presentation of an equivalent system in the next is shown. Throughout this chapter three example systems are used as a guide through this process: Minimal Logic based on a single implication connective, Intuitionistic Logic and Classical Logic. Discussion of other logics and issues to do with quantification are deferred until Chapters 3 and 4 respectively.

2.2 Axiomatic Presentations

The axiomatic school is rooted in Frege's seminal Begriffsschrift of 1879[Fre67]. The tree-like notation for logical connectives used there was found too expansive for most people and so the axiomatic school is founded in notational terms by Hilbert and Bernays[Hil67], although the turnstile symbol ($\vdash$) originates with Frege.

An axiomatic presentation of a system consists of:

1. a syntactic category of formulas built over a category of propositions containing the connective '$\rightarrow$' over formulas; additional connectives are often used. For example, "If p is a proposition and e and f are formulas then p, $\neg e$, $e \wedge f$, $e \vee f$, $e \rightarrow f$ are formulas. Nothing else is a formula.";
2. metavariables over formulas, 'φ' 'ψ' 'θ';
3. a judgement category, '$\vdash \varphi$';
4. a number of formula schemas called *axioms*; and,
5. a rule of proof called Modus Ponens.

$$\frac{\vdash \varphi \quad \vdash \varphi \rightarrow \psi}{\vdash \psi} \text{ Modus Ponens}$$

A presentation may have several rules as well as Modus Ponens when quantifiers or modal operators are present in the category of formulas (chapters 3 and 4).

The judgement category designates formulas that are consequences of the axioms under the rules of proof. Axioms are schemas and stand for all the formulas that can be obtained by making consistent[1] substitutions of the axiom's schematic variables with different formulas. A category of judgements is defined by all the instances of axioms together with all the deductions possible from them by way of the rules of proof.

A proof of a formula in this 'pure' axiomatic framework is a tree formed by taking leaves to be instances of the axioms (*e.g.* the axioms A1–9 below). The following type definitions give an example of the structure of a pure axiomatic proof.

[1] a consistent substitution is one in which each occurrence of a particular schematic variable in the axiom or rule is made to stand for the same formula.

Tree(α, β) ::= *Leaf(α)*
| *Node(β, Tree(α, β) list)*

Axiomatic-proof-tree ::= *Tree(Axiom, Rule-of-proof)*

Axiom ::= *Formula*
Rule-of-proof ::= *Conclusion list* → *Conclusion*
Conclusion ::= *Formula*

The *Axiomatic-proof-tree* is a specialisation of the *Tree* type whose leaves are *Axioms* and whose internal nodes name the ***Rule-of-proof*** to be used at that point in the tree. The rule can then be thought of as a function which takes a list of conclusions of the proofs above it and returns a new conclusion. For backwards proof construction, the other direction - from the conclusion to the antecedents - is more useful. The application of the rule constructs a formula that is the conclusion of the proof. This example data-structure will be refined to illustrate the structure of proofs in the later examples.

The following displays show the axiomatic presentations which are to be used for or three example systems: Minimal logic in which there is a single implication connective; Classical logic with conjunction, disjunction and negation in addition to implication; and Intuitionistic logic in which there is a restriction on the nature of the negation. These axiomatic presentations are widely used and appear to be due to Kleene ([Kle52], p. 82).

Axioms for Minimal Logic

A1 $\varphi\rightarrow(\psi\rightarrow\varphi)$
A2 $(\varphi\rightarrow\psi)\rightarrow((\varphi\rightarrow(\psi\rightarrow\theta))\rightarrow(\varphi\rightarrow\theta))$

Axioms for Classical Logic

A1–A2 +
A3 $\varphi\rightarrow(\psi\rightarrow\varphi\wedge\psi)$
A4a $\varphi\wedge\psi\rightarrow\varphi$ A4b $\varphi\wedge\psi\rightarrow\psi$
A5a $\varphi\rightarrow\varphi\vee\psi$ A5b $\psi\rightarrow\varphi\vee\psi$
A6 $(\varphi\rightarrow\theta)\rightarrow((\psi\rightarrow\theta)\rightarrow(\varphi\vee\psi\rightarrow\theta))$
A7 $(\varphi\rightarrow\psi)\rightarrow((\varphi\rightarrow\neg\psi)\rightarrow\neg\varphi)$
A8 $\neg\neg\varphi\rightarrow\varphi$

Axioms for Intuitionistic Logic

A1–A7 +
A9 $\varphi \rightarrow (\neg\varphi \rightarrow \psi)$

An 'impure' axiomatic system is an extension of a pure system so that the *Leaves* of the proof can contain arbitrary formulas as well as axioms. The conclusion of an impure proof can be stated as '$\varphi_1, \ldots, \varphi_n \vdash \psi$' using a *sequent* where the φ_i are the non-axiom formulas present in the proof. The deduction theorem provides a means of transforming an impure proof into a pure one.

2.2.1 Deduction Theorem

Suppose there is a proof of '$\varphi, \varphi_1, \ldots, \varphi_n \vdash \psi$,' then the Deduction Theorem says that a proof of '$\varphi_1, \ldots, \varphi_n \vdash \varphi \rightarrow \psi$'; may be obtained, and hence, by repeated application, a proof of the form '$\vdash (\varphi_n \rightarrow \cdots \rightarrow (\varphi \rightarrow \psi) \cdots)$'. The proof of this meta-theorem is given by induction over the structure of the axiomatic proofs.

For the leaves of the proof, each assumption and axiom must survive the transformation from ψ to $\varphi \rightarrow \psi$. Each rule in the body of the proof must survive the passage from

$$\frac{\psi_1 \quad \cdots \quad \psi_n}{\theta}$$

to

$$\frac{\varphi \rightarrow \psi_1 \quad \cdots \quad \varphi \rightarrow \psi_n}{\varphi \rightarrow \theta}$$

which will be shown for the case of *modus ponens* - but must be verified for each rule if other rules are present.

A systematic procedure is defined that transforms a proof of the form $\varphi, \varphi_1, \ldots, \varphi_n \vdash \psi$ to a proof of the form $\varphi_1, \ldots, \varphi_n \vdash \varphi \rightarrow \psi$ - and by repeated application to a pure proof. The proof is an induction on the structure of the initial impure axiomatic proof.

1. θ is a leaf of the proof. Now θ is either an instance of an axiom or else an assumed formula.

   (a) θ is an axiom or an assumption not equal to φ. Construct a new leaf with an instance of axiom A1 and modus ponens:

   $$\frac{\theta \quad \theta \rightarrow (\varphi \rightarrow \theta)}{\varphi \rightarrow \theta}$$

(b) If the θ is equal to φ replace the leaf with the following derivation[2] of $\varphi \rightarrow \varphi$:

$$\dfrac{\overbrace{\varphi \rightarrow ((\varphi \rightarrow \varphi) \rightarrow \varphi)}^{\text{A1}} \qquad \dfrac{\overbrace{\varphi \rightarrow (\varphi \rightarrow \varphi)}^{\text{A1}} \qquad \overbrace{(\varphi \rightarrow (\varphi \rightarrow \varphi)) \rightarrow \varphi \rightarrow ((\varphi \rightarrow \varphi) \rightarrow \varphi) \rightarrow (\varphi \rightarrow \varphi)}^{\text{A2}}}{\varphi \rightarrow ((\varphi \rightarrow \varphi) \rightarrow \varphi) \rightarrow (\varphi \rightarrow \varphi)}}{\varphi \rightarrow \varphi}$$

2. $\dfrac{\psi \quad \psi \rightarrow \theta}{\theta}$ is an internal node. Recursively transform the proofs of ψ and $\psi \rightarrow \theta$ to $\varphi \rightarrow \psi$ and $\varphi \rightarrow (\psi \rightarrow \theta)$ respectively, and construct the new proof

$$\dfrac{\varphi \rightarrow \psi \quad \varphi \rightarrow (\psi \rightarrow \theta)}{\varphi \rightarrow \theta}$$

as follows

$$\dfrac{\varphi \rightarrow \psi \quad \overbrace{(\varphi \rightarrow \psi) \rightarrow ((\varphi \rightarrow (\psi \rightarrow \theta)) \rightarrow (\varphi \rightarrow \theta))}^{\text{A2}}}{\dfrac{(\varphi \rightarrow (\psi \rightarrow \theta)) \rightarrow (\varphi \rightarrow \theta) \qquad \varphi \rightarrow (\psi \rightarrow \theta)}{\varphi \rightarrow \theta}}$$

The Deduction Theorem will be of considerable use in the following sections.

2.3 Natural Deduction Presentations

Natural deduction removes a great deal of the clutter from the axiomatic style. Originally developed by Gentzen in his seminal paper [Gen69], Natural Deduction has been refined by the work of Prawitz [Pra65], and is increasingly popular in Computer Science as a formal framework for conducting proofs of properties of specifications (*e.g.* [Jon86]). Rather than having numerous axioms, natural deduction uses a number of *rules of inference.* Assumptions may be introduced at any stage, and may be discharged by certain of the rules. The discharge of assumptions is not local to the application of an inference rule, but may affect the proofs above it.

ND-proof-tree ::= *Tree(Assumption, Rule-of-inference)*

Assumption ::= *Formula*
Rule-of-inference ::= *ND-proof-tree list* → *(Assumption list, Conclusion)*
Conclusion ::= *Formula*

[2]The precise details of the proof vary according to the choice of axioms. The ones used here correspond to the types of the combinators K (axiom A1) and S (axiom A2). Also $\varphi \rightarrow \varphi$ corresponds to I = SKK.

A proof tree in the natural deduction framework is illustrated above. Here the leaves are assumed formulas rather than axioms. The internal nodes are applications of rules of inference that take the proofs above it to a pair consisting of the undischarged assumptions and the conclusion of the proof. In this way, a rule may adjust the assumptions appropriately.

In Natural Deduction presentations, the rules of inference are divided into two classes: *introduction* and *elimination.* For each connective there is an introduction rule and an elimination rule (and often more than one of each).

Nomenclature: A natural deduction rule that introduces (eliminates) a connective x is called '$x\mathcal{I}$' ('$x\mathcal{E}$').

Natural Deduction Rules for Minimal Logic

$$\frac{\begin{array}{c}[\varphi]\\ \vdots\\ \psi\end{array}}{\varphi\to\psi}\to\mathcal{I} \qquad \frac{\varphi \quad \varphi\to\psi}{\psi}\to\mathcal{E}$$

Taking the rules for minimal logic where there is just one connective ($\to$), there are two rules. Assumed formulas are written in square brackets and their discharge is indicated with a superscript. The index of the superscript is shared with the conclusion of the rule responsible for the discharge of the assumptions. So in our example of Minimal Logic the $\to\mathcal{I}$ rule may cause the discharge of (any number of) instances of the assumption $[\varphi]$ in the derivations of leading to ψ. The following illustrates the derivation of axiom A2 above.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\varphi\to(\psi\to\theta)]^2 \quad [\varphi]^1}{\psi\to\theta}\to\mathcal{E} \quad \dfrac{[\varphi\to\psi]^3 \quad [\varphi]^1}{\psi}\to\mathcal{E}}{\theta}\to\mathcal{E}}{\varphi\to\theta^1}\to\mathcal{I}}{(\varphi\to(\psi\to\theta))\to(\varphi\to\theta)^2}\to\mathcal{I}}{(\varphi\to\psi)\to((\varphi\to(\psi\to\theta))\to(\varphi\to\theta))^3}\to\mathcal{I}$$

Note that the topmost instance of the inference rule $\to\mathcal{I}$ discharges two applications of the assumption φ (both indexed with 1).

It is worth remarking that the rule $\rightarrow\mathcal{I}$ does the essential work of the Deduction Theorem in that it allows us to go from a proof of $\begin{matrix}\Gamma\\ [\varphi]\\ \vdots\\ \psi\end{matrix}$ to a proof of $\begin{matrix}\Gamma\\ \vdots\\ \varphi\rightarrow\psi\end{matrix}$. We can build on this observation through the use of the sequent notation.

2.4 From Axioms to Natural Deduction Rules

Heuristics are now given which, by considering each axiom in turn, translate a system presented with axioms into an equivalent Natural Deduction presentation.

The axioms A1 and A2, together with the rule of proof modus ponens, define the meaning of the implication connective through the transformation embodied in the deduction theorem (for which they are essential). The rule introducing an implication can therefore be viewed as an application of the deduction theorem. Similarly, the rule eliminating an implication can be seen to be given directly by modus ponens.

The remaining axioms can be combined with arbitrary occurrences of the modus ponens rule to derive their natural deduction, introduction and elimination rules. In cases where the application of modus ponens leaves an argument of the form '$\varphi\rightarrow\theta$', an occurrence of the rule $\rightarrow\mathcal{E}$ may be inserted to allow the new rule to discharge the assumption 'φ', should this be required.

Take axiom A3 $[\varphi\rightarrow(\psi\rightarrow\varphi\wedge\psi)]$ and two applications of Modus Ponens

$$\dfrac{\dfrac{\varphi \qquad \text{A3}}{\psi\rightarrow\varphi\wedge\psi} \qquad \psi}{\varphi\wedge\psi}$$

From which is derived the rule:

$$\dfrac{\varphi \quad \psi}{\varphi\wedge\psi}\,\wedge\mathcal{I}$$

Similarly for axiom A4a $[\varphi\wedge\psi\rightarrow\varphi]$ and axiom A4b $[\varphi\wedge\psi\rightarrow\psi]$

$$\dfrac{\varphi\wedge\psi \quad \text{A4a}}{\varphi} \qquad \dfrac{\varphi\wedge\psi \quad \text{A4b}}{\psi}$$

results in

$$\dfrac{\varphi\wedge\psi}{\varphi}\,\wedge\mathcal{E}a \qquad \text{and} \qquad \dfrac{\varphi\wedge\psi}{\psi}\,\wedge\mathcal{E}b$$

Also for axiom A5a $[\varphi\rightarrow\varphi\vee\psi]$ and axiom A5b $[\psi\rightarrow\varphi\vee\psi]$

$$\dfrac{\varphi \quad \text{A5a}}{\varphi\vee\psi} \qquad \dfrac{\psi \quad \text{A5b}}{\varphi\vee\psi}$$

arriving at $\vee\mathcal{I}a$ and $\vee\mathcal{I}b$

$$\frac{\varphi}{\varphi\vee\psi}\,\vee\mathcal{I}a \qquad \text{and} \qquad \frac{\psi}{\varphi\vee\psi}\,\vee\mathcal{I}b$$

Also interesting is axiom A6 $[(\varphi\rightarrow\psi)\rightarrow((\psi\rightarrow\theta)\rightarrow(\varphi\vee\psi\rightarrow\theta))]$. Applying modus ponens twice gives

$$\dfrac{\dfrac{\varphi\rightarrow\theta \qquad \text{A6}}{(\psi\rightarrow\theta)\rightarrow(\varphi\vee\psi\rightarrow\theta)} \qquad \psi\rightarrow\theta}{\varphi\vee\psi\rightarrow\theta}$$

Cutting this once more with $\varphi\vee\psi$ produces

$$\dfrac{\dfrac{\varphi\rightarrow\theta \quad \psi\rightarrow\theta}{\varphi\vee\psi\rightarrow\theta} \qquad \varphi\vee\psi}{\theta}$$

Applying $\rightarrow\mathcal{I}$ (derived above) to $\varphi\rightarrow\theta$ and $\psi\rightarrow\theta$ the following rule is obtained

$$\dfrac{\dfrac{\begin{array}{c}[\varphi]\\ \vdots\\ \theta\end{array}}{\varphi\rightarrow\theta} \qquad \dfrac{\begin{array}{c}[\psi]\\ \vdots\\ \theta\end{array}}{\psi\rightarrow\theta} \qquad \varphi\vee\psi}{\theta}$$

This may be simplified to the rule

$$\dfrac{\begin{array}{ccc}[\varphi] & [\psi] & \\ \vdots & \vdots & \\ \theta & \theta & \varphi\vee\psi\end{array}}{\theta}\,\vee\mathcal{E}$$

Applying a similar analysis to axiom A7 $[(\varphi\rightarrow\psi)\rightarrow((\varphi\rightarrow\neg\psi)\rightarrow\neg\varphi)]$

$$\dfrac{\dfrac{\varphi\rightarrow\psi \qquad \text{A7}}{(\varphi\rightarrow\neg\psi)\rightarrow\neg\varphi} \qquad \varphi\rightarrow\neg\psi}{\neg\varphi}$$

applying $\rightarrow\mathcal{I}$ twice gives

$$\dfrac{\begin{array}{cc}[\varphi] & [\varphi]\\ \vdots & \vdots\\ \psi & \neg\psi\end{array}}{\neg\varphi}\,\neg\mathcal{I}$$

The axiom A8 $[\neg\neg\varphi\rightarrow\varphi]$ is straightforward

$$\dfrac{\neg\neg\varphi \quad \text{A8}}{\varphi}$$

giving

$$\frac{\neg\neg\varphi}{\varphi}\,\neg\neg\mathcal{E}$$

Likewise axiom A9 $[\varphi\rightarrow(\neg\varphi\rightarrow\psi)]$ becomes

$$\frac{\varphi \quad \neg\varphi}{\psi}\,\neg\mathcal{E}$$

These rules are summarised in the following tables. Assumptions that may be discharged by a rule are indicated in square brackets.

Natural Deduction Rules for Classical Logic

Minimal Logic +

$$\frac{\varphi \quad \psi}{\varphi\wedge\psi}\,\wedge\mathcal{I} \qquad \frac{\varphi\wedge\psi}{\varphi}\,\wedge\mathcal{E}a \qquad \frac{\varphi\wedge\psi}{\psi}\,\wedge\mathcal{E}b$$

$$\frac{\varphi}{\varphi\vee\psi}\,\vee\mathcal{I}a \qquad \frac{\psi}{\varphi\vee\psi}\,\vee\mathcal{I}b \qquad \frac{\begin{array}{ccc}[\varphi] & [\psi] & \\ \vdots & \vdots & \\ \theta & \theta & \varphi\vee\psi\end{array}}{\theta}\,\vee\mathcal{E}$$

$$\frac{\begin{array}{cc}[\varphi] & [\varphi] \\ \vdots & \vdots \\ \psi & \neg\psi\end{array}}{\neg\varphi}\,\neg\mathcal{I} \qquad \frac{\neg\neg\varphi}{\varphi}\,\neg\neg\mathcal{E}$$

Natural Deduction Rules for Intuitionistic Logic

Classical Logic $-\ \neg\neg\mathcal{E}$ +

$$\frac{\varphi \quad \neg\varphi}{\psi}\,\neg\mathcal{E}$$

2.5 Introduction and Elimination Presentations

A disadvantage of the use of Natural Deduction is that its mechanism for the discharge of assumptions is global to the subproofs of a rule. However, the discharge of assumptions can be made local (and explicit) by the use of sequents. The locality arises from the rule's ability to indicate the discharge of assumptions by the

manipulation of sequents rather than subproofs. The use of sequents to reason about the assumptions used in an argument has already been discussed. Writing '$\Gamma \vdash \psi$' means that ψ follows from the assumptions Γ. Sequents represent arguments and sequent rules transform one or more arguments into another argument, preserving, it is hoped, the validity of the new argument with respect to the system concerned.

The natural deduction rules given above can be recast to use sequents. Recall that the notation $\Gamma \vdash \varphi$ is used to mean that the proof of φ depends on the assumptions Γ or as has sometimes been written: $\begin{array}{c}\Gamma\\ \vdots\\ \varphi.\end{array}$ Inference rules using sequents can now make the manipulation of assumptions local (no $\vdots$'s required); this is more convenient and importantly, more expressive, as it allows greater control to be exercised over mechanisms concerning how and when assumptions can be introduced or eliminated. Moreover, this can be done without adjusting the basic infrastructure as would be necessary if a Natural Deduction presentation were adopted. The localisation of assumptions also simplifies data structures used in the environment.

2.6 From Natural Deduction to Sequents

Armed with the Deduction Theorem, the Natural Deduction rules may be recast as follows. A proof of φ in the Natural Deduction setting becomes a proof of the sequent $\Gamma \vdash \varphi$ by collecting together all the assumptions on which the proof of φ rests and calling them Γ. For example

$$\dfrac{\begin{array}{ccc}[\varphi] & [\psi] & \\ \vdots & \vdots & \\ \theta & \theta & \varphi \vee \psi\end{array}}{\theta}\ \vee\mathcal{E} \qquad \text{becomes} \qquad \dfrac{\Gamma, \varphi \vdash \theta \quad \Delta, \psi \vdash \theta \quad \Theta \vdash \varphi \vee \psi}{\Gamma, \Delta, \Theta \vdash \theta}\ \vee\vdash\mathcal{E}$$

Nomenclature: A sequent style introduction (elimination) rule for the connective x is called '$x\vdash\mathcal{I}$' ('$x\vdash\mathcal{E}$').

An assumption can be introduced in a natural deduction proof and then used immediately:

$$\dfrac{\dfrac{[\varphi]^1}{\varphi}}{\varphi \rightarrow \varphi^1}$$

which can be captured by the basic sequent

$$\dfrac{\dfrac{}{\varphi \vdash \varphi}\ \textit{basic}}{\vdash \varphi \rightarrow \varphi}$$

Returning to the illustration of the form of the proof tree required for this formalism, it can be seen that a proof tree in this sequent framework has leaves consisting of sequents with varying structures, whose details are given below:

$$\begin{aligned}
\textit{Sequent-proof-tree}(\gamma) &::= \textit{Tree}(\gamma,\ \textit{Sequent-rule}(\gamma)) \\
\textit{Sequent-rule}(\gamma) &::= \textit{Sequent-proof-tree}(\gamma)\ \textit{list} \rightarrow \gamma \\
\\
\textit{Classical-proof-tree} &::= \textit{Sequent-proof-tree}(\textit{Formula set}\ \#\ \textit{Formula set}) \\
\textit{Intuitionistic-proof-tree} &::= \textit{Sequent-proof-tree}(\textit{Formula set}\ \#\ \textit{Formula}) \\
\textit{Linear-proof-tree} &::= \textit{Sequent-proof-tree}(\textit{Formula list}\ \#\ \textit{Formula list})
\end{aligned}$$

The internal nodes are again applications of the sequent rules of inference which rewrite the proofs above them into a sequent representing the conclusion of the proof. The definition of *Sequent-proof-tree* and *Sequent-rule* is parameterised by an appropriate choice of the sequent structure.

A Natural Deduction rule derived from the axiomatic basis considered so far allows an inference rule to discharge any number of occurrences of the assumption simultaneously. Consequently, it is permissible, in the new sequent setting, to make as many copies of the assumptions as are required; all the copies can be discharged in a single step. This observation allows all the assumptions to be shared ($\Gamma \cup \Delta \cup \Theta$) among all the sub-proofs of the sequent form of the rule. When this is done, the following slightly simplified presentation of the rule can be obtained for $\vee\mathcal{E}$ above

$$\dfrac{\Gamma, \varphi \vdash \theta \quad \Gamma, \psi \vdash \theta \quad \Gamma \vdash \varphi \vee \psi}{\Gamma \vdash \theta}$$

If assumptions are shared among sub-proofs in this way, they must be introduced into the branch of the proof at some stage. This can be at the leaves or inside the body of the proof. For the first alternative, it is necessary to adjust the definition of a basic sequent to be

$$\dfrac{}{\varphi, \Gamma \vdash \varphi}\ \textit{basic}$$

rather than

$$\dfrac{}{\varphi \vdash \varphi}$$

as our means of introducing assumptions. The other alternative, which is more in keeping with the spirit of the use of assumptions in the natural deduction setting, is

to keep the basic sequent, but to add a rule specifically to allow the introduction of assumptions into the body of the derivation:

$$\frac{\Gamma \vdash \varphi}{\Gamma, \psi \vdash \varphi}\ thin\vdash$$

From a theorem proving viewpoint, in which rules are used to decompose a goal to a number of subgoals, it is better to adopt the first strategy over the second as it is harder to decide when an assumption is no longer required for the proof of the judgement.

2.6.1 Structural Rules

The use of structural rules in the sequent calculus may further be motivated by re-examining the earlier discussion of the Deduction Theorem[3]. In the earlier section it was seen that to discharge an assumption using the particular axiomatisation of classical logic it had to be shown that:

1. each axiom of the system could be transformed from ψ to $\varphi \rightarrow \psi$, for an arbitrary φ, and
2. each rule could be transformed from $\dfrac{\psi_1 \cdots \psi_n}{\theta}$ to $\dfrac{\varphi \rightarrow \psi_1 \quad \cdots \quad \varphi \rightarrow \psi_n}{\varphi \rightarrow \theta}$.

Consider however, the following presentation of *linear implication* in [Gab89].

Axioms for Linear Implication

L1 $\varphi \rightarrow \varphi$
L2 $(\varphi \rightarrow (\psi \rightarrow \theta)) \rightarrow (\psi \rightarrow (\varphi \rightarrow \theta))$
L3 $(\varphi \rightarrow \psi) \rightarrow ((\theta \rightarrow \varphi) \rightarrow (\theta \rightarrow \psi))$
L4 $(\varphi \rightarrow \psi) \rightarrow ((\psi \rightarrow \theta) \rightarrow (\varphi \rightarrow \theta))$

For linear implication, the intention is that an assumption must be used *precisely once.*

Axioms for Relevant Implication

Linear Implication +
R1 $(\varphi \rightarrow (\psi \rightarrow \theta)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \theta))$

[3] Although this discussion relates to non-standard logics of the next chapter

For relevant implication, the intention is that an assumption *cannot be discharged unless it is used*, but it can be used repeatedly as permitted by axiom R1. Hence, neither system allows weakening, or monotonicity as it was called before.

To see how these axioms give rise to a Deduction Theorem the conditions given above are recast slightly:

1. the assumption being discharged can be transformed from ψ to $\varphi \rightarrow \psi$, for an arbitrary φ, and
2. that each rule could be transformed from $\dfrac{\psi_1 \cdots \psi_n}{\theta}$ to

$$\frac{\psi_1 \quad \cdots \quad \varphi \rightarrow \psi_i \quad \cdots \quad \psi_n}{\varphi \rightarrow \theta}$$

for precisely one arbitrary i for the case of *linear implication*, and to

$$\frac{\varphi \rightarrow \psi_1 \quad \cdots \quad \varphi \rightarrow \psi_n}{\varphi \rightarrow \theta}$$

for an arbitrary number of antecedents in the case of *relevant implication.*

There is only a single rule of proof, Modus Ponens, for the Linear and Relevant systems. Condition 1 is satisfied by an appropriate instantiation of axiom L1. For condition 2 there are three subcases:

$$\frac{\varphi \rightarrow \psi \quad \psi \rightarrow \theta}{\varphi \rightarrow \theta}\,(a) \qquad \frac{\psi \quad \varphi \rightarrow (\psi \rightarrow \theta)}{\varphi \rightarrow \theta}\,(b) \qquad \frac{\varphi \rightarrow \psi \quad \varphi \rightarrow (\psi \rightarrow \theta)}{\varphi \rightarrow \theta}\,(c)$$

of these, the last is not valid for linear implication as it allows φ to be used twice (axiom R1 is required). Each has the following justification:

$$\frac{\dfrac{\psi \rightarrow \theta \quad \text{axiom L3}}{(\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \theta)} \qquad \varphi \rightarrow \psi}{\varphi \rightarrow \theta}\,(a)$$

$$\frac{\psi \qquad \dfrac{\varphi \rightarrow (\psi \rightarrow \theta) \quad \text{axiom L2}}{\psi \rightarrow (\varphi \rightarrow \theta)}}{\varphi \rightarrow \theta}\,(b)$$

$$\frac{\dfrac{\varphi \rightarrow \psi \qquad \text{axiom L4}}{(\psi \rightarrow (\varphi \rightarrow \theta)) \rightarrow (\varphi \rightarrow (\varphi \rightarrow \theta))} \qquad \dfrac{\varphi \rightarrow (\psi \rightarrow \theta) \quad \text{axiom L2}}{\psi \rightarrow (\varphi \rightarrow \theta)}}{\dfrac{\varphi \rightarrow (\varphi \rightarrow \theta) \qquad\qquad\qquad\qquad\qquad\qquad \text{axiom R1}}{\varphi \rightarrow \theta}\,(c)}$$

The restrictions Linear and Relevant logics place on the use of assumptions is translated directly in the sequent calculus to conditions on the manipulation of assumptions:

Exchange	$\dfrac{\Gamma,\varphi,\psi,\Gamma'\vdash\Delta}{\Gamma,\psi,\varphi,\Gamma'\vdash\Delta}$	$\dfrac{\Gamma\vdash\Delta,\varphi,\psi,\Delta'}{\Gamma\vdash\Delta,\psi,\varphi,\Delta'}$	axiom A2	axioms L2–4
Contraction	$\dfrac{\Gamma,\varphi,\varphi,\Gamma'\vdash\Delta}{\Gamma,\varphi,\Gamma'\vdash\Delta}$	$\dfrac{\Gamma\vdash\Delta,\varphi,\varphi,\Delta'}{\Gamma\vdash\Delta,\varphi,\Delta'}$	axiom A2	axiom R1
Weakening	$\dfrac{\Gamma\vdash\Delta}{\varphi,\Gamma\vdash\Delta}$	$\dfrac{\Gamma\vdash\Delta}{\Gamma\vdash\varphi,\Delta}$	axiom A1	

So the choice of structural rule depends on the logic.

Weakening	*Contraction*	*Exchange*	*systems*
√	√	√	Most logics
×	√	√	Relevance logic
×	×	√	Linear logic
×	×	×	Type inference

This is an important observation, as it gives greater expressibility of sequent presentations over the number of families of logics that can be supported without resorting to additional (global) controls on the manipulation of assumptions.

There are two approaches to the treatment of weakening in the cases of Classical and Intuitionistic logics:

1. At each rule, the assumptions may be rearranged to match the assumptions each sub-proof requires (*e.g.* Γ,Δ,Θ above). This is preferable for top-down proofs or forward reasoning.

2. The assumptions used by each sub-proof may be made uniform by sharing all the assumptions amongst each sub-proof. In general this leads to redundant assumptions. These assumptions are harmless and can be absorbed by generalising the notion of *basic sequent.* This is preferable for bottom-up proofs or backwards reasoning.

The three tables below show the application of the second alternative to the Natural Deduction rules above.

Introduction & Elimination Rules for Minimal Logic

$$\frac{}{\Gamma,\varphi\vdash\varphi}\ \textit{basic} \qquad \frac{\Gamma,\varphi\vdash\psi}{\Gamma\vdash\varphi\to\psi}\ \to\vdash\mathcal{I} \qquad \frac{\Gamma\vdash\varphi\quad\Gamma\vdash\varphi\to\psi}{\Gamma\vdash\psi}\ \to\vdash\mathcal{E}$$

Introduction & Elimination Rules for Classical Logic

Minimal Logic +

$$\frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\ \wedge\vdash\mathcal{I} \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\ \wedge\vdash\mathcal{E}a \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}\ \wedge\vdash\mathcal{E}b$$

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}\ \vee\vdash\mathcal{I}a \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}\ \vee\vdash\mathcal{I}b \qquad \frac{\Gamma, \varphi \vdash \theta \quad \Gamma, \psi \vdash \theta \quad \Gamma \vdash \varphi \vee \psi}{\Gamma \vdash \theta}\ \vee\vdash\mathcal{E}$$

$$\frac{\Gamma, \varphi \vdash \psi \quad \Gamma, \varphi \vdash \neg\psi}{\Gamma \vdash \neg\varphi}\ \neg\vdash\mathcal{I} \qquad \frac{\Gamma \vdash \neg\neg\varphi}{\Gamma \vdash \varphi}\ \neg\neg\vdash\mathcal{E}$$

Introduction & Elimination Rules for Intuitionistic Logic

Classical Logic $- \neg\neg\vdash\mathcal{E} +$

$$\frac{\Gamma \vdash \varphi \quad \Gamma \vdash \neg\varphi}{\Gamma \vdash \psi}\ \neg\vdash\mathcal{E}$$

2.7 All-introduction Sequent Rules

Although the sequent versions of the introduction and elimination rules presented above clarify the discharge of assumptions, they do not completely predict the structure of the proof. What is required are rules that can be used in the backwards, or bottom-to-top, sense by the computer theorem prover. That is, in order to decide whether some judgement (here sequent) holds, the rules decompose the judgement into a number of *smaller* judgements. These judgements may themselves be (recursively) decomposed until they are small enough to be trivial (or basic). Fortunately, the sequent introduction and elimination rules presented in the previous section may be transformed into a form suitable for this purpose. The subformula property plays an important role in providing guidance through this process.

Nomenclature: A rule introducing the connective x on the left-hand (right-hand) side of a turnstile is named '$x\vdash_{\mathsf{S}}$' ('$\vdash_{\mathsf{S}} x$'), where S is the name of the system to which the rule belongs. When it is clear from the context which S is being used, the subscript is omitted.

2.7.1 Subformulas and the Subformula Property

The subformulas of a formula are given by $SF(\varphi)$, defined as follows:

$$\begin{aligned} SF(\varphi) &= \{\varphi\}, \text{ if } \varphi \in \Phi \\ SF(\neg\varphi) &= \{\neg\varphi\} \cup SF(\varphi) \\ SF(\varphi \wedge \psi) &= \{\varphi \wedge \psi\} \cup SF(\varphi) \cup SF(\psi) \\ SF(\varphi \vee \psi) &= \{\varphi \vee \psi\} \cup SF(\varphi) \cup SF(\psi) \\ SF(\varphi \rightarrow \psi) &= \{\varphi \rightarrow \psi\} \cup SF(\varphi) \cup SF(\psi) \end{aligned}$$

where Φ denotes the set of atomic propositions. The definition is lifted to single-conclusioned sequents $\psi_1, \ldots, \psi_n \vdash \varphi$ as follows

$$SF(\psi_1, \ldots, \psi_n \vdash \varphi) = SF(\varphi) \cup \left[\bigcup_{i=1}^{n} SF(\psi_i)\right]$$

A similar lifting is made for multiply-conclusioned sequents:

$$SF(\psi_1, \ldots, \psi_n \vdash \varphi_1, \ldots, \varphi_m) = \left[\bigcup_{i=1}^{n} SF(\psi_i)\right] \cup \left[\bigcup_{i=1}^{m} SF(\varphi_i)\right]$$

A general formulation of the subformula property is given in §5.2.13, but the definition here is sufficient for the present.

An inference rule is said to enjoy the subformula property when all formulas occurring in sequents above the horizontal line belong to the set of subformulas of the sequent occurring in the conclusion of the rule below the line. That is the rule

$$\frac{\Gamma_1 \vdash \psi_1 \quad \cdots \quad \Gamma_m \vdash \psi_m}{\Gamma \vdash \varphi}$$

has the subformula property when $SF(\Gamma \vdash \varphi) \supseteq \bigcup_{i=1}^{m} SF(\Gamma_i \vdash \psi_i)$

2.8 From Sequents to All-introduction Rules

Reviewing the introduction and elimination rules above, it can be seen that all but one of the *introduction* rules already enjoy the subformula property but not all of the *elimination* rules do. Taking the elimination rule $\vee$-E as an example of one such rule

$$\frac{\Gamma,\varphi\vdash\theta \quad \Gamma,\psi\vdash\theta \quad \Gamma\vdash\varphi\vee\psi}{\Gamma\vdash\theta}\ \vee\vdash\mathcal{E}$$

we note immediately that this rule does not have the subformula property since, for example, the conclusion of $\Gamma\vdash\varphi\vee\psi$ is syntactically separate from the conclusion of the rule $\Gamma\vdash\theta$.

But this rule may be reformulated in the following way: Assume that the assumptions are $\Gamma,\varphi\vee\psi$ rather than Γ. This can be done by weakening (using the rule '*thin*$\vdash$' above); the rule then becomes:

$$\frac{\dfrac{\Gamma,\varphi\vdash\theta}{\Gamma,\varphi\vee\psi,\varphi\vdash\theta}\ thin\vdash \qquad \dfrac{\Gamma,\psi\vdash\theta}{\Gamma,\varphi\vee\psi,\psi\vdash\theta}\ thin\vdash \qquad \dfrac{\Gamma\vdash\varphi\vee\psi}{\Gamma,\varphi\vee\psi\vdash\varphi\vee\psi}\ thin\vdash}{\Gamma,\varphi\vee\psi\vdash\theta}$$

Since '$\Gamma,\varphi\vee\psi\vdash\varphi\vee\psi$' is a basic sequent, the original proof of $\Gamma\vdash\varphi\vee\psi$ may be ignored completely. Instead of showing that '$\varphi\vee\psi$' was a consequence of the assumptions Γ the new rule insists that this consequence is already explicitly an assumption. This is precisely what is required for a more directed search strategy. The new rule simplifies to:

$$\frac{\Gamma,\varphi\vdash\theta \quad \Gamma,\psi\vdash\theta}{\Gamma,\varphi\vee\psi\vdash\theta}\ \vee\vdash$$

This rule now enjoys the subformula property as required.

2.8.1 The cut rule

The *cut* rule is derived from $\rightarrow\vdash\mathcal{E}$ by an application of the $\rightarrow\vdash\mathcal{I}$ rule as follows:

$$\frac{\Gamma\vdash\varphi \qquad \dfrac{\Gamma,\varphi\vdash\psi}{\Gamma\vdash\varphi\rightarrow\psi}\ \rightarrow\vdash\mathcal{I}}{\Gamma\vdash\psi}\ \rightarrow\vdash\mathcal{E}$$

giving

$$\frac{\Gamma\vdash\varphi \quad \Gamma,\varphi\vdash\psi}{\Gamma\vdash\psi}\ cut$$

2.8.2 Generalising rules

In most cases the technique described above produces suitable all-introduction sequent rules, as it is here for the rules $\vee\vdash$ and $\wedge\vdash$. Occassionally however, it is necessary to generalise a rule slightly in order to make it more widely applicable. This amounts to building in one or more applications of the cut rule. Taking $\rightarrow\vdash\mathcal{E}$ as an example of this

$$\frac{\Gamma\vdash\varphi \quad \Gamma\vdash\varphi\rightarrow\psi}{\Gamma\vdash\psi}\rightarrow\vdash\mathcal{E}$$

Weakening both sides with $\varphi\rightarrow\psi$ produces

$$\frac{\dfrac{\Gamma\vdash\varphi}{\Gamma,\varphi\rightarrow\psi\vdash\varphi}\,thin\vdash \qquad \dfrac{\Gamma\vdash\varphi\rightarrow\psi}{\Gamma,\varphi\rightarrow\psi\vdash\varphi\rightarrow\psi}\,thin\vdash}{\Gamma,\varphi\rightarrow\psi\vdash\psi}$$

and hence

$$\frac{\Gamma\vdash\varphi}{\Gamma,\varphi\rightarrow\psi\vdash\psi}\,bad\rightarrow\vdash$$

but this rule is not sufficiently general as $SF(\varphi\rightarrow\psi)\cap SF(\psi)\neq\varnothing$ and consequently it can only be used in a restricted number of circumstances. For example the derivation of axiom A2 quickly becomes stuck:

$$\cfrac{\cfrac{\cfrac{\cfrac{\text{stuck!}}{\varphi,\varphi\rightarrow(\psi\rightarrow\theta),\varphi\rightarrow\psi\vdash\theta}\,?}{\varphi\rightarrow(\psi\rightarrow\theta),\varphi\rightarrow\psi\vdash\varphi\rightarrow\theta}}{\varphi\rightarrow\psi\vdash(\varphi\rightarrow(\psi\rightarrow\theta))\rightarrow(\varphi\rightarrow\theta)}}{\vdash(\varphi\rightarrow\psi)\rightarrow((\varphi\rightarrow(\psi\rightarrow\theta))\rightarrow(\varphi\rightarrow\theta))}$$

and cannot proceed using $bad\rightarrow\vdash$ without resorting to the cut rule. The objective is to avoid the cut rule as it introduces an element of indirection in the proof (and it does not enjoy the subformula property). By incorporating an instance of the cut rule in the rule $bad\rightarrow\vdash$ a rule can be obtained that avoids this restriction of overlapping subformulas and still gives a syntactically determined path through the proof. The result is the more general rule:

$$\frac{\dfrac{\Gamma\vdash\varphi}{\Gamma,\varphi\rightarrow\psi\vdash\psi} \quad \Gamma,\psi\vdash\theta}{\Gamma,\varphi\rightarrow\psi\vdash\theta}$$

which simplifies to

$$\frac{\Gamma\vdash\varphi \quad \Gamma,\psi\vdash\theta}{\Gamma,\varphi\rightarrow\psi\vdash\theta}\rightarrow\vdash$$

This again enjoys the sub-formula property as required. It is easy to see that this rule specialises to $bad\rightarrow\vdash$ when $\psi=\theta$. Notice that the application of the cut rule

did not introduce a spurious cut-formula into the rule. 'Spurious' means a formula which is not a sub-formula of the consequent of the rule. The proof of axiom A2 above may be continued using $\vdash\!\rightarrow$ as follows:

$$\dfrac{\dfrac{\dfrac{\psi \vdash \psi \quad \theta \vdash \theta}{\psi, \psi\rightarrow\theta \vdash \theta}\,{\rightarrow}\vdash}{\psi, \varphi\rightarrow(\psi\rightarrow\theta), \varphi \vdash \theta}\,{\rightarrow}\vdash}{\text{stuck!}}\,{\rightarrow}\vdash$$

2.8.3 Combining Rules

In some cases, several of our original elimination rules may be combined to get a single simpler rule that captures the effect of applying both of the originals simultaneously. Here, $\wedge\vdash\mathcal{E}a$ and $\wedge\vdash\mathcal{E}b$ are used as our example:

$$\dfrac{\Gamma \vdash \varphi\wedge\psi}{\Gamma \vdash \varphi}\,\wedge\vdash\mathcal{E}a \quad \text{and} \quad \dfrac{\Gamma \vdash \varphi\wedge\psi}{\Gamma \vdash \psi}\,\wedge\vdash\mathcal{E}b$$

and by the following argument

$$\dfrac{\dfrac{\dfrac{\Gamma, \varphi\wedge\psi \vdash \varphi\wedge\psi}{\Gamma, \varphi\wedge\psi \vdash \varphi}\,\wedge\vdash\mathcal{E}a \qquad \dfrac{\Gamma, \varphi\wedge\psi \vdash \varphi\wedge\psi}{\Gamma, \varphi\wedge\psi \vdash \psi}\,\wedge\vdash\mathcal{E}b \qquad \Gamma, \varphi, \psi \vdash \theta}{\Gamma, \varphi, \varphi\wedge\psi \vdash \theta}\,cut}{\Gamma, \varphi\wedge\psi \vdash \theta}\,cut$$

we obtain

$$\dfrac{\Gamma, \varphi, \psi \vdash \theta}{\Gamma, \varphi\wedge\psi \vdash \theta}\,\wedge\vdash$$

This is really just for economy as it was possible to stay with two separate rules:

$$\dfrac{\Gamma, \varphi\wedge\psi, \varphi \vdash \theta}{\Gamma, \varphi\wedge\psi \vdash \theta} \quad \text{and} \quad \dfrac{\Gamma, \varphi\wedge\psi, \psi \vdash \theta}{\Gamma, \varphi\wedge\psi \vdash \theta}$$

which has to keep $\varphi\wedge\psi$ as an assumption in case ψ (resp. φ) was required later in the proof. The rule $\wedge\vdash$ allows formula $\varphi\wedge\psi$ to be discarded once and for all during the proof of $\Gamma, \varphi\wedge\psi \vdash \theta$. The result is a shorter proof.

If there are sets of formulas on both sides of the turnstile ($\vdash$), a similar argument may be given for $\vee$ using $\vee\vdash\mathcal{I}a$ and $\vee\vdash\mathcal{I}b$:

$$\dfrac{\dfrac{\Gamma \vdash \varphi, \psi, \Delta \quad \dfrac{\Gamma, \psi \vdash \psi, \Delta}{\Gamma, \psi \vdash \varphi\vee\psi, \Delta}\,\vee\vdash\mathcal{I}b}{\Gamma \vdash \varphi, \varphi\vee\psi, \Delta}\,cut \qquad \dfrac{\Gamma, \varphi \vdash \varphi, \Delta}{\Gamma, \varphi \vdash \varphi\vee\psi, \Delta}\,\vee\vdash\mathcal{I}a}{\Gamma \vdash \varphi\vee\psi, \Delta}\,cut$$

but this rule does not hold for Intuitionistic Logic, as will be shown below.

The rule $\rightarrow\vdash\mathcal{I}$ already enjoys the sub-formula property as does the rule '*basic*'. Combining these with $\rightarrow\vdash$ above produces the table of rules for minimal logic below:

All-introduction Rules for Minimal Logic

$$\frac{}{\Gamma,\varphi\vdash\varphi}\ \textit{basic} \qquad \frac{\Gamma\vdash\varphi \quad \Gamma,\psi\vdash\theta}{\Gamma,\varphi\rightarrow\psi\vdash\theta}\ \rightarrow\vdash \qquad \frac{\Gamma,\varphi\vdash\psi}{\Gamma\vdash\varphi\rightarrow\psi}\ \vdash\rightarrow$$

2.8.4 Negation

As they presently stand, the introduction and elimination rules for negation ($\neg\vdash\mathcal{I}$, $\neg\neg\vdash\mathcal{E}$, and $\neg\vdash\mathcal{E}$) do not have the sub-formula property.

$$\frac{\Gamma,\varphi\vdash\psi \quad \Gamma,\varphi\vdash\neg\psi}{\Gamma\vdash\neg\varphi}\,\neg\vdash\mathcal{I} \qquad \frac{\Gamma\vdash\varphi \quad \Gamma\vdash\neg\varphi}{\Gamma\vdash\psi}\,\neg\vdash\mathcal{E} \qquad \frac{\Gamma\vdash\neg\neg\varphi}{\Gamma\vdash\varphi}\,\neg\neg\vdash\mathcal{E}$$

It is necessary to find satisfactory all-introduction forms of these rules: the first and second for Intuitionistic logic and the first and third for Classical logic.

A satisfactory form of $\neg\vdash\mathcal{E}$ may be obtained by making $\neg\varphi$ an immediately available assumption:

$$\frac{\dfrac{\Gamma\vdash\varphi}{\Gamma,\neg\vdash\varphi}\,thin\vdash \qquad \dfrac{\Gamma\vdash\neg\varphi}{\Gamma,\neg\varphi\vdash\neg\varphi}\,thin\vdash}{\Gamma,\neg\varphi\vdash\psi}\,\neg\vdash\mathcal{E}$$

This simplifies to the derived rule:

$$\frac{\Gamma\vdash\varphi}{\Gamma,\neg\varphi\vdash\psi}\,\neg\vdash$$

The rule $\neg\vdash\mathcal{I}$ derives $\neg\varphi$ on the basis that assuming φ leads to a contradiction (both ψ and $\neg\psi$ are derivable). So, letting $\bot$ stand for the contradictory formula '$\psi\wedge\neg\psi$', the rule:

$$\frac{\Gamma,\varphi\vdash\bot}{\Gamma\vdash\neg\varphi}\,\vdash\neg$$

can be seen to derive $\neg\vdash\mathcal{I}$ as follows:

$$\cfrac{\cfrac{\cfrac{\Gamma,\varphi\vdash\psi \quad \Gamma,\varphi\vdash\neg\psi}{\Gamma,\varphi\vdash\psi\wedge\neg\psi}\,\vdash\wedge}{\Gamma,\varphi\vdash\bot}\,defn.}{\Gamma\vdash\neg\varphi}\,\vdash\neg$$

However the rule $\vdash\neg$ does not satisfy the sub-formula property as it was stated earlier, since $\bot$ is not a sub-formula of the conclusion. Since the sequent '$\Gamma\vdash\bot$' has the same interpretation as '$\Gamma\vdash$', the definition of $SF(\varphi)$ is extended to include $\bot$

$$SF'(\varphi) = \{\bot\} \cup SF(\varphi)$$

With $\neg\vdash$ and $\vdash\neg$ there are sufficient rules to cover the axiomatic presentation of Intuitionistic logic. The rules for the Intuitionistic fragment are summarised below:

All-introduction Rules for Intuitionistic Logic

Minimal Logic +

$$\frac{\Gamma, \varphi, \psi \vdash \theta}{\Gamma, \varphi \wedge \psi \vdash \theta} \wedge\vdash_{\mathsf{I}} \qquad \frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi} \vdash_{\mathsf{I}} \wedge$$

$$\frac{\Gamma, \varphi \vdash \theta \quad \Gamma, \psi \vdash \theta}{\Gamma, \varphi \vee \psi \vdash \theta} \vee\vdash_{\mathsf{I}} \qquad \frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi} \vdash_{\mathsf{I}} \vee a \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi} \vdash_{\mathsf{I}} \vee b$$

$$\frac{\Gamma \vdash \varphi}{\Gamma, \neg\varphi \vdash \psi} \neg\vdash_{\mathsf{I}} \qquad \frac{\Gamma, \varphi \vdash \bot}{\Gamma \vdash \neg\varphi} \vdash_{\mathsf{I}} \neg$$

There is still a need for an explanation for the classical axiom A8 and its corresponding rule $\neg\neg\vdash\mathcal{E}$ which replace the intuitionistic axiom A9 and rule $\neg\vdash\mathcal{E}$. Without this rule, the derivation of the classically valid formula $\varphi \vee \neg\varphi$ is not possible using the Intuitionistic rules alone, *e.g.*

$$\dfrac{\dfrac{\text{stuck!}}{\vdash \varphi}}{\vdash \varphi \vee \neg\varphi} \qquad \dfrac{\dfrac{\dfrac{\text{stuck!}}{\varphi \vdash \bot}}{\vdash \neg\varphi}}{\vdash \varphi \vee \neg\varphi}$$

The problem arises because the rules for $\vee$ introduction on the right ($\vdash_{\mathsf{I}} \vee a$ and $\vdash_{\mathsf{I}} \vee b$) force us to decide which of φ or $\neg\varphi$ is provable. Once the decision has been taken, there is no opportunity to revise it. In contrast, the effect of $\neg\neg\vdash\mathcal{E}$, in combination with the other rule for $\neg\vdash\mathcal{I}$, in a classical presentation is to blur this clear decision, by allowing the use of whichever of the disjuncts is convenient. Therefore, a mechanism is needed that can record the different choices that are available. One way of doing this is to use the *restart* rule.

2.8.5 Restart Rule

The restart rule permits a proof to be restarted when it gets stuck with the original sequent and any assumptions accumulated earlier in the proof. The restart rule requires an extension to the linguistic structures given at the beginning of the chapter. The category of formulas is extended with a new operator '$[\Gamma \vdash \varphi]$', and rules are provided for introducing and eliminating the restart rule. The former is used to remember the initial sequent, and make it available for "restarting" later in a derivation. The restart rule is due to Gabbay[Gab91].

> "If p is a proposition and e and f are formulas and Γ is a set of formulas then p, $\neg e$, $e \wedge f$, $e \vee f$, $e \rightarrow f$, $[\Gamma \vdash e]$ are formulas. Nothing else is a formula."

Initial sequents of the form $\Gamma \vdash \varphi$ are transformed to $\Gamma, [\Gamma \vdash \varphi] \vdash \varphi$ by the rule

$$\frac{\Gamma, [\Gamma \vdash \varphi] \vdash \varphi}{\Gamma \vdash \varphi}\ \textit{start}$$

The *start* rule does not have the subformula property and its use is constrained by an appropriate 'once' segment in a heuristic, see §7.6.1. The rule *restart* provides a means of returning to the original sequent if necessary:

$$\frac{\Delta, \Gamma, [\Gamma \vdash \varphi] \vdash \varphi}{\Delta, [\Gamma \vdash \varphi] \vdash \bot}\ \textit{restart}$$

Using these two rules it is now possible to derive $\varphi \vee \neg\varphi$ as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\varphi \vdash \varphi}{\varphi \vdash \varphi \vee \neg\varphi}\vdash_{\mathrm{I}} \vee a}{\varphi, [\vdash \varphi \vee \neg\varphi] \vdash \bot}\ \textit{restart}}{[\vdash \varphi \vee \neg\varphi] \vdash \neg\varphi}\vdash_{\mathrm{I}} \neg}{[\vdash \varphi \vee \neg\varphi] \vdash \varphi \vee \neg\varphi}\vdash_{\mathrm{I}} \vee b}{\vdash \varphi \vee \neg\varphi}\ \textit{start}$$

In this proof, the lower part uses the rule $\vdash_{\mathrm{I}} \vee b$ to conclude $\neg\varphi$. The proof then becomes 'stuck' and the *restart* rule is used. The upper part of the derivation then uses the *other* rule $\vdash_{\mathrm{I}} \vee a$, to make a different choice, and now derives the basic sequent. It is because the restart rule allows the theorem prover to make several attempts at the derivation, while accumulating their assumptions, that classical logic can be obtained from the intuitionistic rules. But the restart rule gives lengthy proofs. A more elegant way of achieving the same end is to use sets of formulas on both sides of the sequent.

2.8.6 Symmetric Structure for Sequents

The rules for classical logic can be arrived at when it is noticed that Classical Logic admits a more general form of sequent: $\Gamma \vdash \Delta$ where Γ and Δ are both sets of formulas. The intended interpretation of the sequent is that when all of the members of Γ are true then at least one of the members of Δ is true. That is $\varphi_1, \ldots, \varphi_n \vdash \psi_1, \ldots, \psi_m$ iff $(\varphi_1 \wedge \cdots \wedge \varphi_n) \rightarrow (\psi_1 \vee \cdots \vee \psi_m)$. A rather computational view of this is that the right-hand side can be thought of as providing a storage area for the alternatives. Compare this with the restart rule which provides a means of making a different set of choices but of keeping intermediate results (assumptions).

It will sometimes be convenient to regard the multiple conclusioned sequent introduced above as a single conclusioned sequent. This may be done by writing all but a single occurrence of the right-hand formulas as negated formulas on the left side. This can only be done for systems in which have negation.

$$\Gamma \vdash \psi, \Delta \longrightarrow (\varphi_1 \wedge \cdots \wedge \varphi_n) \rightarrow (\psi \vee \psi_1 \vee \cdots \vee \psi_m)$$

$$\Big\downarrow \text{ rearrangement}$$

$$(\varphi_1 \wedge \cdots \wedge \varphi_n \wedge \neg\psi_1 \wedge \cdots \wedge \neg\psi_m) \rightarrow \psi \longrightarrow \Gamma, \neg\Delta \vdash \psi$$

All-introduction Rules for Classical Logic

$$\frac{}{\Gamma, \varphi \vdash \varphi, \Delta}\ basic$$

$$\frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \wedge \psi \vdash \Delta}\ \wedge\vdash_C \qquad \frac{\Gamma \vdash \varphi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta}\ \vdash_C \wedge$$

$$\frac{\Gamma, \varphi \vdash \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vee \psi \vdash \Delta}\ \vee\vdash_C \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \varphi \vee \psi, \Delta}\ \vdash_C \vee$$

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \rightarrow \psi \vdash \Delta}\ \rightarrow\vdash_C \qquad \frac{\Gamma, \varphi \vdash \psi, \Delta}{\Gamma \vdash \varphi \rightarrow \psi, \Delta}\ \vdash_C \rightarrow$$

$$\frac{\Gamma \vdash \varphi, \Delta}{\Gamma, \neg\varphi \vdash \Delta}\ \neg\vdash_C \qquad \frac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \neg\varphi, \Delta}\ \vdash_C \neg$$

2.9 Equivalence of Axiomatic and Sequent Presentations

In the above case studies, Minimal Logic was extended by adding axioms to generate axiomatic presentations of Intuitionistic and Classical logics. It was shown how these presentations could be recast in the Natural Deduction style, using introduction and elimination rules. The Natural Deduction style could then be converted to sequents, and hence a suitable all-introduction sequent presentation could be derived.

However, the question remains as to whether the above translation actually does capture the system.

To verify this the equivalence of the two presentations must be proven: axioms with Modus Ponens, and the all-introduction presentation using sequents.

2.9.1 Axiomatic implies Sequent

Assuming that there is an axiomatic proof of $\Gamma \vdash \psi$; it can be shown that this proof can be translated into a sequent proof of the same judgement. The proof proceeds by induction on the structure of the axiomatic proof.

For the base cases, transform the leaves as follows: If the leaf, θ, is an instance of an axiom then take a sequent proof of the axiom and instantiate it appropriately. Alternatively if θ is an assumption, replace it with its corresponding basic sequent $\theta \vdash \theta$.

For the inductive case, transform the internal nodes as follows: If the rule is an instance of Modus Ponens

$$\frac{\varphi \quad \varphi \rightarrow \psi}{\psi}$$

then by the induction hypothesis it is safe to assume that the translations of φ and $\varphi \rightarrow \psi$ are $\Delta \vdash \varphi$ and $\Theta \vdash \varphi \rightarrow \psi$ respectively. Let $\Gamma = \Delta \cup \Theta$ and combine them as follows

$$\dfrac{\dfrac{\dfrac{\Delta \vdash \varphi}{\Gamma \vdash \varphi}\,thinning\vdash* \qquad \Gamma, \psi \vdash \psi}{\Gamma, \varphi \rightarrow \psi \vdash \psi}\,\rightarrow\vdash \qquad \dfrac{\Theta \vdash \varphi \rightarrow \psi}{\Gamma \vdash \varphi \rightarrow \psi}\,thinning\vdash*}{\Gamma \vdash \psi}\,M.P.$$

If the sequent version of Modus Ponens is a derived rule with respect to the other sequent rules, then all occurrences of it in the proof constructed by the above induction may be replaced by other rules in the presentation of the system (see §5.5).

2.9.2 Sequent implies Axiomatic

For the other direction, suppose that there is a sequent proof of $\Gamma \vdash \Delta$, for some $\Gamma \cup \Delta \neq \varnothing$. It can be shown by induction on the structure of the sequent proof that a corresponding axiomatic proof may be obtained. The deduction theorem is used to handle the manipulation of assumptions. Two-sided multiple conclusioned sequents can be recast, using the observation in §2.8.6 as an 'impure' axiomatic proof. The proof uses the following notation: '$\Gamma \vdash \varphi, \Delta$' is written as $\begin{matrix}\Gamma \\ \neg\Lambda \\ \vdots \\ \varphi\end{matrix}$ where $\neg\Lambda$ stands for $\{\neg\varphi | \varphi \in \Delta\}$; '$\Gamma \vdash$' is written as $\begin{matrix}\Gamma \\ \vdots \\ \bot\end{matrix}$ where $\bot$ stands for a contradictory formula as before. This 'vertical' notation comes into play for rules which combine several sequents together.

Base Case

In the base case, the basic sequent has the form $\Gamma, \varphi \vdash \varphi, \Delta$ for which the following argument holds:

$$\begin{array}{l} \Gamma \\ \varphi \\ \neg\Lambda \\ \vdots \\ \varphi \end{array}$$

since φ follows by axioms A1–2.

Inductive Cases

There is one inductive case for each rule. Some of these have sub-cases depending on whether Δ can be empty, *i.e.* if the right-hand side of the sequent does not contain an explicit formula. For the case when Δ is empty, the conclusion of the sequent γ can be taken to be $\bot$ and the negated conclusions Λ to be $\varnothing$. Otherwise, when Δ is non-empty it is possible to split Δ into γ and the remaining formulas Λ.

Each case is presented by showing how the impure axiomatic proofs of the antecedents of the rule, which are obtained from the induction hypothesis, can be transformed into an impure axiomatic proof representing the conclusion of the rule. The only steps used involve applications of the available axioms and the rule Modus Ponens to component parts of the sequents. The cases are written in a left-to-right sense as follows: the antecedents, or upper sequents, of the rule are written in a box on the left; a construction that justifies the conclusion is written between arrows; the conclusion or lower sequent of the rule is written in a box on the right.

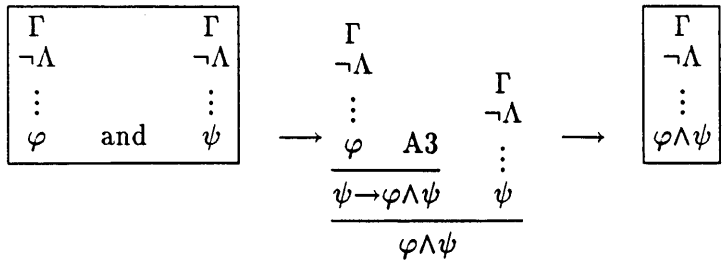
Consider the rule $\wedge\vdash_C$

$$\text{case } \wedge\vdash_C: \quad \frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi\wedge\psi \vdash \Delta}$$

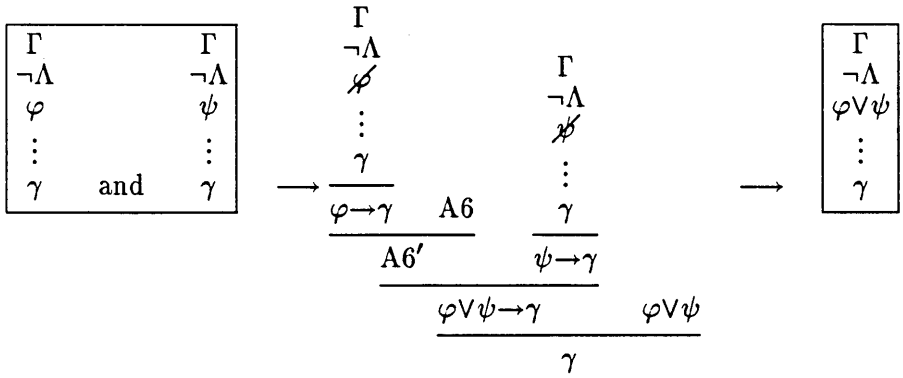
$$\boxed{\begin{array}{l} \Gamma \\ \neg\Lambda \\ \varphi \\ \psi \\ \vdots \\ \gamma \end{array}} \longrightarrow \begin{array}{l} \Gamma \\ \neg\Lambda \\ \dfrac{\varphi\wedge\psi \quad \text{A4a}}{\dfrac{\varphi\wedge\psi \quad \varphi \text{ A4b}}{\psi}} \\ \vdots \\ \gamma \end{array} \longrightarrow \boxed{\begin{array}{l} \Gamma \\ \neg\Lambda \\ \varphi\wedge\psi \\ \vdots \\ \gamma \end{array}}$$

in which the two assumptions φ and ψ of the upper sequent are converted to $\varphi\wedge\psi$ by applications of the axiom A4a and the axiom A4b.

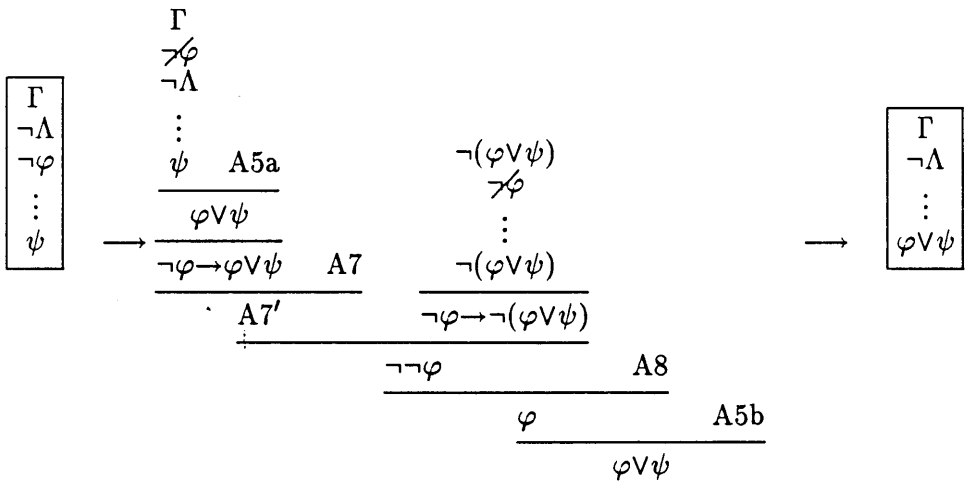
case $\vdash_C\wedge$: $\dfrac{\Gamma\vdash\varphi,\Delta\quad\Gamma\vdash\psi,\Delta}{\Gamma\vdash\varphi\wedge\psi,\Delta}$



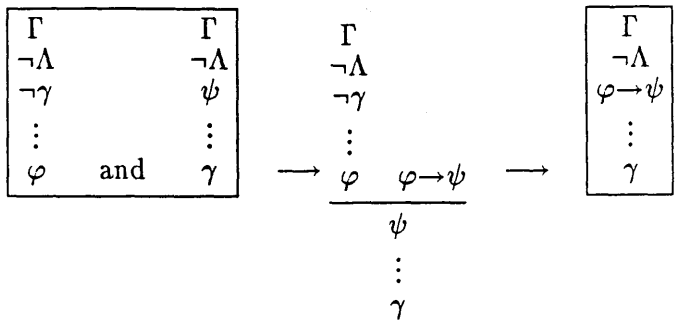
case $\vee\vdash_C$: $\dfrac{\Gamma,\varphi\vdash\Delta\quad\Gamma,\psi\vdash\Delta}{\Gamma,\varphi\vee\psi\vdash\Delta}$



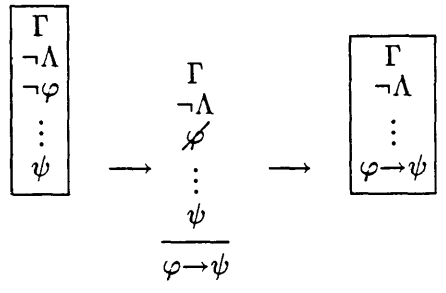
case $\vdash_C\vee$: $\dfrac{\Gamma\vdash\varphi,\psi,\Delta}{\Gamma\vdash\varphi\vee\psi,\Delta}$



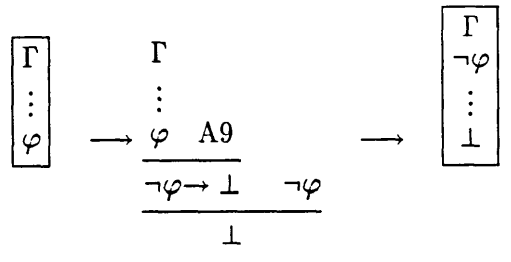
case $\to\vdash_C$: $\dfrac{\Gamma\vdash\varphi,\Delta\quad\Gamma,\psi\vdash\Delta}{\Gamma,\varphi\to\psi\vdash\Delta}$



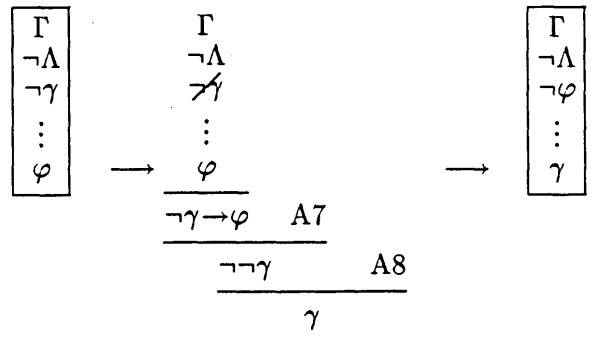
case $\vdash_C\!\rightarrow$: $\dfrac{\Gamma, \varphi \vdash \psi, \Delta}{\Gamma \vdash \varphi\rightarrow\psi, \Delta}$



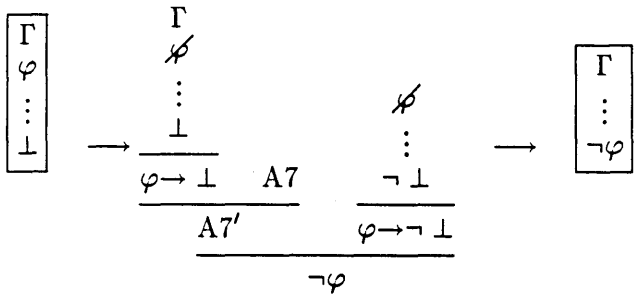
case $\neg\vdash_C a$: $\dfrac{\Gamma \vdash \varphi}{\Gamma, \neg\varphi \vdash \bot}$



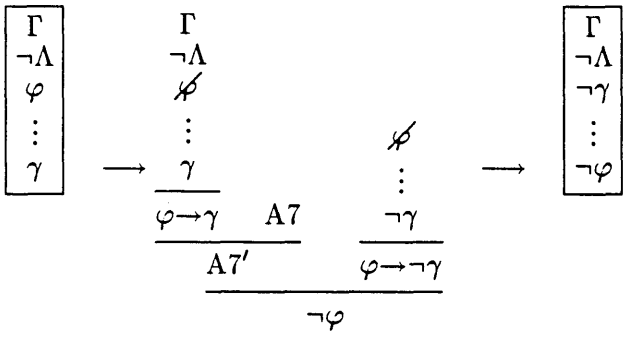
case $\neg\vdash_C b$: $\dfrac{\Gamma \vdash \varphi, \Delta}{\Gamma, \neg\varphi \vdash \Delta}$



case $\vdash_C \neg a$: $\dfrac{\Gamma, \varphi \vdash \bot}{\Gamma \vdash \neg\varphi}$



case $\vdash_C \neg b$: $\dfrac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \neg\varphi, \Delta}$



□

Remarks

Only a few cases require the axiom A8 for classical negation. These are $\neg\vdash_C$ (but only when Δ is not empty) and $\vdash_C \vee$. However, the proof of $\neg\vdash_C$ goes through using axiom A9 when the right-hand side is restricted to a single formula. Similarly, $\vdash_C \vee$ does not hold intuitionistically when more than a single formula is permitted on the right-hand side. Hence, for the intuitionistic case, two rules are needed:

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi} \quad \text{and} \quad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}$$

These can be verified using axiom A5a and axiom A5b respectively.

2.10 Conclusion

This chapter has shown a way of developing a presentation of a logic from a collection of axioms, or Natural Deduction introduction and elimination rules, into an all-introduction presentation. This presentation is suitable for the environment and techniques for verifying that the transformation preserves the meaning of the logic have been demonstrated.

An all-introduction presentation of a logic is preferred over the others described in this chapter as this type of presentation:

- transforms arguments rather than sentences;
- is backwards directed;
- provides for localised discharge of assumptions;
- allows for the structural properties to be varied.

These features give the potential of encoding a variety of non-standard logics, as was seen briefly in §2.6.1[4] Presentations suitable for modal and many-valued logics will be shown in the next chapter.

[4] Although Intuitionistic logic is sometimes considered to be non-standard, when non-standard is taken to mean 'not classical'

Chapter 3

Non-standard logics

This chapter develops the argument about the suitability of all-introduction presentations of logics, by discussing how a number of non-standard logics can be presented in the environment. The systems discussed here are modal and many-valued logics. The presentation of each system is shown and related to the presentations of other styles as was done for Intuitionistic and Classical logics in the previous chapter. Techniques for the derivation of properties of modal systems within the environment are also included. A simple non-monotonic system is also given.

3.1 Introduction

The chapter is structured as follows: The normal modal logics based on the modal system K are examined, and suitable all-introduction presentations are developed. It is shown how composite systems can be constructed which combine the properties of simpler systems. Then, in §3.3, a procedure is described which has been used to discover the distinct modalities of a modal system. This is used to look at the modalities for several classical and intuitionistic presentations of modal systems. An example of an epistemic system in §3.4 demonstrates how a modal operator can be indexed by individuals, giving a multi-modal system. In §3.5 it is shown how finite, many-valued systems can be encoded using all-introduction presentations. In §3.6 the treatment by the environment of logics presented using hypersequents is discussed. Finally, in §3.7 a system containing a defeasible operator 'unless' is presented.

3.2 Modal Logics

Modal logics make distinctions between contingent and necessary truths. For example if ψ_1 = *"Margaret Thatcher is Prime Minister"* then at the time and place of writing ψ_1 is true, but probably not for much longer, and so ψ_1 is not necessarily true. In contrast, if ψ_2 = *"The Earth is the third planet of the Sun"* then ψ_2 is necessarily true, written $\Box\psi_2$. There are many other interpretations that have been given to $\Box\varphi$ depending on the application and properties of the modal system. Examples are: it is necessarily true that φ; it will always be the case that φ; it ought to be true that φ; it is known that φ; it is believed that φ; it is provable that φ; and, when the program terminates, φ will hold (see [Gol87]). Possibility is written as $\Diamond$ and is defined as $\Diamond\varphi = \neg\Box\neg\varphi$. It gives rise to a corresponding variety of readings[1].

3.2.1 Semantics

Although this thesis is primarily concerned with proof theoretic presentations of systems it is instructive to look at the traditional way in which modal systems are given a semantics. This is because there is a close correspondence between the axioms in a Hilbert presentation of a modal system and aspects of its possible world semantics.

The semantics of modal logics is generally given by the *possible worlds* interpretation due to Kripke[Kri59]. In this interpretation, the set of possible worlds is taken to be: all possible configurations, states of knowledge or belief, points in time, *etc.* Worlds are related by a relation R that defines whether a world is accessible from another. So for $\Box\varphi$ to be 'globally' true φ, must be true in each of the possible worlds, whereas for $\Diamond\varphi$ to hold just one possible world will suffice. Relative to a particular world w, the formula $\Box\varphi$ holds iff φ holds in all worlds accessible or reachable from w. The reachability relation R between worlds becomes a focus for study, as different modal axioms force R to have particular properties.

Valuations

To see the relationship between conditions of the accessibility relation R and the axioms of a modal system, a valuation on formulas relative to a possible world is introduced. In this definition, a valuation on propositions returns the set of worlds

[1] Although either $\Box$ or $\Diamond$ can be taken as primitive and the other defined in terms of it, $\Box$ is taken as primitive here.

at which the proposition is true: $V(p) \subset W$. The expression $w \Vdash \varphi$ (read as "w forces φ") means that φ is true at a world w.

$$\begin{aligned}
w &\Vdash p && \text{if} && w \in V(p) \\
w &\Vdash \neg\varphi && \text{if} && w \nVdash \varphi \\
w &\Vdash \varphi\wedge\psi && \text{if} && w \Vdash \varphi \text{ and } w \Vdash \psi \\
w &\Vdash \varphi\vee\psi && \text{if} && \text{either } w \Vdash \varphi \text{ or } w \Vdash \psi \\
w &\Vdash \varphi\rightarrow\psi && \text{if} && w \Vdash \varphi \text{ implies } w \Vdash \psi \\
w &\Vdash \Box\varphi && \text{if} && \forall w'.wRw' \text{ implies } w' \Vdash \varphi \\
w &\Vdash \Diamond\varphi && \text{if} && \exists w'.wRw' \text{ and } w' \Vdash \varphi
\end{aligned}$$

A model M of a modal system S is given by a triple $\langle W, R, V\rangle$ where W is a set of possible worlds, R is a relation between worlds $R \subset W \times W$, and V is a valuation $\Phi \rightarrow \mathbf{P}(W)$.

A formula φ is true in a model M iff $\forall w \in W.w \Vdash \varphi$. A frame F of a modal system S is a pair $\langle W, R\rangle$ where W and R are as before. A formula φ is valid in a frame F iff it is true in all models produced from F by adding an arbitrary valuation.

$\vdash_S\varphi$ is written to mean that φ is a theorem of the system S.

The modal logics examined here are *normal.* A *normal* modal logic is one in which all the possible worlds are considered equal in status, and in which none is distinguished or set apart from the others. Each normal modal logic is an extension of the smallest normal modal logic K, which we will examine first.

3.2.2 Axiomatic presentation

The modal logic K is usually presented axiomatically as an extension of propositional classical logic (of §2.2).

The additions are to its language are the two unary modal operators $\Box$ and $\Diamond$ over formulas. Figure 3.1 illustrates the language of a presentation used in the environment.

In addition to the the axioms and the inference rule Modus Ponens given in the presentation of the classical system, there is an axiom stating the distribution of $\Box$ over $\rightarrow$

$$\vdash \Box(\varphi\rightarrow\psi)\rightarrow(\Box\varphi\rightarrow\Box\psi) \qquad \textit{distribution}$$

and an additional rule of inference:

$$\frac{\vdash \varphi}{\vdash \Box\varphi}\ \textit{necessitation}$$

Syntactic categories are: **Sequent**, **Formula**. The judgement is Sequent.

Formula

a:Formula	'∧' *b*:Formula	→	AND(*a*,*b*)
a:Formula	'∨' *b*:Formula	→	OR(*a*,*b*)
a:Formula	'→' *b*:Formula	→	IMPLIES(*a*,*b*)
a:Formula	'↔' *b*:Formula	→	IFF(*a*,*b*)
	'¬' *a*:Formula	→	NOT(*a*)
	'□' *a*:Formula	→	BOX(*a*)
	'◇' *a*:Formula	→	DIA(*a*)
	t:^[a-z]$	→	ID(*t*)

Sequent

a:SetOf(Formula)	'⊢' *b*:SetOf(Formula)	→	SEQ(*a*,*b*)

Figure 3.1: Language of Classical Modal Systems

Normality is preserved in the sense that if 'φ' is true for an arbitrary world then '$\Box\varphi$' is as well. Similarly, if '$\Box(\varphi\rightarrow\psi)$' holds at some world, then by the distributivity axiom and modus ponens so does '$\Box\varphi\rightarrow\Box\psi$'.

3.2.3 Other normal modal logics

There are many modal logics based on the smallest modal logic K. Some of these are presented in the table 3.2. In that table, a new system S is produced by extending an existing system T with some axioms $A_1 \cdots A_n$ (see table 3.3). This is written as $S = T + A_1 + \cdots + A_n$.

The relationship between axiomatic presentations of modal logics and their possible world semantics has been widely researched for the case of normal modal logics; see for example [HC84]. It was realised that modal axioms correspond to constraints on the nature of the accessibility relation R, and thereby on the classes of possible models for the system.

3.2.4 Deduction Theorem

Recall from chapter 2 that the deduction theorem allows assumptions to be manipulated by allowing the passage from $\varphi, \Gamma \vdash \psi$ to $\Gamma \vdash \varphi\rightarrow\psi$.

The proof given for the deduction theorem in §2.2.1 had two principal cases: one acting on the leaves of the proof tree and the other acting on the rules forming body of the tree. For these cases, the following conditions respectively are true:

System		*Extension*	*Conditions*
K4	=	K + 4	transitivity
KB	=	K + B	symmetry
T	=	K + T	reflexivity
D	=	T + D	idealisation
D4	=	T + 4	idealisation, transitivity
DB	=	T + B	idealisation, symmetry
S4	=	T + 4	reflexivity, transitivity
B	=	T + B	reflexivity, symmetry
S5	=	S4 + B	reflexivity, symmetry, transitivity
	or	S4 + E	

Figure 3.2: Extensions of K

Named	*After*	*Axiom*	*Condition*
T		$\vdash \Box\varphi\rightarrow\varphi$	reflexivity
D	Deontic	$\vdash \Box\varphi\rightarrow\Diamond\varphi$	serial
4		$\vdash \Box\varphi\rightarrow\Box\Box\varphi$	transitivity
E	Euclidean	$\vdash \Diamond\varphi\rightarrow\Box\Diamond\varphi$	euclidean
B	Brouwer	$\vdash \varphi\rightarrow\Box\Diamond\varphi$	symmetric
M	McKinsey	$\vdash \Box\Diamond\varphi\rightarrow\Diamond\Box\varphi$	atomicity
G	Geach	$\vdash \Diamond\Box\varphi\rightarrow\Box\Diamond\varphi$	directed

Figure 3.3: Modal Axioms

Condition	*Constraint on R*
reflexivity	$\forall s.sRs$
serial	$\forall s\exists t.sRt$
transitivity	$\forall s\forall t\forall u.(sRt\wedge tRu\rightarrow sRu)$
euclidean	$\forall s\forall t\forall u.(sRt\wedge sRu\rightarrow tRu)$
symmetric	$\forall s\forall t.(sRt\rightarrow tRs)$
atomicity	transitivity + $\forall s\exists t.(sRt\wedge\forall u.(tRu\rightarrow t = u))$
directed	$\forall s\forall t\forall u.(sRt\wedge sRu\rightarrow\exists v.(tRv\wedge uRv))$

Figure 3.4: Conditions and their first-order constraints on R

1. each axiom of the system can be transformed from ψ to $\varphi \rightarrow \psi$, for an arbitrary φ, and
2. each rule can be transformed from $\dfrac{\psi_1 \cdots \psi_n}{\theta}$ to $\dfrac{\varphi \rightarrow \psi_1 \quad \cdots \quad \varphi \rightarrow \psi_n}{\varphi \rightarrow \theta}$.

This strategy is re-examined here for the modal system K which has an additional rule and another axiom.

The necessitation rule is only applicable if the derivation of φ does not depend on any assumptions. When this is the case, $\Box\varphi$ may also be concluded from no assumptions. Were this not the case, then

$$\frac{\varphi \vdash \varphi}{\varphi \vdash \Box\varphi}$$

would be a valid argument and the meaning of $\Box$ would collapse.

For a given application of *necessitation*

$$\frac{\begin{array}{c}\vdots \\ \psi\end{array}}{\Box\psi}\ \textit{Nec.}$$

it can be deduced from the condition of application of the rule that $\Box\psi$ does not depend on any assumptions in the proof of which it is the conclusion; the proof is 'pure' in the sense described in Chapter 2. So, by one application of axiom A2 and Modus Ponens, we can derive the new formula $\varphi \rightarrow \Box\psi$, thereby satisfying the first of the two conditions.

$$\frac{\dfrac{\begin{array}{c}\vdots \\ \psi\end{array}}{\Box\psi}\ \textit{Nec.} \qquad \overbrace{\Box\psi \rightarrow (\varphi \rightarrow \Box\psi)}^{\text{A2}}}{\varphi \rightarrow \Box\psi}\ \textit{M.P.}$$

So although the necessitation rule is a *rule* it forces the derivation tree above it to behave like an *axiom.*

The distribution axiom also satisfies condition 1 above, so the deduction theorem holds for the system K. The existence of a deduction theorem for the system a sequent presentation of the system K to be constructed.

3.2.5 An all-introduction presentation of K

As described in the previous section, the necessitation rule is only applicable when there are no assumptions present. To use this rule in an all-introduction presentation any, assumptions that are present must first be moved to the right-hand side. After

introducing the '$\Box$' on the right, the distribution axiom allows the newly formed formula to be broken down once again. The following example illustrates this for the case when there is one active assumption φ.

$$\dfrac{\dfrac{\dfrac{\varphi \vdash \psi}{\vdash \varphi \rightarrow \psi}\vdash\!\!\rightarrow}{\vdash \Box(\varphi \rightarrow \psi)}\textit{Nec.} \qquad \dfrac{\dfrac{\vdash \Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)}{\Box(\varphi \rightarrow \psi) \vdash \Box\varphi \rightarrow \Box\psi}\vdash\!\!\rightarrow}{\Box(\varphi \rightarrow \psi), \Box\varphi \vdash \Box\psi}\vdash\!\!\rightarrow}{\Box\varphi \vdash \Box\psi}\textit{M.P.}$$

or more concisely

$$\dfrac{\varphi \vdash \psi}{\Box\varphi \vdash \Box\psi}$$

Repeated applications of the distribution axiom suggest that the following is the general form:

$$\dfrac{\Gamma \vdash \varphi}{\Box\Gamma \vdash \Box\varphi}\,K$$

where $\Box\Gamma$ is a pattern constraining each of the elements of Γ to take the form $\Box\psi$.

3.2.5.1 Justification

To confirm that this rule is sufficient it must be shown (i) that the distribution axiom and the necessitation rule follow from it, and (ii) that given the distribution axiom and the necessitation rule, the rule K can be derived.

(i) The distribution axiom is derived as follows

$$\dfrac{\dfrac{\dfrac{\dfrac{\varphi \vdash \varphi \quad \psi \vdash \psi}{\varphi \rightarrow \psi, \varphi \vdash \psi}\rightarrow\!\!\vdash}{\Box(\varphi \rightarrow \psi), \Box\varphi \vdash \Box\psi}K}{\Box(\varphi \rightarrow \psi) \vdash \Box\varphi \rightarrow \Box\psi}\vdash\!\!\rightarrow}{\vdash \Box(\varphi \rightarrow \psi) \rightarrow \Box\varphi \rightarrow \Box\psi}\vdash\!\!\rightarrow$$

Note that the rule for K reduces to the necessitation rule when Γ is empty.

(ii) K is shown by induction on the number of assumptions in Γ.

Γ_n is written as $\psi_1, \ldots, \psi_n$.

In the case when $i = 0$ the result is immediate by the necessitation rule.

For the case $0 < i \leq n$ assume that the rule holds for $i - 1$ and argue as follows

$$\dfrac{\dfrac{\Box\psi_i \vdash \Box\psi_i \quad \Box\varphi \vdash \Box\varphi}{\Box\psi_i, \Box\psi_i \rightarrow \Box\varphi \vdash \Box\varphi}\rightarrow\vdash \quad \dfrac{\dfrac{\dfrac{\Gamma_i \vdash \varphi}{\psi_i, \Gamma_{i-1} \vdash \varphi}\vdash\rightarrow}{\Gamma_{i-1} \vdash \psi_i \rightarrow \varphi}}{\Box\Gamma_{i-1} \vdash \Box(\psi_i \rightarrow \varphi)}\,Ind.Hyp.}{\Box\Gamma_{i-1} \vdash \Box\psi_i \rightarrow \Box\varphi}\,Distr.}{\Box\Gamma_i \vdash \Box\varphi}\,M.P.$$

The intuitive reading of the rule K is that to conclude $\Box\varphi$ from Γ one must strengthen all the assumptions to $\Box\Gamma$.

3.2.6 Possibility and Necessity

The above analysis is adequate for Intuitionistic sequents, which are restricted to a single conclusion on the right-hand side. The all-introduction presentation is not sufficient for a Classical sequent having sets on both sides, because of the rules for negation in the all-introduction presentation, described in §2.8.4.

Using the standard interpretation of a sequent '$\Gamma \vdash \Delta$' as '$\bigwedge\Gamma \rightarrow \bigvee\Delta$' or '$(\psi_1 \wedge \cdots \wedge \psi_n) \rightarrow (\varphi_1 \vee \cdots \vee \varphi_m)$' and by defining '$\Diamond\varphi$' as '$\neg\Box\neg\varphi$' the assumptions and conclusions can be separated according to whether the patterns are $\Box\neg$ or $\Box$; the normal rule may be written as two separate rules:

$$\frac{\Gamma, \neg\Delta \vdash \varphi}{\Box\Gamma, \Box\neg\Delta \vdash \Box\varphi} \quad \text{and} \quad \frac{\Gamma, \neg\Delta \vdash \neg\varphi}{\Box\Gamma, \Box\neg\Delta \vdash \Box\neg\varphi}$$

Using $\Box\neg\varphi \rightarrow \neg\Diamond\varphi$ and the rules for negation, by switching sides, the versions of these rules are obtained, which treat $\Box$ and $\Diamond$ together.

$$\frac{\Gamma \vdash \varphi, \Delta}{\Box\Gamma \vdash \Box\varphi, \Diamond\Delta} \quad \text{and} \quad \frac{\varphi, \Gamma \vdash \Delta}{\Diamond\varphi, \Box\Gamma \vdash \Diamond\Delta}$$

A presentation is more suitable for the environment when it is possible to suppress any explicit thinning rules in favour of generalised basic sequents, adding additional thinning only to those rules which require it. For example, the patterns in the conclusion of the first rule above do not match any non-boxed formula on the left-hand side of the sequent, and consequently, explicit thinning is required at this point. The following rules, known as *K-a* and *K-b*, incorporate the necessary thinning.

$$\frac{\Gamma \vdash \varphi, \Delta}{\Gamma', \Box\Gamma \vdash \Box\varphi, \Diamond\Delta, \Delta'}\,K\text{-}a \quad \text{and} \quad \frac{\varphi, \Gamma \vdash \Delta}{\Diamond\varphi, \Box\Gamma, \Gamma' \vdash \Diamond\Delta, \Delta'}\,K\text{-}b$$

See also table 3.5 in which *K-a* and *K-b* are labelled as "K, D, T".

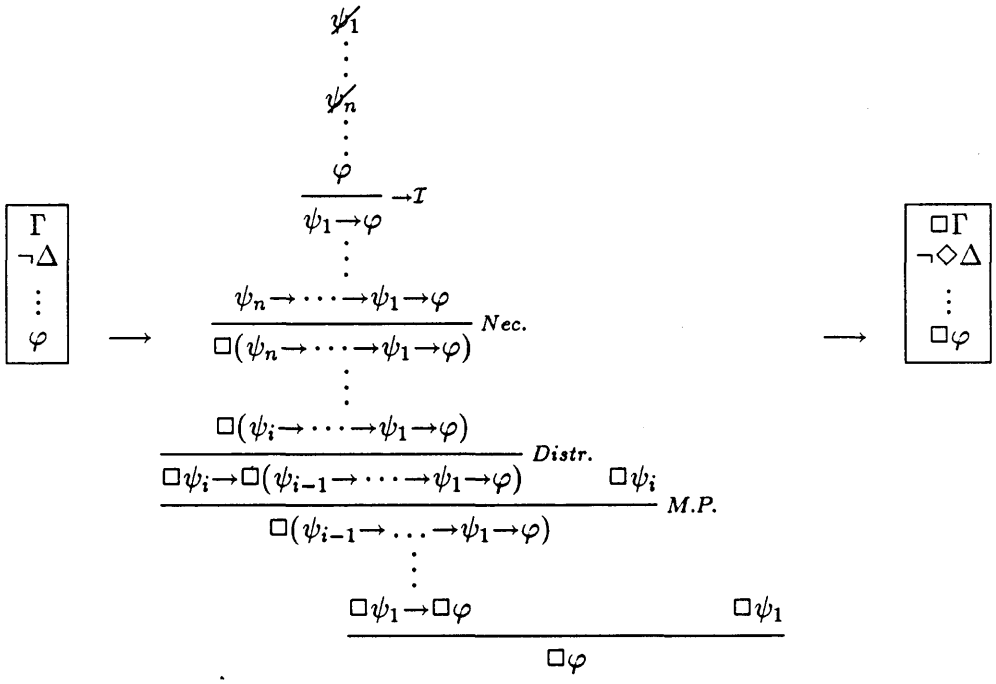
3.2.7 Equivalence

A more rigorous justification of the system **K** is now presented, which extends the equivalence proof for the Classical presentation in Chapter 2, using the modal rules *K-a* and *K-b*.

The *sequent* $\Rightarrow$ *axiomatic* direction was covered in the previous section in the sense that the sequent rules were used to derive the axioms and rules of the Hilbert presentation.

For the *axiomatic* $\Rightarrow$ *sequent* direction additional inductive cases must be considered for the *K-a* and *K-b* rules. In the following let $\psi_1, \ldots, \psi_n$ represent the assumptions $\Gamma \cup \neg\Delta$. The proof involves an inner induction on the number of these assumptions.

case *K-a*:
$$\frac{\Gamma \vdash \varphi, \Delta}{\Box\Gamma \vdash \Box\varphi, \Diamond\Delta}$$



Writing assumptions having the form $\Box\neg\psi_i$ as $\neg\Diamond\psi_i$ gives the $\neg\Diamond\Delta$ used in the rule. The rest are collected together as $\Box\Gamma$.

The construction used in the proof of the rule *K-b* is similar.

3.2.8 Presentations of other modal systems

The earliest formulation that the author has discovered of the all-introduction presentation of **K** shown above is by Curry in[Cur77]. This is however rather sketchy

especially with regard to the combined treatment of $\Box$ and $\Diamond$. A systematic study of a number of many modal systems is found in Fitting[Fit83] but the treatment there is mostly concerned with modal tableaux systems which, although related to the sequent presentations given here, are motivated from a semantic perspective rather than the axiomatic approach used in this thesis.

The next five sections present derivations of all-introduction presentations of extensions of the modal system K based on the axioms T, D, 4, B & E. All except T make use of a slightly more general form of the inductive argument given to justify the modal rule *K*. The modal system *T* is justified first, and then a schematic version of the induction is provided for the other systems.

3.2.8.1 System T

The axiom representing reflexivity $\vdash \Box\varphi\rightarrow\varphi$ can be seen to be equivalent to the rule:

$$\frac{\varphi, \Gamma \vdash \psi}{\Box\varphi, \Gamma \vdash \psi}\, T$$

The rule *T* entails the reflexivity axiom as follows:

$$\dfrac{\dfrac{\varphi \vdash \varphi}{\Box\varphi \vdash \varphi}\, T}{\vdash \Box\varphi\rightarrow\varphi}\, \vdash\rightarrow$$

Conversely, for arbitrary Γ, a single instance of the axiom, and an application of cut derive the rule *T*:

$$\frac{\Box\varphi \vdash \varphi \quad \varphi, \Gamma \vdash \psi}{\Box\varphi, \Gamma \vdash \psi}\, M.P.$$

This rule is independent of the system K. To have a presentation of T (*i.e.* $\mathsf{K}+T$) the rules for both T and K must be included.

Systems D, 4, B & E

The following argument relies on the fact that each of the axioms D, 4, B & E for the other properties can be stated in the form $\vdash \beta\rightarrow\Box\alpha$.

Suppose that the rule

$$\frac{\Gamma_1, \ldots, \Gamma_n \vdash \psi}{\Delta_1, \ldots, \Delta_m \vdash \Box\psi}\, R$$

is intended to represent the contribution of the axiom *A* to the modal system formed by extending the basic modal system K. The assumptions in the upper-sequent Γ_i and the assumptions in the lower-sequent Δ_j may be modal patterns. A *modal pattern* is defined to be a sequence of unary operators, including negation, prefixing a metavariable over a collection of formulas *e.g.* '$\Box\neg\Gamma$' or 'Δ'.

The proof that the rule faithfully implements the axiom is in two parts.

$R \Rightarrow A$ The rule entails the axiom. Use R to derive A.

$\mathsf{K} + A \Rightarrow R$ The system K together with the axiom entail the rule. Proceed by induction on the size of the assumptions, Θ, in the lower-sequent, *e.g.* $\Theta = \bigcup_i^m \Delta_i$.

1. When Θ is empty R follows from necessitation.
2. When Θ is non-empty. Take some $\theta \in \Theta$ and argue by sub-cases according to the modal pattern it matches:

$$\dfrac{\dfrac{\overbrace{\theta \vdash \Box\alpha}^{\text{axiom}} \quad \Box\psi \vdash \Box\psi}{\theta, (\Box\alpha \rightarrow \Box\psi) \vdash \Box\psi}\,{\rightarrow\!\vdash} \quad \dfrac{\dfrac{\dfrac{\theta', \Gamma_1, \ldots, \Gamma_n \vdash \psi}{\alpha, \Gamma_1, \ldots, \Gamma_n \vdash \psi}\,\textit{Various}}{\Gamma_1, \ldots, \Gamma_n \vdash \alpha \rightarrow \psi}\,{\vdash\!\rightarrow}}{\Delta_1, \ldots, \Delta_m \vdash \Box(\alpha \rightarrow \psi)}\,\textit{Ind.Hyp.}}{\theta, \Delta_1, \ldots, \Delta_m \vdash \Box\psi}\,\textit{Distr.}\;\textit{M.P.}$$

in which, reading from bottom-to-top:

- θ is a formula matching the pattern Δ_i.
- $\Box\alpha$ is constructed using an appropriate instance of the axiom A.
- The conclusion of the rule and $\Box\alpha$ are packaged using $\rightarrow\!\vdash$ and pushed through with the aid of the distribution axiom and the induction hypothesis.
- Further steps may be needed before the appropriate form is achieved.

The following sections give examples of this schema when it is applied to the axioms 4, D, B & E.

3.2.8.2 System 4

We show the equivalence of the axiom $\vdash \Box\varphi \rightarrow \Box\Box\varphi$ representing transitivity with the modal rule

$$\dfrac{\Box\Gamma \vdash \varphi}{\Box\Gamma \vdash \Box\varphi}\,4$$

Note that the rule collapses to the axiom:

$$\dfrac{\dfrac{\Box\varphi \vdash \Box\varphi}{\Box\varphi \vdash \Box\Box\varphi}\,4}{\vdash \Box\varphi \rightarrow \Box\Box\varphi}\,{\vdash\!\rightarrow}$$

To show the other direction proceed by induction as before. The base case is straightforward by necessitation. For the inductive case argue as follows:

$$\dfrac{\dfrac{\dfrac{\Box\varphi,\Box\Gamma\vdash\psi}{\Box\Gamma\vdash\Box\varphi\rightarrow\psi}\vdash\rightarrow}{\Box\Gamma\vdash\Box(\Box\varphi\rightarrow\psi)}\,Ind.Hyp.\qquad \dfrac{\Box\varphi\vdash\Box\Box\varphi \quad \Box\psi\vdash\Box\psi}{\Box\varphi,\Box\Box\varphi\rightarrow\Box\psi\vdash\Box\psi}\rightarrow\vdash}{\Box\Gamma\vdash\Box\Box\varphi\rightarrow\Box\psi}\,Distr.}{\Box\varphi,\Box\Gamma\vdash\Box\psi}\,M.P.$$

3.2.8.3 System D

For the deontic axiom $\vdash\Box\varphi\rightarrow\Diamond\varphi$ note that this is the same as $\vdash\Box\neg\varphi\rightarrow\neg\Box\varphi$. The corresponding modal rule is

$$\dfrac{\Gamma,\varphi\vdash}{\Box\Gamma,\Box\varphi\vdash}\,D$$

This derives the axiom as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\varphi\vdash\varphi}{\neg\varphi,\varphi\vdash}\neg\vdash}{\Box\neg\varphi,\Box\varphi\vdash}\,D}{\Box\neg\varphi\vdash\neg\Box\varphi}\vdash\neg$$

For the other direction, the rule for K given above is used to save an explicit induction:

$$\dfrac{\dfrac{\dfrac{\dfrac{\varphi,\Gamma\vdash}{\Gamma\vdash\neg\varphi}\vdash\neg}{\Box\Gamma\vdash\Box\neg\varphi}\,K \qquad \overbrace{\vdash\Box\neg\varphi\rightarrow\neg\Box\varphi}^{\text{axiom}}}{\Box\Gamma\vdash\neg\Box\varphi}\,M.P. \qquad \dfrac{\Box\varphi\vdash\Box\varphi}{\Box\varphi,\neg\Box\varphi\vdash}\neg\vdash}{\Box\varphi,\Box\Gamma\vdash}\,M.P.$$

But note that the rule for D must be used with the rule for K.

3.2.8.4 System B

The symmetric normal modal system B is given by distribution, necessitation and the axiom $\vdash\varphi\rightarrow\Box\Diamond\varphi$ or equivalently$\vdash\varphi\rightarrow\Box\neg\Box\neg\varphi$. These are equivalent to the modal rule

$$\dfrac{\Gamma,\neg\Box\neg\Delta\vdash\varphi}{\Box\Gamma,\Delta\vdash\Box\varphi}\,B$$

This rule allows the derivation of the B axiom

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\neg\varphi \vdash \neg\varphi}{\Box\neg\varphi \vdash \Box\neg\varphi}\,K}{\neg\Box\neg\varphi \vdash \neg\Box\neg\varphi}\,\vdash\neg,\neg\vdash}{\varphi \vdash \Box\neg\Box\neg\varphi}\,B}{\vdash \varphi\rightarrow\Box\neg\Box\neg\varphi}\,\vdash\rightarrow$$

For the other direction argue by induction on the size of $\Theta = \Box\Gamma \cup \Delta$ and by subcases depending on the structure of the particular $\theta \in \Theta$. The base case, when Θ is empty follows from necessitation. To show the case for $i+1$, it is sufficient to assume the rule holds for i.

Case (i) θ has the form $\Box\varphi$

$$\dfrac{\dfrac{\overbrace{\Box\varphi \vdash \Box\neg\Box\neg\Box\varphi}^{\text{axiom}} \quad \Box\psi \vdash \Box\psi}{\Box\varphi, \Box\neg\Box\neg\Box\varphi\rightarrow\Box\psi \vdash \Box\psi}\,\rightarrow\vdash \qquad \dfrac{\dfrac{\dfrac{\overbrace{\neg\Box\neg\Box\varphi \vdash \varphi}^{\text{axiom}} \quad \varphi, \Gamma, \neg\Box\neg\Delta \vdash \psi}{\Gamma, \neg\Box\neg\Delta, \neg\Box\neg\Box\varphi \vdash \psi}\,M.P.}{\Gamma, \neg\Box\neg\Delta \vdash \neg\Box\neg\Box\varphi\rightarrow\psi}\,\vdash\rightarrow}{\Box\Gamma, \Delta \vdash \Box(\neg\Box\neg\Box\varphi\rightarrow\psi)}\,Ind.Hyp.}{\Box\Gamma, \Delta \vdash \Box\neg\Box\neg\Box\varphi\rightarrow\psi}\,Distr.}{\Box\varphi, \Box\Gamma, \Delta \vdash \psi}$$

Case (ii) θ has some other form φ

$$\dfrac{\dfrac{\overbrace{\varphi \vdash \Box\neg\Box\neg\varphi}^{\text{axiom}} \quad \Box\psi \vdash \Box\psi}{\varphi, \Box\neg\Box\neg\varphi\rightarrow\Box\psi \vdash \Box\psi}\,\rightarrow\vdash \qquad \dfrac{\dfrac{\dfrac{\Gamma, \neg\Box\neg\varphi, \neg\Box\neg\Delta \vdash \psi}{\Gamma, \neg\Box\neg\Delta \vdash \neg\Box\neg\varphi\rightarrow\psi}\,\vdash\rightarrow}{\Box\Gamma, \Delta \vdash \Box(\neg\Box\neg\varphi\rightarrow\psi)}\,Ind.Hyp.}{\Box\Gamma, \Delta \vdash \Box\neg\Box\neg\varphi\rightarrow\Box\psi}\,Distr.}{\Box\Gamma, \varphi, \Delta \vdash \Box\psi}\,M.P.$$

3.2.8.5 System E

The axiom E $\vdash \neg\Box\varphi\rightarrow\Box\neg\Box\varphi$ gives rise to the rule

$$\dfrac{\neg\Box\Gamma \vdash \varphi}{\neg\Box\Gamma \vdash \Box\varphi}\,E$$

This rule easily derives the axiom for E:

$$\dfrac{\dfrac{\dfrac{\neg\Box\varphi \vdash \neg\Box\varphi}{\neg\Box\varphi \vdash \Box\neg\Box\varphi}\,E}{\vdash \neg\Box\varphi\rightarrow\Box\neg\Box\varphi}\,\vdash\rightarrow$$

Conversely, by induction on elements in $\neg\Box\Gamma$:

The case when Γ is empty follows directly from necessitation.

When Γ is non-empty:

$$\dfrac{\dfrac{\overbrace{\vdash \neg\Box\varphi\rightarrow\Box\neg\Box\varphi}^{\text{axiom}}}{\neg\Box\varphi \vdash \Box\neg\Box\varphi}\,M.P. \qquad \Box\psi \vdash \Box\psi}{\neg\Box\varphi, \Box\neg\Box\varphi\rightarrow\Box\psi \vdash \Box\psi}\rightarrow\vdash \qquad \dfrac{\dfrac{\dfrac{\neg\Box\varphi, \neg\Box\Gamma \vdash \Box\psi}{\neg\Box\Gamma \vdash \neg\Box\varphi\rightarrow\Box\psi}\vdash\rightarrow}{\neg\Box\Gamma \vdash \Box(\neg\Box\varphi\rightarrow\psi)}\,Ind.Hyp.}{\neg\Box\Gamma \vdash \Box\neg\Box\varphi\rightarrow\Box\psi}\,Distr.}{\neg\Box\varphi, \neg\Box\Gamma \vdash \Box\psi}\,M.P.$$

Systems	$\Box$	$\Diamond$
K, D, T	$\dfrac{\Gamma \vdash \varphi, \Delta}{\Box\Gamma, \Gamma' \vdash \Box\varphi, \Diamond\Delta, \Delta'}$	$\dfrac{\varphi, \Gamma \vdash \Delta}{\Diamond\varphi, \Box\Gamma, \Gamma' \vdash \Diamond\Delta, \Delta'}$
K4, D4	$\dfrac{\Box\Gamma, \Gamma \vdash \varphi, \Diamond\Delta, \Delta}{\Box\Gamma, \Gamma' \vdash \Box\varphi, \Diamond\Delta, \Delta'}$	$\dfrac{\varphi, \Box\Gamma, \Gamma \vdash \Diamond\Delta, \Delta}{\Diamond\varphi, \Box\Gamma, \Gamma' \vdash \Diamond\Delta, \Delta'}$
T, D, S4, S5	$\dfrac{\varphi, \Gamma \vdash \Delta}{\Box\varphi, \Gamma \vdash \Delta}$	$\dfrac{\Gamma \vdash \varphi, \Delta}{\Gamma \vdash \Diamond\varphi, \Delta}$
S4	$\dfrac{\Box\Gamma \vdash \varphi, \Diamond\Delta}{\Box\Gamma, \Gamma' \vdash \Box\varphi, \Diamond\Delta, \Delta'}$	$\dfrac{\varphi, \Box\Gamma \vdash \Diamond\Delta}{\Diamond\varphi, \Box\Gamma, \Gamma' \vdash \Diamond\Delta, \Delta'}$
S5	$\dfrac{\Box\Gamma, \Diamond\Gamma' \vdash \varphi, \Box\Delta, \Diamond\Delta'}{\Box\Gamma, \Diamond\Gamma', \Gamma'' \vdash \Box\varphi, \Box\Delta, \Diamond\Delta', \Delta''}$	$\dfrac{\varphi, \Box\Gamma, \Diamond\Gamma' \vdash \Box\Delta, \Diamond\Delta'}{\Diamond\varphi, \Box\Gamma, \Diamond\Gamma', \Gamma'' \vdash \Box\Delta, \Diamond\Delta', \Delta''}$
D	$\dfrac{\Gamma \vdash \Delta}{\Box\Gamma, \Gamma' \vdash \Diamond\Delta, \Delta'}$	
D4	$\dfrac{\Box\Gamma, \Gamma \vdash \Diamond\Delta, \Delta}{\Box\Gamma, \Gamma' \vdash \Diamond\Delta, \Delta'}$	
DB	$\dfrac{\Gamma, \Diamond\Gamma', \Diamond\Gamma'' \vdash \Delta, \Box\Delta', \Box\Delta''}{\Box\Gamma, \Diamond\Gamma', \Gamma'' \vdash \Diamond\Delta, \Box\Delta', \Delta''}$	
KB, DB, B	$\dfrac{\Gamma, \Diamond\Gamma', \Diamond\Gamma'' \vdash \varphi, \Delta, \Box\Delta', \Box\Delta''}{\Box\Gamma, \Diamond\Gamma', \Gamma'' \vdash \Box\varphi, \Box\Delta, \Diamond\Delta', \Delta''}$	$\dfrac{\varphi, \Gamma, \Diamond\Gamma', \Diamond\Gamma'' \vdash \Delta, \Diamond\Delta', \Box\Delta''}{\Diamond\varphi, \Box\Gamma, \Diamond\Gamma', \Gamma'' \vdash \Box\Delta, \Diamond\Delta', \Delta''}$

Figure 3.5: Summary of Classical Normal Modal Rules

3.2.9 Composite systems

It is possible to construct presentations of hybrid systems having the required combinations of properties if the rules of the separate systems are combined. The table 3.5 summarises the rules for the systems: K4, KB, D4, DB, S4, S5.

Axioms may be used together to form composite rules by iterating the construction of the formula used in the schematic proof shown in previous section. Instead of the cut formula used there, $\Box\alpha\rightarrow\Box\psi$, take the formula $\Box\beta_1\rightarrow\cdots\rightarrow\Box\beta_n\rightarrow\Box\psi$, where each β_i corresponds to an axiom of the composite system as before. The required rule can be obtained by iterating the use of the distribution axiom, and by using appropriate rearrangement after the induction hypothesis has been applied. The example given here is for the case of S4 which combines the rules for K and 4 into a single rule.

3.2.9.1 System S4

The modal logic S4 is K extended by the axioms T and 4. The rules for these were

$$\frac{\Gamma\vdash\varphi}{\Box\Gamma\vdash\Box\varphi}\,K \qquad \frac{\varphi,\Gamma\vdash\psi}{\Box\varphi,\Gamma\vdash\psi}\,T \qquad \frac{\Box\Gamma\vdash\varphi}{\Box\Gamma\vdash\Box\varphi}\,4$$

Combining the effect of K and 4 forms the rule:

$$\frac{\Gamma,\Box\Gamma\vdash\varphi}{\Box\Gamma\vdash\Box\varphi}\,K4$$

with the following justification

$$\frac{\dfrac{\dfrac{\Box\varphi\vdash\Box\varphi}{\varphi,\Box\varphi\vdash\Box\varphi}\,thin}{\Box\varphi\vdash\Box\Box\varphi}\,K4 \qquad \dfrac{\dfrac{\Gamma\vdash\varphi}{\Gamma,\Box\Gamma\vdash\varphi}\,thinning*}{\Box\Gamma\vdash\Box\varphi}\,K4$$

and conversely

$$\dfrac{\Box\varphi\rightarrow\Box\varphi \quad \dfrac{\overbrace{\Box\varphi\vdash\Box\Box\varphi}^{\text{axiom}} \quad \Box\psi\vdash\Box\psi}{\Box\varphi,\Box\Box\varphi\rightarrow\Box\psi\vdash\Box\psi}\,{\rightarrow}\vdash}{\Box\varphi,\Box\varphi\rightarrow\Box\Box\varphi\rightarrow\Box\psi\vdash\Box\psi}\,{\rightarrow}\vdash \qquad \dfrac{\dfrac{\dfrac{\varphi,\Box\varphi,\Box\Gamma\vdash\psi}{\Box\Gamma\vdash\varphi\rightarrow\Box\varphi\rightarrow\psi}\,{\vdash\rightarrow}\times2}{\Box\Gamma\vdash\Box(\varphi\rightarrow\Box\varphi\rightarrow\psi)}\,Ind.Hyp.}{\Box\Gamma\vdash\Box\varphi\rightarrow\Box\Box\varphi\rightarrow\Box\psi}\,Distr.\times 2$$

$$\Box\varphi,\Box\Gamma\vdash\Box\psi\;M.P.$$

The rule T is needed as well. To see that T cannot be merged with K and 4, it is enough to note that K and 4 require that the right-hand side of the conclusion of the rule is of the form '$\Box\varphi$'. As this is more specific than T, which does not make any restriction, the $K4$ and T rules are not compatible. The rule for T is kept apart from the combined rule for K4, and together T and $K4$ give S4.

3.2.10 Sub-formula property

The rules in table 3.5 involving symmetry based on axiom B - namely those for B, DB, KB and S5 - do not have the subformula property, and are potentially divergent. Consider the rule B derived above:

$$\frac{\Gamma, \neg\Box\neg\Delta \vdash \varphi}{\Box\Gamma, \Delta \vdash \Box\varphi}\, B$$

The term $\neg\Box\neg\Delta$ is not a subformula of the lower sequent. Consequently, repeated use of this rule will lead to larger and larger terms and potential divergence of the theorem prover. The intuitive effect of the rule is that what was known to the current world becomes possible in the next; it is this 'inheritance' that leads to the combinatorial explosion.

A different type of presentation, involving *hypersequents* has been developed to find cut-free, all-introduction presentations of symmetric modal systems. Hypersequent presentations can be modelled cleanly in the environment, and are described in §3.6.

3.2.11 Strategies for Modal Systems

It is often convenient to represent a collection of modal systems together as one system. An example might consist of the language shown in figure 3.1 and the rules in table 3.5 together with the rules of the propositional core developed in §2.8.6.

The modal rules are named by the systems in which they are applicable. Thus the strategy "`<S4,left,right>[BASIC]`" picks the modal rules labelled as appropriate for S4 as well as the classical logic core.

System	Strategy	
K		`<K,left,right>[BASIC]`
K4	= K + 4	`<K4,left,right>[BASIC]`
T	= K + T	`<T,left,right>[BASIC]`
D	= K + D	`<D,left,right>[BASIC]`
D4	= D + 4	`<D4,left,right>[BASIC]`
B		`<B,left,right>[BASIC]`
KB	= K + B	`<KB,left,right>[BASIC]`
DB	= D + B	`<DB,left,right>[BASIC]`
S4	= K + T + 4	`<S4,left,right>[BASIC]`
S5	= K + T + 4 + B	`<S5,left,right>[BASIC]`

Figure 3.6: Some modal strategies

3.3 Determining Modalities

3.3.1 Introduction

Finding the distinct modalities of a modal system gives an indication of its structure. If the modal system models a temporal logic, then the existence of a finite number of modalities says that there is no need to iterate tenses beyond certain combinations. For example, saying "it will be the case that it will rain" is not essentially different from "it will rain" in English.

Similarly, the modalities may denote sequences of actions, perhaps updates that may take place in some database system. In this case, the existence of a finite number of distinct modalities indicates that it is sufficient to focus study on representative sequences rather than arbitrary ones, with a consequent reduction in the complexity of the analysis.

The process used for determining the distinct modalities present in a modal system involves a large number of smaller derivations in the system itself. The success or failure of these derivations guides the algorithm in its search for the modalities.

This process illustrates the sort of "normal form" result which is desirable about a logical system being developed in an environment such as this. Other examples might include disjunctive normal forms for binary connectives, or prenex-forms for quantifiers. These results show how combinations of logical operations interact and give the user of the environment valuable insights into the combinatorial complexity of the system.

3.3.2 Modalities

A *modality* is defined as a string consisting of zero or more occurrences of unary operators, for example $\{\neg, \Box, \Diamond\}$. The zero case is written as the null string, '$-$'. If the modal system has $\neg\Box = \Diamond\neg$ and $\neg\neg = -$, as it would if based on classical logic, then every modality can be written with at most a single negation at the front; not all systems have this property. Example modalities include: $\Box\Box$ "what will be will be", $\Diamond\Box$, "what may be will be", *etc.* The modal systems discussed below provide a further source of interpretations.

3.3.3 Equivalence

Two modalities, m, n, are *equivalent* in a given modal system iff the result of replacing m by n (or m by n) in any formula is always equivalent, in that system, to the original formula. The modalities are otherwise not equivalent or *distinct.*

If m and n are equivalent in a certain system, and m contains fewer modal operators than n then n is said to be reducible to m in the system.

3.3.4 Finiteness

Some modal systems such as K have an infinite number of distinct modalities. Others have a finite number; for instance classical S4 has only the 14 listed in figure 3.7. Further collections are given in the examples below.

–	□	◇	□◇	◇□	□◇□	◇□◇
¬	¬□	¬◇	¬□◇	¬◇□	¬□◇□	¬◇□◇

Figure 3.7: S4 modalities

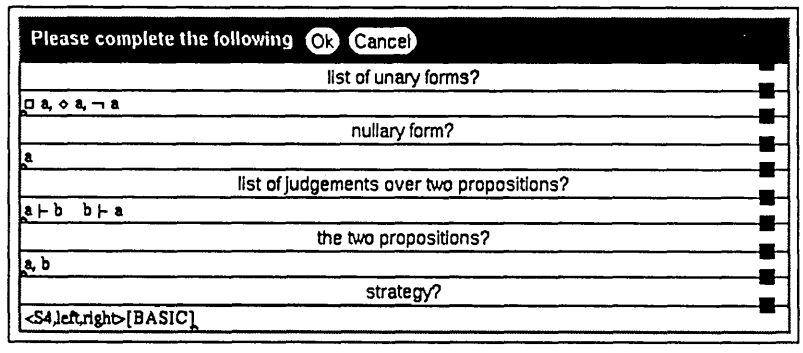


Figure 3.8: Modalities dialogue

3.3.5 Algorithm

The environment uses the following algorithm to determine the distinct modalities for a given modal system. On pressing the Find Modalities button, the user of the environment is prompted for some details of the system and for a strategy to use (see figure 3.8 and also chapter 6). The algorithm is parameterised with respect to the following.

1. Those operators that are to be used to form the modalities. These are usually $\Box$, $\Diamond$, and $\neg$ given above. However, is it useful to have control over whether negation should be considered. The modalities may be other operators such as the "i believes φ" or in symbols '$B\langle i\rangle$', see §3.4 below, rather than the operators box, diamond, *etc.*

2. The initial case of the null modality. Unary operators are stated in terms of this case. For example, suppose 'a' was given here then the constructions given for the first question would be '$\Box a$', '$\Diamond a$', and '$\neg a$'.

3. Judgements that are used to decide when two modalities are equivalent. The judgement used is often '$\vdash \varphi \leftrightarrow \psi$', but other forms of judgement may be necessary if, for example, the core language does not contain $\rightarrow$ or $\leftrightarrow$. In this case, replacing '$\vdash \varphi \leftrightarrow \psi$' with '$\varphi \vdash \psi$' and '$\psi \vdash \varphi$' might be appropriate.

4. A strategy for the theorem prover to use.

The algorithm starts by considering a tree of modalities rooted at the null modality (illustrated in figure 3.9). The null modality is the shortest of all the modalities and as such is taken as a distinct modality in its own right. The tree is grown to consider the modalities at the next level. This level is formed by taking any irreducible modalities from the previous level and forming new modalities by substituting the unary constructors for the base cases in each of them. For example

$$\{\Box a, \Diamond a\} \mapsto \{\Box\Box a, \Box\Diamond a, \Box\neg a, \Diamond\Box a, \Diamond\Diamond a, \Diamond\neg a\}$$

Each new candidate modality of the new level is considered with respect to the distinct modalities that are known so far, using the judgements given by the user.

Suppose that the set of modalities found at some point in the computation is partitioned into non-empty sets of modalities, each of which is labelled by one of the shortest modalities contained within it, such that $m \in M_n$ iff all the judgements given in the dialogue hold for appropriate n and m. This will usually be $m\varphi \vdash n\varphi$ and $n\varphi \vdash m\varphi$. So $M_- = \{-, \neg\neg, \ldots\}$; $M_\Box = \{\Box, \Box\Box, \ldots\}$; $M_\Diamond = \{\Diamond, \Diamond\Diamond, \ldots\}$; *etc.*

For each new modality encountered, either there is some n such that m satisfies our condition, or else the modality is irreducible and so forms a new set $M_m = \{m\}$, having itself as a label.

When there are no new modalities discovered at a particular level, then all the modalities are reducible to some existing (shorter) modality. In such a case there are no new candidate modalities to check and the search can be closed. The distinct modalities are taken as the labels of the sets in M.

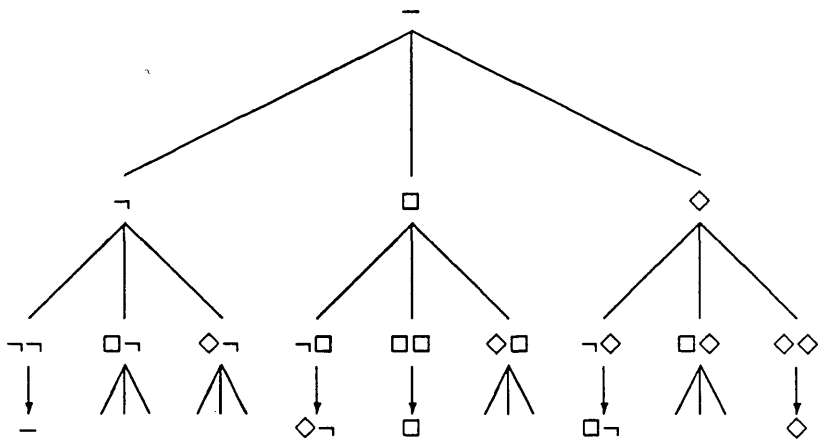


Figure 3.9: Modalities generated to depth two

The following sections show the results obtained by applying this type of analysis to a number of modal systems in the environment. In particular it is shown that modal systems built over intuitionistic bases have many more modalities, and that they can be seen to reduce to their classical counterparts under the introduction of double negation elimination.

3.3.6 Classical S4

Figure 3.10 gives an example of the results produced by the environment for S4. The listing shows the commentary given on the progress of the algorithm as it executes. The numbers at the left-hand side indicate the level of the modality tree that is being considered at that point. The reductions of modalities that are possible at that level are displayed and, when the analysis of the level has been completed, all the new distinct modalities that have been found are displayed. The search closes once a level is found that contains no new distinct modalities.

The relationship between distinct modalities found by the above procedure can be given further structure by discovering the relationships between the modalities under the entailment relation ('$\vdash$') of the system. The Interderivability button is used for this purpose (figure 6.3 on page 169). The environment prompts for a list of formulas and constructs a matrix indicating their relationships under derivability. This is shown in figure 3.11 for the case of S4. The matrix is also displayed as a graph and can be interpreted as shown in figure 3.12. The diagram splits into two parts: one for positive modalities and one for the negative modalities.

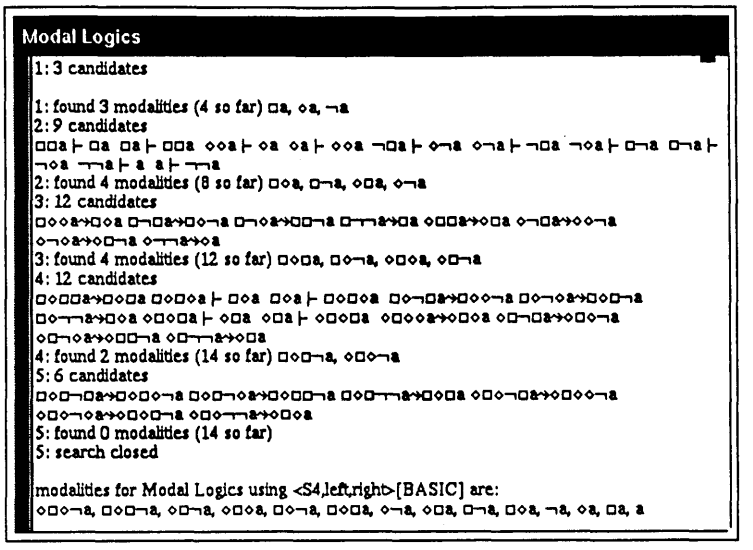


Figure 3.10: Modalities search for classical S4

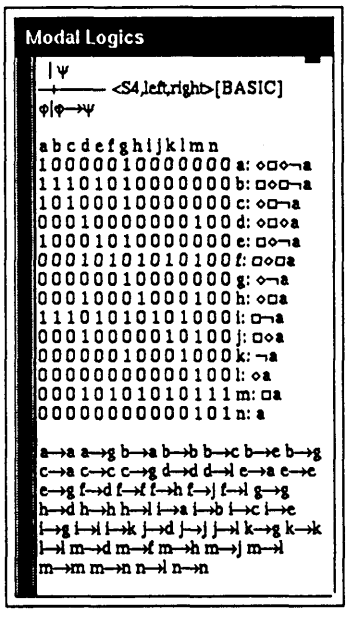


Figure 3.11: Interderivability of modalities for classical S4

3.3.7 Classical S5

Figure 3.13 shows a similar analysis, derived from the environment, for the system S5. The system has fewer distinct modalities than S4 of which it is an extension. It can be shown that the modalities collapse from the diagram given for S4 to that given for S5 accordingly

Positive			*Negative*		
◇□,□◇□	→	□	◇□¬,□◇□¬	→	□¬
□◇,◇□◇	→	◇	□◇¬,◇□◇¬	→	◇¬

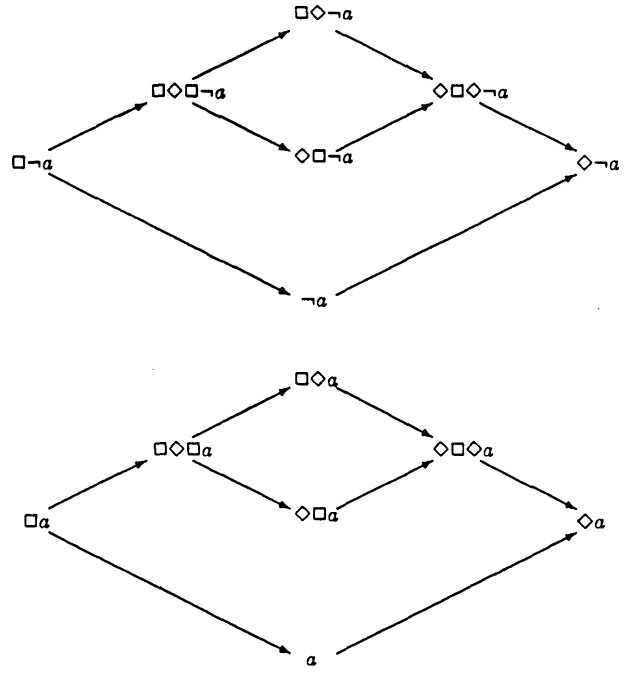


Figure 3.12: Modality Diagram for S4

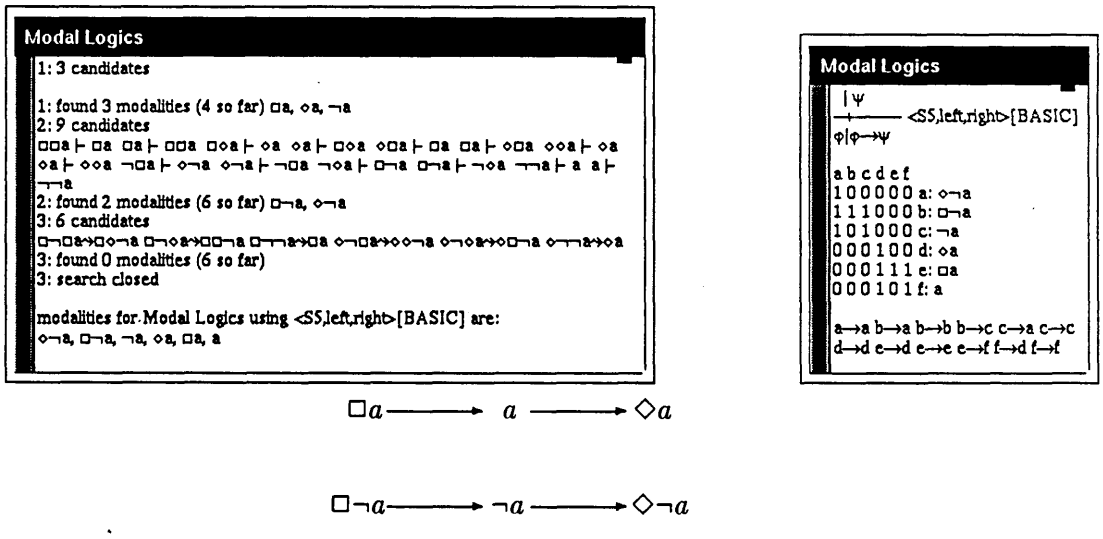


Figure 3.13: Classical S5

3.3.8 Intuitionistic Modal Systems

Intuitionistic modal systems are constructed in a manner similar to the classical systems presented earlier; they are however, restricted to single-conclusioned intuitionistic sequents and intuitionistic rules for negation. The modal rules used are shown in figure 3.14, and are taken together with the rules for the intuitionistic presentation

given in the previous chapter and summarised on page 54. The classical rule of double negation elimination does not hold in this system, which means we should expect a greater number of distinct modalities for the intuitionistic versions of the classical systems. In fact, this presentation produces a collection of 31 modalities as shown in figure 3.15. There is also an interesting diagram of the modalities, which is derived from figure 3.16 and shown in figure 3.17.

Systems	*Rule*
K	$\dfrac{\Gamma, \Gamma' \vdash \varphi}{\Box\Gamma, \neg\Box\Gamma', \Delta \vdash \Box\varphi}$
T, S4, S5	$\dfrac{\varphi, \Gamma \vdash \psi}{\Box\varphi, \Gamma \vdash \psi}$
S4	$\dfrac{\Box\Gamma \vdash \varphi}{\Box\Gamma, \Delta \vdash \Box\varphi}$
S5	$\dfrac{\Box\Gamma, \neg\Box\Gamma' \vdash \varphi}{\Box\Gamma, \neg\Box\Gamma', \Delta \vdash \Box\varphi}$

Figure 3.14: Rules for intuitionistic modal family

It is worth noting that if double negation elimination is introduced into the system the modalities are reducible to the 14 of S4.

Positive	*Negative*
$\neg\neg a \rightarrow -$	
$\neg\Box\neg\Box\neg\Box\neg a, \Box\neg\Box\neg\Box a, \neg\neg\Box\neg\Box\neg\Box a, \neg\neg\Box a \rightarrow \Box$	$\Box\neg a, \neg\neg\Box\neg a \rightarrow \Box\neg$
$\neg\Box\neg a \rightarrow \Diamond$	$\neg\Box a, \neg\Box\neg\neg a \rightarrow \Diamond\neg$
$\Box\neg\Box\neg a, \neg\neg\Box\neg\Box\neg a \rightarrow \Box\Diamond$	$\Box\neg\Box\neg\neg a, \Box\neg\Box a, \neg\neg\Box\neg\Box a, \neg\neg\Box\neg\Box\neg\neg a \rightarrow \Box\Diamond\neg$
$\neg\Box\neg\Box a, \neg\Box\neg\Box\neg\neg a \rightarrow \Diamond\Box$	$\neg\Box\neg\Box\neg a \rightarrow \Diamond\Box\neg$
$\Box\neg\neg a, \Box\neg\Box\neg\Box\neg\neg a, \neg\neg\Box\neg\Box\neg\Box\neg\neg a, \neg\neg\Box\neg\Box\neg\neg a \rightarrow \Box\Diamond\Box$	$\Box\neg\Box\neg\Box\neg a, \neg\neg\Box\neg\Box\neg\Box\neg a \rightarrow \Box\Diamond\Box\neg$
$\neg\Box\neg\Box\neg\Box\neg a \rightarrow \Diamond\Box\Diamond$	$\neg\Box\neg\Box\neg\Box a, \neg\Box\neg\Box\neg\Box\neg\neg a \rightarrow \Diamond\Box\Diamond\neg$

Similarly, the intuitionistic version of S5 reduces from 13 modalities to the 6 ones of

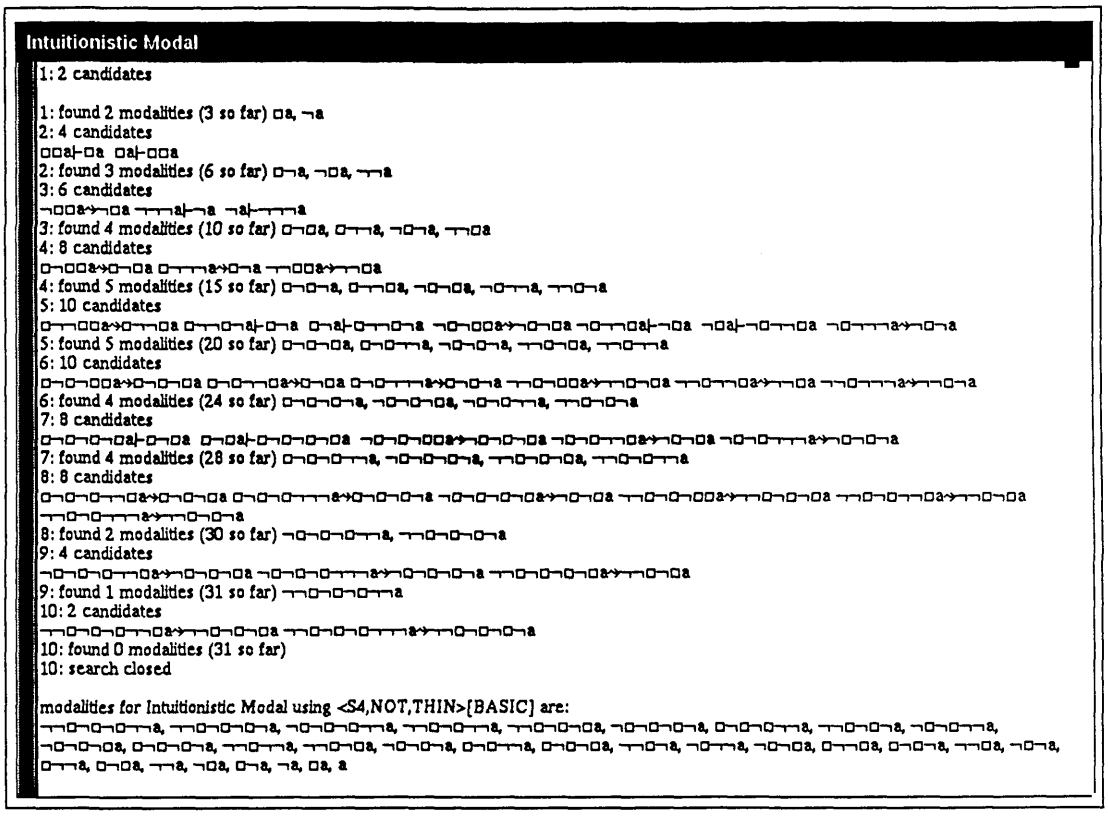


Figure 3.15: Modalities search for modal system IS4

the classical system.

Positive			*Negative*	
	$\neg\neg$	$\rightarrow$	$-$	
$\Box\neg\neg\Box, \Box\neg\neg, \neg\neg\Box\neg\neg, \neg\neg\Box$	$\rightarrow$	$\Box\neg\neg\Box\neg$	$\rightarrow$	$\Box\neg$

3.3.9 Conclusion

The modalities search described here, provides considerable insight into the structure of a modal system, especially when it is combined with a check for interderivability of the distinct modalities. The information obtained from these searches can build a user's confidence that the all-introduction presentation concerned captures some of the structure required. The interaction of modalities with other connectives can be explored in an similar way.

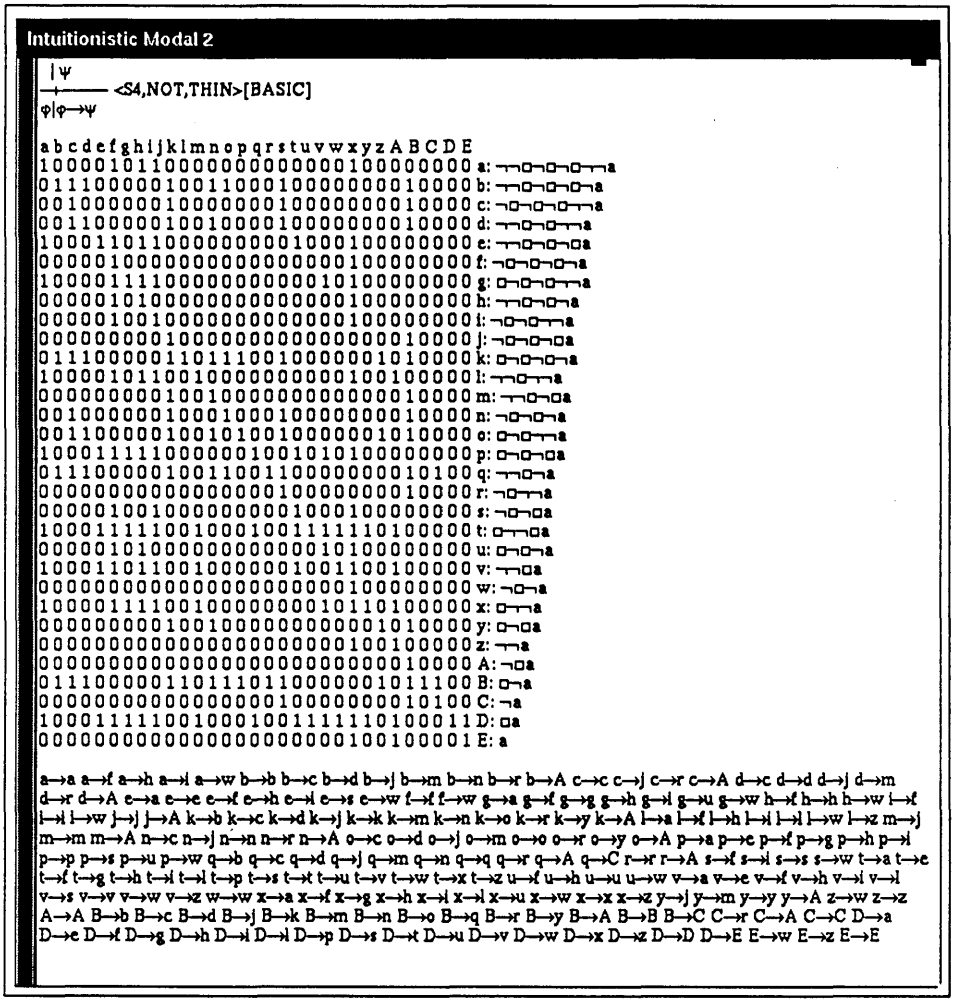


Figure 3.16: Interderivability search among modalities in the system IS4

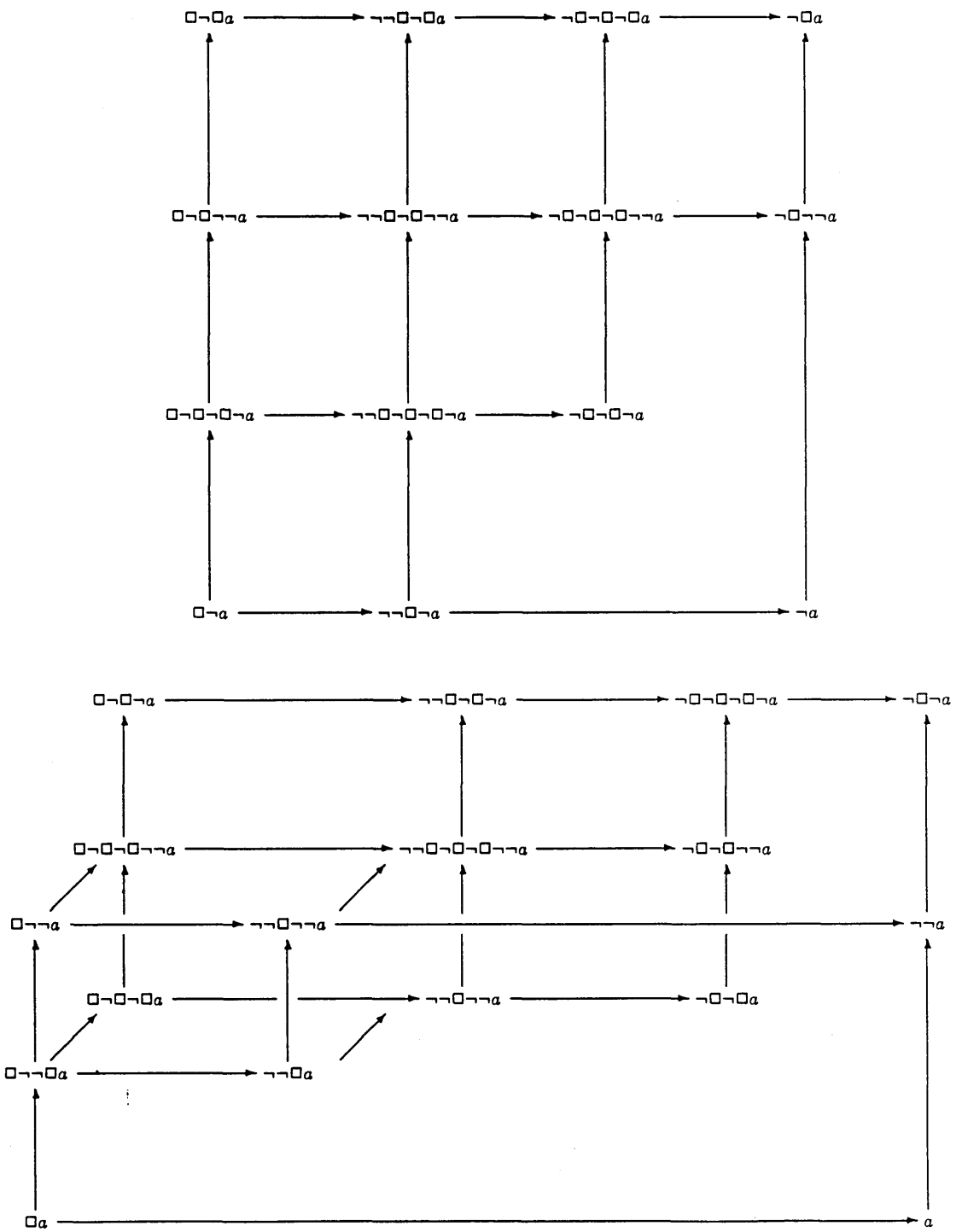


Figure 3.17: Modality diagram for IS4

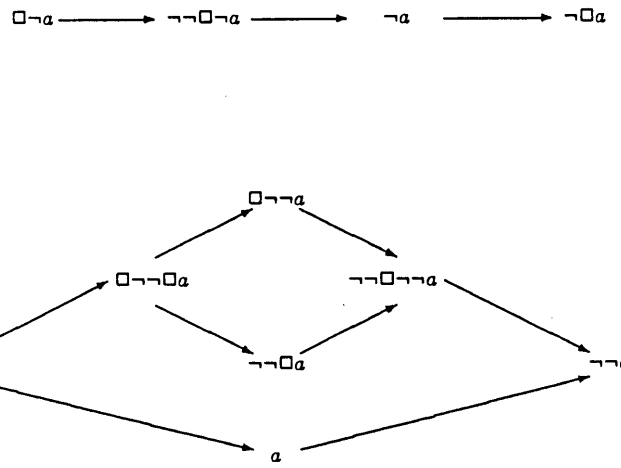


Figure 3.18: Modality diagram for IS5

3.4 Logics of Knowledge and Belief

Logics of Knowledge and Belief are used to model the reasoning of independent systems or individuals about themselves and each other. In such systems, a modality is indexed by the agents that are capable of performing deductions. In this way a family of related modalities is introduced.

Given a collection of agents or individuals I with sets of beliefs, then the notion of an individual i believing some fact φ may be modelled as the modality '$B\langle a\rangle\varphi$'. This gives rise to a family of operators indexed by the individuals concerned whose behaviour is defined to be like '$\Box$'. Beliefs can be nested; for example, the fact "a believes that b believes φ" may be represented as '$B\langle a\rangle B\langle b\rangle\varphi$'.

The rules shown below will keep the modalities separate although there is no essential need for them to do so. For example there may be a rule that captures '$\vdash B\langle a\rangle\varphi\rightarrow B\langle a\rangle B\langle a\rangle\varphi$'; this is a variant of the rule '4' (in §3.2.8.2 above). This is sometimes called "positive introspection". Note that the rule must prohibit '$\vdash B\langle i\rangle\varphi\rightarrow B\langle j\rangle B\langle i\rangle\varphi$' for $i \neq j$, which would have the effect of everyone believing everybody else's beliefs.

The definition of a logic of belief is achieved in the environment by providing it a with language and rules as follows; an additional syntactic category is added to the language for individuals.

3.4.1 Language of a Logic of Belief

Syntactic categories are: Sequent, Formula, Individual.

Judgement is Sequent.

Formation Rules are

Formula

'B' '⟨' *i*:Individual '⟩' *a*:Formula	→	BELIEVES(*i*,*a*)
a:Formula '∧' *b*:Formula	→	AND(*a*,*b*)
a:Formula '∨' *b*:Formula	→	OR(*a*,*b*)
a:Formula '→' *b*:Formula	→	IMPLIES(*a*,*b*)
a:Formula '↔' *b*:Formula	→	IFF(*a*,*b*)
'¬' *a*:Formula	→	NOT(*a*)
t:^[A-Z]$ *i*:Individual	→	PROP(*t*,*i*)

Individual

i:^[abc]$	→	ID(*i*)

Sequent

a:SetOf(Formula) '⊢' *b*:SetOf(Formula)	→	SEQ(*a*,*b*)

3.4.2 Inference Rules

In addition to the usual rules for classical logic given in §2.8.6, add the following rule:

$$\frac{\Gamma \vdash \varphi}{\Gamma', B\langle i\rangle\Gamma \vdash B\langle i\rangle\varphi, \Delta}\ B\langle i\rangle$$

which is a variant of the rule for K given above. The individuals index the modality B. Informally, this rule says that to conclude that an individual a believes some fact φ, we must also make any facts - here the Γ's - which might entail φ part of a's set of beliefs: hence $B\langle a\rangle\Gamma$.

It is also possible to add the rule:

$$\frac{B\langle i\rangle\Gamma \vdash \varphi}{\Gamma', B\langle i\rangle\Gamma \vdash B\langle i\rangle\varphi, \Delta}\ B4\langle i\rangle$$

which is like rule 4 above. This expresses the axiom '$\vdash B\langle i\rangle\varphi \rightarrow B\langle i\rangle B\langle i\rangle\varphi$' that if i believes some fact φ then i *believes* he believes it.

3.4.3 Wise-man Example

The following example is adapted from [GN87], p. 215, where it is specialised to two wise-men from many for brevity:

> *Suppose there are two wise men who are told by their king that at least one of them has a white spot on his forehead; actually both of them have white spots on their foreheads. We assume that each wise man can see the other but not his own forehead. Thus each knows whether the other has a white spot. Suppose that the first wise man 'a' says "I do not know whether I have a white spot" then the second wise man 'b' can conclude that he also has a white spot on his forehead.*

Analysing this informal description of the problem, and allowing that wise-men reason perfectly and can see all the consequences of a set of assumptions, yields the following more formal description. 'Wi' is taken to mean that the wise-man called 'i' has a white spot on his forehead.

1. If a believes that a does not have a white spot on his forehead then b believes that a does not have a white spot either. $B\langle a\rangle\neg Wa \rightarrow B\langle b\rangle\neg Wa$

2. a and b both know that either one of them must have a spot, so a believes that b believes that either a or b have a white spot on their foreheads. $B\langle a\rangle B\langle b\rangle(Wa \vee Wb)$

3. b says that he does not know if he has a white spot, and therefore a believes that b does not believe he has a white spot on his forehead. $B\langle a\rangle \neg B\langle b\rangle Wb$

4. *Conclusion:* a believes that he has a white spot on his forehead. $B\langle a\rangle Wa$

The following is the proof tree obtained for the above argument when it is phrased as a sequent in the environment.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{Wa \vdash Wb, Wa}{\neg Wa, Wa \vdash Wb}\,\neg\vdash \quad \neg Wa, Wb \vdash Wb}{Wa \vee Wb, \neg Wa \vdash Wb}\,\vee\vdash}{B\langle b\rangle Wa \vee Wb, B\langle b\rangle \neg Wa \vdash Wa, B\langle b\rangle Wb}\,B\langle b\rangle}{B\langle b\rangle Wa \vee Wb, \neg B\langle b\rangle Wb, B\langle b\rangle \neg Wa \vdash Wa}\,\neg\vdash \quad \dfrac{\dfrac{B\langle b\rangle Wa \vee Wb, Wa \vdash Wa, B\langle b\rangle Wb}{B\langle b\rangle Wa \vee Wb \vdash Wa, \neg Wa, B\langle b\rangle Wb}\,\vdash\neg}{B\langle b\rangle Wa \vee Wb, \neg B\langle b\rangle Wb \vdash Wa, \neg Wa}\,\neg\vdash}{\neg Wa \rightarrow B\langle b\rangle \neg Wa, B\langle b\rangle Wa \vee Wb, \neg B\langle b\rangle Wb \vdash Wa}\,\rightarrow\vdash}{\underbrace{B\langle a\rangle \neg Wa \rightarrow B\langle b\rangle \neg Wa}_{1}, \underbrace{B\langle a\rangle B\langle b\rangle Wa \vee Wb}_{2}, \underbrace{B\langle a\rangle \neg B\langle b\rangle Wb}_{3} \vdash \underbrace{B\langle a\rangle Wa}_{4}}\,B\langle a\rangle$$

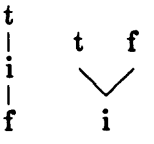
Note that there are two applications of the epistemic rule $B\langle i\rangle$, one for each if a and b. The proof tree above each of the applications amounts to reasoning done by a and b using their separate sets of beliefs.

3.4.4 Conclusion

The example shows how modalities can be indexed by individuals, allowing "multi-modal" systems to be expressed within the environment. Modal action logics can also be presented in a similar fashion. In these there are actions, rather than individuals, and the actions may have a more complex structure than the constants used for the individuals above.

3.5 Three and more valued logics

Many-valued logics arose in the 1920s as a means of addressing problems with "paradoxes", and the consideration of whether propositions such as '$\varphi \vee \neg\varphi$' should be universally true - as they are in classical logic. Many arguments have been found for rejecting the two-valued view that classical logic bestows upon this proposition. For example, consider the case when the proposition, φ, is taken to mean *"the defendant is guilty"*, and given to a jury in a court of law. Under Scottish law, the jury's answer need not be just "yes" or "no", as it is in England, but may also be "not proven". Here, "not proven" is taken to mean that insufficient evidence was given to decide the guilt (**t**) or the innocence (**f**) of the defendant. If a truth-value, **i** is to be assigned to "not proven" it will lie between or below the others:



Examples of three-valued logics found in Computer Science focus on the possibility of a computation failing to terminate for some input. The second of the two figures may be taken to represent the possible outcomes of running a semi-decision procedure for some property. At some time after the procedure has been started, either the procedure has terminated having decided the property and returning **t** or **f**, or else it is "still computing" or **i**. If the observer is Zeus, to whom time means nothing, then **i** is a legitimate truth-value, but for earthly observers the process might terminate at the next moment and so still has the *potential* on becoming **t** or **f**.

Cliff Jones' version of the Vienna Definition Method (VDM) uses a Logic of Partial Functions (LPF) [Che86] to address the issue of functions failing to denote values. LPF is similar to Kleene's strong three-valued system (see below) plus an operation, called Δ, which determines whether a proposition is defined

$$\Delta(\varphi) = \begin{cases} \mathbf{f} & \text{if } \varphi \text{ undefined} \\ \mathbf{t} & \text{otherwise} \end{cases}$$

3.5.1 Valuation Systems

Consider the system containing a single category S, populated with a number of constant symbols P, and with formation rules given by a collection of operations $O = \langle o_1, \ldots, o_n \rangle$.

A valuation system V is a triple $\langle M, D, F \rangle$ where

- M is a set with at least two "truth-values"
- D is a non-empty set of designated truth-values $D \subset M$. These are intended to distinguish those truth values that are "true" from those which are not true, since we may have gradations of truth in the many-valued setting. For example, the following choices may be made for the example above:

$$D = \begin{cases} \{\mathbf{t}\} & \text{"definitely true"} \\ \{\mathbf{t},\mathbf{i}\} & \text{"possibly true"} \\ \{\mathbf{t},\mathbf{f}\} & \text{"defined"} \end{cases}$$

- F is a set of functions corresponding to the operators in the language, such that $F = \{f_1, \ldots, f_n\}$ and $f_i : M^{d_i} \to M$, where d_i is the arity of the operator o_i. The functions f_i encode the truth tables for their o_i.

Let $a : P \to M$ be an assignment mapping propositions to truth values in M. Then functions in F induce a valuation $V_a : S \to M$ on the sentences as follows:

$$\begin{aligned} V_a(p) &= a(p) && \text{if } p \in P \\ V_a(o_i(x_1, \ldots, x_{d_i})) &= f_i(V_a(x_1), \ldots, V_a(x_{d_i})) \end{aligned}$$

The role of D is to determine the validity of a sentence in the language. A sentence is valid if all assignments of truth-values to its propositions force the sentence to take a designated value under the valuation

$$\vdash \varphi \text{ iff } \forall a. V_a(\varphi) \in D$$

So for the case of classical logic, take $M = \{\mathbf{t},\mathbf{f}\}$ and $D = \{\mathbf{t}\}$ with the familiar truth tables shown in figure 3.19 for the operators $O = \langle \neg, \wedge, \vee, \to \rangle$ respectively.

	ψ	t	f	t	f	t	f
φ	$\neg\varphi$	$\varphi\wedge\psi$		$\varphi\vee\psi$		$\varphi\to\psi$	
t	f	t	f	t	t	t	f
f	t	f	f	t	f	t	t

Figure 3.19: Classical two-valued truth-tables

Now suppose that it is wished to model *underdetermination* in the sense given above. Take $M = \{\mathbf{t},\mathbf{i},\mathbf{f}\}$ and $D = \{\mathbf{t}\}$ where $\mathbf{i}$ is viewed as truth-value lying below $\mathbf{t}$ and $\mathbf{f}$. The truth tables of Kleene's strong[2] three-valued logic are shown in table 3.20. In this system, $\mathbf{i}$ is intended to represent 'undecided' in the sense used in the introduction.

[2]the *weak* version takes the value $\mathbf{i}$ whenever $\mathbf{i}$ heads a row or a column.

	ψ	t	i	f	t	i	f	t	i	f
φ	$\neg\varphi$	$\varphi\wedge\psi$			$\varphi\vee\psi$			$\varphi\rightarrow\psi$		
t	f	t	i	f	t	t	t	t	i	f
i	i	i	i	f	t	i	i	t	i	i
f	t	f	f	f	t	i	f	t	t	t

Figure 3.20: Kleene's three-valued logic

3.5.2 Many-valued all-introduction systems

The following technique is offered as a means of encoding a finite many-valued system in the "sequent" all-introduction setting preferred here. The technique presented here is similar to a tableaux encoding of multi-valued systems in [Car87].

Given such a valuation system, a "generalised" sequent presentation may be formulated in the following way. Define a generalised sequent as an n-tuple, $\Gamma_1|\Gamma_2|\cdots|\Gamma_n$, where $n = |M|$ and the Γ's are sets of formulas in the language. For the case when $n = 2$, write $\Gamma \vdash \Delta$ as before. The intention is that each 'slot' Γ_i should correspond with a truth-value in M. It is assumed that the correspondence of a truth-value with its position in the sequent is given by some map $g : M \rightarrow \{1, \ldots, n\}$.

3.5.2.1 Axioms

Now for each $i, j \in M, i \neq j$ let

$$\frac{}{\Gamma_1|\cdots|\varphi,\Gamma_i|\cdots|\varphi,\Gamma_j|\cdots|\Gamma_n}$$

be an axiom.

Taking a refutational account, the sequent system can be regarded as formulating the systematic search for a counterexample. In this context, an axiom states that if the proposition, φ, is simultaneously assigned the two different truth-values - there is no counterexample in this case (as the induced valuation is not single-valued on φ).

3.5.2.2 Logical Rules

Next the logical rules for each connective o_i may be constructed as follows. Let the inverse of o_i's associated function f_i be $f_i^{-1}(m) = \{\langle u_1, \ldots, u_{d_i}\rangle | f(u_1, \ldots, u_{d_i}) = m\}$.

Now consider the behaviour of f^{-1} at each point m in M. If $f_i^{-1}(m)$ is empty then add the axiom

$$\frac{}{\Gamma_1|\cdots|o_i(\varphi_1,\ldots,\varphi_{d_i}),\Gamma_{g(m)}|\cdots|\Gamma_n}$$

This axiom states that it is inconsistent (or "inconceivable") to have a formula with such an assignment of truth-values. Certainly, no counterexample can be constructed when this is the case.

Otherwise, if $f_i^{-1}(m) = X$ is not empty, a logical rule is defined whose antecedents, S_i, are generalised sequents formed from the tuples in X.

$$\frac{S_1 \quad \cdots \quad S_{|X|}}{\Gamma_1|\cdots|o_i(\varphi_1,\ldots,\varphi_{d_i}),\Gamma_{g(m)}|\cdots|\Gamma_n}$$

Supposing that $\langle u_1,\ldots,u_{d_i}\rangle$ is such a tuple then a generalised sequent, S, is added. S is constructed by taking those of the u_i which equal m for each $m \in M$ and adding the corresponding φ_i to that part of the sequent indexed by m. This can be restated as: if $o_i(\varphi_1,\ldots,\varphi_{d_i})$ is to be assigned the truth value m, then each of its constituent subformulas $\varphi_1,\ldots,\varphi_{d_i}$ must be assigned the following truth values. When there are several permutations of assignments, *i.e.* $|X| > 1$, these are considered 'in parallel'. In this way all possibilities of discovering a counterexample are explored.

Example

In the case of classical logic, there are two truth-values, $\mathbf{t}$ and $\mathbf{f}$, of which, $\mathbf{t}$ is designated. Therefore a two sided-sequent is sufficient.

The labelling function g takes $\mathbf{t} \mapsto 1$ and $\mathbf{f} \mapsto 2$, which can be more succinctly written as $\Gamma_{\mathbf{t}} \vdash \Gamma_{\mathbf{f}}$.

To see how the construction of logical rules works, consider the truth-table for conjunction (figure 3.19). There is one subcase for each truth-value. In this example these are given by $f_\wedge^{-1}(\mathbf{t}) = \{\langle \mathbf{t},\mathbf{t}\rangle\}$ and $f_\wedge^{-1}(\mathbf{f}) = \{\langle \mathbf{t},\mathbf{f}\rangle,\langle \mathbf{f},\mathbf{t}\rangle,\langle \mathbf{f},\mathbf{f}\rangle\}$. If the construction presented above is applied to each of these sets of pairs, the following logical rules are obtained.

$$\frac{\varphi,\psi,\Gamma\vdash\Delta}{\varphi\wedge\psi,\Gamma\vdash\Delta}\,f_\wedge^{-1}(\mathbf{t}) \qquad \frac{\varphi,\Gamma\vdash\psi,\Delta \quad \psi,\Gamma\vdash\varphi,\Delta \quad \Gamma\vdash\varphi,\psi,\Delta}{\Gamma\vdash\varphi\wedge\psi,\Delta}\,f_\wedge^{-1}(\mathbf{f})$$

The first is recognised as the familiar rule $\wedge\vdash_C$, but the second is rather more complicated than expected. A small observation can be used to simplify the second rule. By introducing an "intermediate truth-value" called '$-$', $= \{\mathbf{t},\mathbf{f}\} = M$, which stands for either of the other two values, $f_\wedge^{-1}(\mathbf{f})$ may be rephrased as $\{\langle \mathbf{f},-\rangle,\langle -,\mathbf{f}\rangle\}$ as follows

$$f_{\wedge}^{-1}(\mathbf{f}) = \{\underbrace{\langle \mathbf{t},\mathbf{f}\rangle, \langle \mathbf{f},\mathbf{f}\rangle}_{\langle -,\mathbf{f}\rangle}\underbrace{\langle \mathbf{f},\mathbf{t}\rangle, \langle \mathbf{f},\mathbf{f}\rangle}_{\langle \mathbf{f},-\rangle}\}$$

which gives the rule

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta}\, f_{\wedge}^{-1}(\mathbf{f})$$

The intermediate truth-value can be interpreted as saying that the actual truth-value that is assigned to that component has *no effect* on the truth-value of the formula as a whole. Hence, it can be dropped from the corresponding upper-sequent.

3.5.3 Examples of the method

If this analysis is repeated for the remaining classical truth-tables of figure 3.19, the familiar sequent calculus presentation of Gentzen (developed in §2.8.6) is obtained.

3.5.3.1 A logic of non-termination

Table 3.21 captures the behaviour of call-by-value evaluation of boolean formulas. The system models the case when a formula is evaluated in a left to right manner, and only those parts that are required to decide whether the truth-value of the formula is 't' or 'f' are evaluated. This is often the case in computer languages, where the use of such a strategy leads to a cleaner coding style by, for example, avoiding the need for an extra conditional on loops. An erroneous or non-terminating result is represented by the 'ω' truth-value. For example, in these tables '$\mathbf{t} \vee \omega = \mathbf{t}$', rather than the corresponding 'ω' in Kleene's system above. The designated truth-values of the system are taken to be $D = \{\mathbf{t}\}$.

3.5.3.2 Kleene's strong three-valued logic

If the method of §3.5.2 is applied to the 3-valued truth-table shown in figure 3.20 the result is the rules given in figure 3.23. (Some of the upper-sequents have been stacked vertically to save some width.) The assignment of truth-values to positions in the sequent is $\Gamma_{\mathbf{t}}|\Gamma_{\mathbf{i}}|\Gamma_{\mathbf{f}}$.

The two sided sequent $\Gamma \vdash_{KI} \varphi$ is defined as holding iff $\Gamma||\varphi$ and $\Gamma|\varphi|$ both hold. *i.e.*

$$\frac{\Gamma||\varphi \quad \Gamma|\varphi|}{\Gamma \vdash_{KI} \varphi}$$

That is, taking a refutational account, it is not possible to construct a counter-example which is consistent with the assignment of a designated value to the assumptions and a non-designated value to the conclusions.

	ψ	t	ω	f	t	ω	f	t	ω	f
φ	$\neg\varphi$	$\varphi\wedge\psi$			$\varphi\vee\psi$			$\varphi\rightarrow\psi$		
t	f	t	ω	f	t	t	t	t	ω	f
ω	ω	ω	ω	ω	ω	ω	ω	ω	ω	ω
f	t	f	f	f	t	ω	f	t	t	t

Figure 3.21: A logic of non-termination

$$\frac{}{\varphi,\Gamma|\varphi,\Delta|\Theta} \qquad \frac{}{\varphi,\Gamma|\Delta|\varphi,\Theta} \qquad \frac{}{\Gamma|\varphi,\Delta|\varphi,\Theta}$$

$$\frac{\varphi,\psi,\Gamma|\Delta|\Theta}{\varphi\wedge\psi,\Gamma|\Delta|\Theta}\wedge|| \qquad \frac{\Gamma|\varphi,\Delta|\Theta \quad \varphi,\Gamma|\psi,\Delta|\Theta}{\Gamma|\varphi\wedge\psi,\Delta|\Theta}|\wedge| \qquad \frac{\Gamma|\Delta|\varphi,\Theta \quad \varphi,\Gamma|\Delta|\psi,\Theta}{\Gamma|\Delta|\varphi\wedge\psi,\Theta}||\wedge$$

$$\frac{\varphi,\Gamma|\Delta|\Theta \quad \psi,\Gamma|\Delta|\varphi,\Theta}{\varphi\vee\psi,\Gamma|\Delta|\Theta}\vee|| \qquad \frac{\Gamma|\varphi,\Delta|\Theta \quad \Gamma|\psi,\Delta|\varphi,\Theta}{\Gamma|\varphi\vee\psi,\Delta|\Theta}|\vee| \qquad \frac{\Gamma|\Delta|\varphi,\psi,\Theta}{\Gamma|\Delta|\varphi\vee\psi,\Theta}||\vee$$

$$\frac{\Gamma|\Delta|\varphi,\Theta \quad \varphi,\psi,\Gamma|\Delta|\Theta}{\varphi\rightarrow\psi,\Gamma|\Delta|\Theta}\rightarrow|| \qquad \frac{\Gamma|\varphi,\Delta|\Theta \quad \varphi,\Gamma|\psi,\Delta|\Theta}{\Gamma|\varphi\rightarrow\psi,\Delta|\Theta}|\rightarrow| \qquad \frac{\varphi,\Gamma|\Delta|\psi,\Theta}{\Gamma|\Delta|\varphi\rightarrow\psi,\Theta}||\rightarrow$$

$$\frac{\Gamma|\Delta|\varphi,\Theta}{\neg\varphi,\Gamma|\Delta|\Theta}\neg|| \qquad \frac{\Gamma|\varphi,\Delta|\Theta}{\Gamma|\neg\varphi,\Delta|\Theta}|\neg| \qquad \frac{\varphi,\Gamma|\Delta|\Theta}{\Gamma|\Delta|\neg\varphi,\Theta}||\neg$$

Figure 3.22: Rules for non-termination logic

For example $\vdash_{Kl} a\rightarrow a$ iff $|a\rightarrow a|$ and $||a\rightarrow a$ is attempted as follows,

$$\dfrac{\dfrac{\dfrac{}{a||a}}{||a\rightarrow a} \qquad \dfrac{\dfrac{}{a|a|} \quad |a| \quad \dfrac{}{|a|a}}{|a\rightarrow a|}}{\vdash_{Kl} a\rightarrow a}$$

but the second derivation does not go through, because $|a|$ is not an axiom, and no further progress can be made.

This derivation points to a more general property of Kleene's system, namely that

$$\forall\varphi.\ \nvdash_{Kl} \varphi$$

To see this, consider the sequent $|\varphi|$. There are four cases depending on the structure

$$\frac{}{\varphi, \Gamma|\varphi, \Delta|\Theta} \qquad \frac{}{\varphi, \Gamma|\Delta|\varphi, \Theta} \qquad \frac{}{\Gamma|\varphi, \Delta|\varphi, \Theta}$$

$$\frac{\Gamma|\Delta|\varphi, \Theta}{\neg\varphi, \Gamma|\Delta|\Theta}\,\neg|| \qquad \frac{\Gamma|\varphi, \Delta|\Theta}{\Gamma|\neg\varphi, \Delta|\Theta}\,|\neg| \qquad \frac{\varphi, \Gamma|\Delta|\Theta}{\Gamma|\Delta|\neg\varphi, \Theta}\,||\neg$$

$$\frac{\varphi, \Gamma|\Delta|\Theta \quad \psi, \Gamma|\Delta|\Theta}{\varphi\vee\psi, \Gamma|\Delta|\Theta}\,\vee|| \qquad \frac{\begin{array}{l}\Gamma|\varphi, \Delta|\psi, \Theta\\ \Gamma|\varphi, \psi, \Delta|\Theta\\ \Gamma|\psi, \Delta|\varphi, \Theta\end{array}}{\Gamma|\varphi\vee\psi, \Delta|\Theta}\,|\vee| \qquad \frac{\Gamma|\Delta|\varphi, \psi, \Theta}{\Gamma|\Delta|\varphi\vee\psi, \Theta}\,||\vee$$

$$\frac{\varphi, \psi, \Gamma|\Delta|\Theta}{\varphi\wedge\psi, \Gamma|\Delta|\Theta}\,\wedge|| \qquad \frac{\begin{array}{l}\psi, \Gamma|\varphi, \Delta|\Theta\\ \Gamma|\varphi, \psi, \Delta|\Theta\\ \varphi, \Gamma|\psi, \Delta|\Theta\end{array}}{\Gamma|\varphi\wedge\psi, \Delta|\Theta}\,|\wedge| \qquad \frac{\Gamma|\Delta|\varphi, \Theta \quad \Gamma|\Delta|\psi, \Theta}{\Gamma|\Delta|\varphi\wedge\psi, \Theta}\,||\wedge$$

$$\frac{\psi, \Gamma|\Delta|\Theta \quad \Gamma|\Delta|\varphi, \Theta}{\varphi\rightarrow\psi, \Gamma|\Delta|\Theta}\,\rightarrow|| \qquad \frac{\begin{array}{l}\varphi, \Gamma|\psi, \Delta|\Theta\\ \Gamma|\varphi, \psi, \Delta|\Theta\\ \Gamma|\varphi, \Delta|\psi, \Theta\end{array}}{\Gamma|\varphi\rightarrow\psi, \Delta|\Theta}\,|\rightarrow| \qquad \frac{\varphi, \Gamma|\Delta|\psi, \Theta}{\Gamma|\Delta|\varphi\rightarrow\psi, \Theta}\,||\rightarrow$$

Figure 3.23: Ternary sequent rules for Kleene's 3-valued logic

of φ: '$\neg a$', '$a\wedge b$', '$a\vee b$', and '$a\rightarrow b$'. In each case, by applying the middle rules of figure 3.23, the sequent is reduced:

$$\frac{|a|}{|\neg a|} \qquad \frac{|b|a \quad |a, b| \quad a|b|}{|a\wedge b|} \qquad \frac{|a|b \quad |a, b| \quad |b|a}{|a\vee b|} \qquad \frac{a|b| \quad |a, b| \quad |a|b}{|a\rightarrow b|}$$

As each case gives rise to a branch of the form '$|\Delta|$', there will be a branch that will never be closed by an axiom. Hence, this system has no theorems of the form $\vdash_{Kl} \varphi$.

For a successful derivation, consider $a, b \vdash_{Kl} a\wedge b$

$$\dfrac{\dfrac{\dfrac{}{a, b||a} \quad \dfrac{}{a, b||b}}{a, b||a\wedge b} \qquad \dfrac{\dfrac{}{a, b|a|} \quad \dfrac{}{a, b|a, b|} \quad \dfrac{}{a, b|b|a}}{a, b|a\wedge b|}}{a, b \vdash_{Kl} a\wedge b}$$

3.5.3.3 Łukasiewicz's three-valued logic

Lukasiewicz's three-valued logic was developed to deal with future contingent statements. It differs from Kleene's by its matrix for implication which is shown in figure 3.24. The leading diagonal is entirely '**t**'s whence $\vdash_{\mathrm{L}} \varphi \rightarrow \varphi$. The motivation here is to regard '**i**' as an intermediate *fractional* truth value *i.e.* $\mathbf{i} = \frac{1}{2}$, whereas $\mathbf{t} = 1$ and $\mathbf{f} = 0$. Then $\varphi \rightarrow \psi$ is defined as

$$\varphi \rightarrow \psi = \begin{cases} \neg\varphi \vee \psi & \varphi > \psi \\ 1 & \text{otherwise} \end{cases}$$

where $\neg\varphi = 1 - \varphi$, $\varphi \vee \psi = \max(\varphi, \psi)$, and $\varphi \wedge \psi = \min(\varphi, \psi)$.

ψ	t	i	f		
φ	$\varphi \rightarrow \psi$			$\Box\varphi$	$\Diamond\varphi$
t	t	i	f	t	t
i	t	t	i	f	t
f	t	t	t	f	f

Figure 3.24: Łukasiewicz's three-valued logic

To adapt the rules for this system, two of the three rules for $\rightarrow$ in table 3.23 must be replaced with the following:

$$\frac{\varphi, \Gamma|\Delta|\Theta \quad \Gamma|\Delta|\psi, \Theta \quad \Gamma|\varphi, \psi, \Delta|\Theta}{\varphi \rightarrow \psi, \Gamma|\Delta|\Theta} \rightarrow||'$$

$$\frac{\varphi, \Gamma|\psi, \Delta|\Theta \quad \Gamma|\varphi, \Delta|\psi, \Theta}{\Gamma|\varphi \rightarrow \psi, \Delta|\Theta} |\rightarrow|'$$

Now $\vdash_{\mathrm{L}} a \rightarrow a$ can be derived as follows:

$$\dfrac{\dfrac{\dfrac{}{a|a|} \quad \dfrac{}{|a|a}}{|a \rightarrow a|} \quad \dfrac{\dfrac{}{a||a}}{||a \rightarrow a}}{\vdash_{\mathrm{L}} a \rightarrow a}$$

3.5.4 Kleene's system revisited

Examination of the three-valued matrices for the connectives and their negations shows that it is possible to find a more elegant set of Gentzen rules for Kleene's system, in the spirit of [Avr88].

	t i f	t i f	t i f	ψ	
φ	$\varphi\wedge\psi$	$\varphi\vee\psi$	$\varphi\rightarrow\psi$	φ	$\neg\varphi$
t	t i f	t t t	t i f	t	f
i	i i f	t i i	t i i	i	i
f	f f f	t i f	t t t	f	t
φ	$\neg(\varphi\wedge\psi)$	$\neg(\varphi\vee\psi)$	$\neg(\varphi\rightarrow\psi)$	$\neg\neg\varphi$	$\neg\varphi$
t	f i t	f f f	f i t	t	f
i	i i t	f i i	f i i	i	i
f	t t t	f i t	f f f	f	t
	f i t	f i t	f i t	$\neg\psi$	

The following table is obtained by reading the above truth-tables with the truth-values identified as follows:

$$\{\mathbf{f},\mathbf{i}\} = \mathbf{if} \quad \{\mathbf{t},\mathbf{i}\} = \neg\mathbf{if}$$
$$\{\mathbf{t}\} = \mathbf{t} \quad \{\mathbf{f}\} = \neg\mathbf{t}$$

Note that $\neg\neg\mathbf{t} = \mathbf{if}$ and $\neg\neg\mathbf{if} = \mathbf{t}$.

ψ	t	if	$\neg$t	$\neg$if	t	if	$\neg$t	$\neg$if	t	if	$\neg$t	$\neg$if
φ			$\wedge$				$\vee$				$\rightarrow$	
t	t	if			t	t			t	if	$\neg$if	$\neg$t
if	if	if			t	if			t	$\neg$if	$\neg$if	$\neg$if
$\neg$t			$\neg$t	$\neg$if			$\neg$t	$\neg$t				
$\neg$if			$\neg$if	$\neg$if			$\neg$t	$\neg$if				

The technique introduced above is next applied to this revised truth-table but making use of $\neg$ to reduce a 4-place sequent to a 2-place sequent. The resulting set of rules is shown in table 3.25.

By examining the rules introducing connectives on the right-hand side of the sequent it is clear that once again, no theorems of the form $\vdash\varphi$ will provable using this presentation. The rules ensure that the 'parity' of the sub-formulas of φ is always preserved, *i.e.* sub-formulas of a formula occurring on the right-hand side are placed on the right-hand side, similarly for rules introducing connectives on the left. This means that the basic sequent will never be derivable if all formulas occur solely on the right-hand side.

$$\varphi, \Gamma \vdash \varphi, \Delta \qquad \varphi, \neg\varphi, \Gamma \vdash \Delta$$

$$\frac{\varphi, \Gamma \vdash \Delta}{\neg\neg\varphi, \Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \varphi, \Delta}{\Gamma \vdash \neg\neg\varphi, \Delta}$$

$$\frac{\varphi, \psi, \Gamma \vdash \Delta}{\varphi \wedge \psi, \Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \varphi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta}$$

$$\frac{\neg\varphi, \Gamma \vdash \Delta \quad \neg\psi, \Gamma \vdash \Delta}{\neg(\varphi \wedge \psi), \Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \neg\varphi, \neg\psi, \Delta}{\Gamma \vdash \neg(\varphi \wedge \psi), \Delta}$$

$$\frac{\psi, \Gamma \vdash \Delta \quad \varphi, \Gamma \vdash \Delta}{\varphi \vee \psi, \Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \varphi \vee \psi, \Delta}$$

$$\frac{\neg\varphi, \neg\psi, \Gamma \vdash \Delta}{\neg(\varphi \vee \psi), \Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \neg\varphi, \Delta \quad \Gamma \vdash \neg\psi, \Delta}{\Gamma \vdash \neg(\varphi \vee \psi), \Delta}$$

$$\frac{\neg\varphi, \Gamma \vdash \Delta \quad \psi, \Gamma \vdash \Delta}{\varphi \rightarrow \psi, \Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \neg\varphi, \psi, \Delta}{\Gamma \vdash \varphi \rightarrow \psi, \Delta}$$

$$\frac{\varphi, \neg\psi, \Gamma \vdash \Delta}{\neg(\varphi \rightarrow \psi), \Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \neg\psi, \Delta \quad \Gamma \vdash \varphi, \Delta}{\Gamma \vdash \neg(\varphi \rightarrow \psi), \Delta}$$

Figure 3.25: Negated Kleene Rules

Lukasiewicz's implication connective (figure 3.24) cannot be captured in this two-sided form as there is no means of allowing the central point of the truth-table to be anything other than 'i'.

3.5.5 Conclusions

This section has presented a scheme for translating finite many-valued systems presented through valuation systems and truth-tables into equivalent all-introduction presentations using generalised sequents. Given such presentations, some properties of the many-valued system become more perspicuous, although occasionally at the cost of a more lengthy presentation. Some optimisations of the translation are possible through the introduction of "don't care" values, shown in the classical logic example.

3.6 Hypersequents

3.6.1 Introduction

Hypersequents provide an example where it is useful to be able to nest structural properties, which is possible using the linguistic structures available in the environment. Hypersequent systems were introduced independently by Pottinger [Pot83] for the presentation of cut-free modal systems using symmetry in their reachability relations *e.g.* S5. They were also introduced by Avron [Avr88] for the treatment of Lukasiewicz's three valued-logic and a relevance logic called RM_3.

A hypersequent is simply a collection of sequents. In presentations of hypersequent systems in the literature, the structural rules are duplicated and divided into *internal* and *external* rules. The external rules act on the hypersequents, and the internal rules act on the sequents which make up the hypersequents. In the environment, the use of appropriate implicit declarations makes the presentation very natural and quite succinct.

This can be illustrated with a reformulation of Pottinger's hypersequent presentation of the modal systems T, S4, and S5.

3.6.2 Pottinger's Modal systems

The presentation defines a new category of *hypersequents* which become the judgement category.

'[' *a*:Sequents ']' → HYPERSEQUENT(*a*)

Figure 3.26: Formation rule for the Hypersequent category

A modal sequent is valid with respect to a class of Kripke frames, $\mathcal{C}$, when it is true in all models based on them, *i.e.*

$\varphi_1, \ldots, \varphi_n \vdash \psi_1, \ldots, \psi_m$ is valid in $\mathcal{C}$ iff $\forall \mathcal{F} \in \mathcal{C}. \forall V. \langle \mathcal{F}, V \rangle. \vdash \varphi_1 \wedge \cdots \wedge \varphi_n \rightarrow \psi_1 \vee \cdots \vee \psi_m$

Pottinger extends this be saying that a hypersequent is valid with respect to a class of frames, precisely when one (or more) of its constituent sequents is valid:

$$[\Gamma_1 \vdash \Delta_1; \cdots; \Gamma_n \vdash \Delta_n] \text{ is valid iff } \exists i.\Gamma_i \vdash \Delta_i \text{ is valid}$$

The advantage of using a hypersequent representation rather than simple sequents, is that a form of the subformula property can be obtained. This was not always possible in the single sequent formulation. In the rules given below, the rule responsible for S5, *S5-BOX* of figure 3.28, inserts the modal formula into its upper sequents. These sequents inherit the formula, and as this is a subformula of the lower sequent, the subformula property is maintained across the rule. A derivation of the sequent '$\Box\neg a \vee \Box\neg\Box\neg a$' is shown in figure 3.29. Figure 3.30 shows the results of the modality analysis of §3.3 for this system.

3.6.3 Summary of the presentation of the system

- The categories used are : Hypersequent, Formula, Sequent, Sequents.
- The judgement categories are: Sequent, Hypersequent.
- The side-condition '**hyper** ($\Box\varphi$) Σ Σ'' is used to insert a formula into the left-hand side of all the sequents of a hypersequent and is used in the rule *S5-BOX*.
- The metavariables used are as follows:

Formula	$\varphi\psi$
SetOf(Sequent)	Σ
SetOf(Formula)	$\Gamma\Delta$

- The rule *seq* is used to form a hypersequent from a single sequent.
- Formation rules for the other categories are shown in figure 3.27.
- The propositional rules have been lifted to hypersequents, *e.g.*

$$\frac{\varphi, \Gamma \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi} \quad \text{becomes} \quad \frac{[\varphi, \Gamma \vdash \psi, \Delta; \Sigma]}{[\Gamma \vdash \varphi \rightarrow \psi, \Delta; \Sigma]} \rightarrow\vdash$$

Formation Rules

'[' *a*:Sequents ']'	→	HYPERSEQUENT(*a*):Hypersequent
a:SetOf(Sequent)[a]	→	SEQS(*a*):Sequents
a:SetOf(Formula) '⊢' *b*:SetOf(Formula)	→	SEQ(*a*, *b*):Sequent
'□' *a*:Formula	→	BOX(*a*):Formula
a:Formula '∧' *b*:Formula	→	AND(*a*, *b*):Formula
a:Formula '∨' *b*:Formula	→	OR(*a*, *b*):Formula
a:Formula '→' *b*:Formula	→	IMPLIES(*a*, *b*):Formula
a:Formula '↔' *b*:Formula	→	IFF(*a*, *b*):Formula
'¬' *a*:Formula	→	NOT(*a*):Formula
t:^[a-z]$	→	ID(*t*):Formula

[a]whose elements are separated with a ';'

Figure 3.27: Formation rules for Hypersequent example

$$\frac{}{[\varphi, \Gamma \vdash \varphi, \Delta; \Sigma]}\ BASIC \qquad \frac{[\Gamma \vdash \Delta]}{\Gamma \vdash \Delta}\ seq$$

$$\frac{[\Box\Gamma, \Gamma' \vdash \Delta; \Box\Gamma \vdash \varphi; \Sigma]}{[\Box\Gamma, \Gamma' \vdash \Box\varphi, \Delta; \Sigma]}\ S4\text{-}S5\text{-}BOX \qquad \frac{[\Box\Gamma, \Gamma' \vdash \Delta; \Gamma \vdash \varphi; \Sigma]}{[\Box\Gamma, \Gamma' \vdash \Box\varphi, \Delta; \Sigma]}\ T\text{-}BOX$$

$$\frac{[\varphi, \Box\varphi, \Gamma \vdash \Delta; \Sigma]}{[\Box\varphi, \Gamma \vdash \Delta; \Sigma]}\ S4\text{-}T\text{-}BOX \qquad \frac{[\varphi, \Box\varphi, \Gamma \vdash \Delta; \Sigma']}{[\Box\varphi, \Gamma \vdash \Delta; \Sigma]}\ \mathtt{hyper}\ (\Box\varphi)\ \Sigma\ \Sigma'\ \ S5\text{-}BOX$$

$$\frac{[\Gamma \vdash \varphi, \Delta; \Sigma] \quad [\Gamma \vdash \psi, \Delta; \Sigma]}{[\Gamma \vdash \varphi \wedge \psi, \Delta; \Sigma]}\ \vdash\wedge \qquad \frac{[\varphi, \psi, \Gamma \vdash \Delta; \Sigma]}{[\varphi \wedge \psi, \Gamma \vdash \Delta; \Sigma]}\ \wedge\vdash$$

$$\frac{[\Gamma \vdash \varphi, \psi, \Delta; \Sigma]}{[\Gamma \vdash \varphi \vee \psi, \Delta; \Sigma]}\ \vee\vdash \qquad \frac{[\varphi, \Gamma \vdash \Delta; \Sigma] \quad [\psi, \Gamma \vdash \Delta; \Sigma]}{[\varphi \vee \psi, \Gamma \vdash \Delta; \Sigma]}\ \vdash\vee$$

$$\frac{[\varphi, \Gamma \vdash \psi, \Delta; \Sigma]}{[\Gamma \vdash \varphi \to \psi, \Delta; \Sigma]}\ \to\vdash \qquad \frac{[\psi, \Gamma \vdash \Delta; \Sigma] \quad [\Gamma \vdash \varphi, \Delta; \Sigma]}{[\varphi \to \psi, \Gamma \vdash \Delta; \Sigma]}\ \vdash\to$$

$$\frac{[\varphi, \Gamma \vdash \Delta; \Sigma]}{[\Gamma \vdash \neg\varphi, \Delta; \Sigma]}\ \neg\vdash \qquad \frac{[\Gamma \vdash \varphi, \Delta; \Sigma]}{[\neg\varphi, \Gamma \vdash \Delta; \Sigma]}\ \vdash\neg$$

Figure 3.28: Hypersequent rules for the modal systems T, S4 and S5

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{[\Box\neg a\vdash; \Box\neg a, \neg a\vdash; \Box\neg a, \neg a\vdash\neg a]}{[\Box\neg a\vdash\neg a; \Box\neg a\vdash; \Box\neg a, \neg a\vdash]}\ S4\text{-}S5\text{-}BOX}{[\vdash\neg a; \vdash; \Box\neg a\vdash]}\ S4\text{-}S5\text{-}BOX}{[\vdash\neg\Box\neg a; \vdash\neg a; \vdash]}\ \vdash\neg}{[\vdash\neg\Box\neg a; \vdash\Box\neg a]}\ S5\text{-}BOX}{[\vdash\Box\neg\Box\neg a, \Box\neg a]}\ S5\text{-}BOX}{[\vdash\Box\neg a\vee\Box\neg\Box\neg a]}\ \vdash\vee}{\vdash\Box\neg a\vee\Box\neg\Box\neg a}\ seq$$

Figure 3.29: Example derivation of the Euclidean Axiom

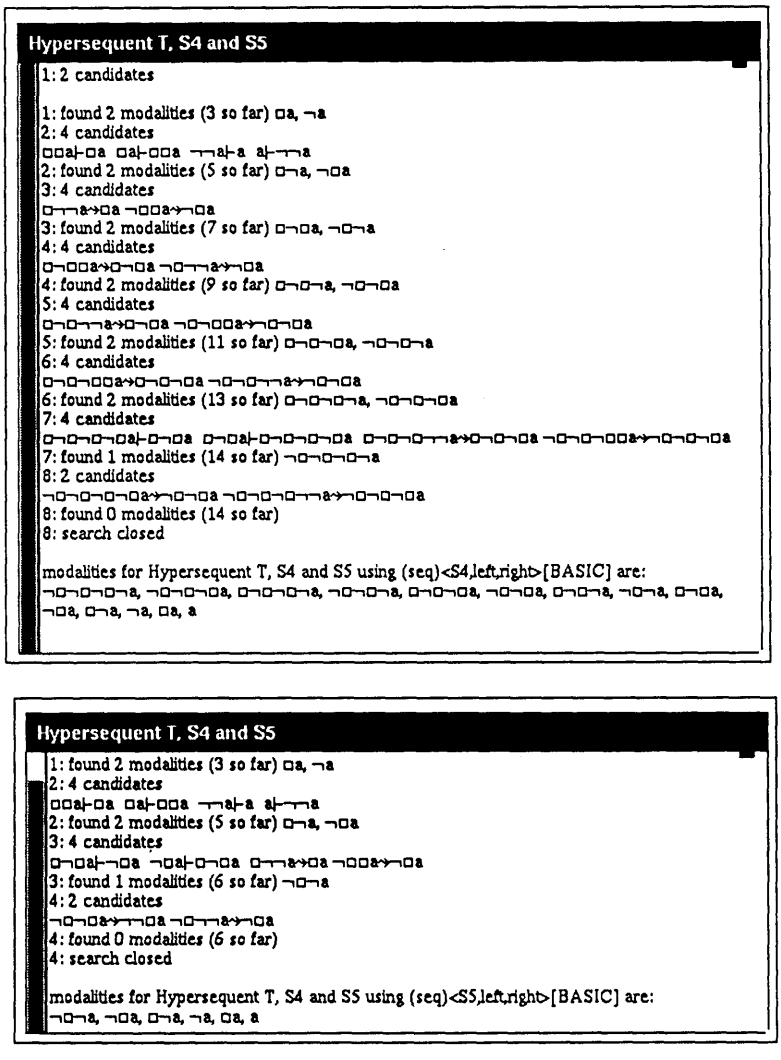


Figure 3.30: Modalities derived for hypersequent presentations of S4 and S5

3.7 Defeasible Reasoning

This section illustrates the use of *negative judgements* to model a simple non-monotonic system. The system consists of classical logic augmented by a new non-monotonic operator called *unless*: the formula '$\varphi \downarrow \psi$' is read as "φ holds *unless* ψ can be shown". The negative content of the unless operator is captured by the use of a negative judgement.

In addition to the usual sequent judgement, '$\vdash$', the new judgement '$\nvdash$' is introduced, the latter being declared as a *negative* judgement in the language of the system. Figure 3.31 gives details of the language declaration used.

```
Category: Formula[φψ] SetOf(Formula)[ΓΔ] Sequent NSequent;
Judgement: Sequent;
Negative judgement: NSequent;
a:SetOf(Formula) '⊢' b:SetOf(Formula) → SEQ(a,b):Sequent
a:SetOf(Formula) '⊬' b:SetOf(Formula) → NSEQ(a,b):NSequent
a:Formula '↓' b:Formula → UNLESS(a,b)
a:Formula '∧' b:Formula → AND(a,b):Formula
a:Formula '∨' b:Formula → OR(a,b):Formula
a:Formula '→' b:Formula → IMPLIES(a,b):Formula
'¬' a:Formula → NOT(a):Formula
a:"[a-z][a-z0-9]*" → ID(a):Formula
```

Figure 3.31: Defeasible system's language

A negative judgement is interpreted negatively when it occurs in the upper part of an inference rule. The negative interpretation is simply to *negate* the outcome of the derivation of the judgement. Thus when a negative judgement leads to a successful derivation the result is considered to be unsuccessful. Similarly, if a negative judgement leads to an unsuccessful derivation, then the outcome is considered satisfactory. In this way a negative judgement can be used to invert the sense of derivability. As the outcome of the derivation is inverted, there is no witness to the successful derivation of such a judgement, and the proof tree simply indicates the negated judgement in the same way as a basic sequent.

All-introduction Rules for Defeasible System

Classical rules (from §2.8.6) +

$$\frac{\Gamma \nvdash \psi, \Delta \quad \Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \downarrow \psi \vdash \Delta} \downarrow\vdash \qquad \frac{\Gamma \vdash \Delta}{\Gamma \nvdash \Delta}\, neg$$

The inference rules for the example non-monotonic system are shown above. The rule $\downarrow\vdash$ is concerned with the introduction of the unless operator '$\downarrow$' as an assumption. The formula '$\varphi \downarrow \psi$' (read as "φ holds *unless* it can be shown that ψ holds") can be introduced as an assumption in an argument of the form '$\Gamma, \varphi \vdash \Delta$', in place of '$\varphi$', when it is known that 'ψ' is not provable.

The rule *neg* does not have a very satisfactory reading, as it appears that $\nvdash$ and $\vdash$ are the same relation. The reason for this is that the negation of $\nvdash$ is implicit, and is declared in the language part, rather than in the rules of the system. The rule *neg* simply states that '$\nvdash$' *uses* the definitions of the judgement '$\vdash$'. Other ways of expressing this 'dual' relationship might be preferable, but it was decided that the additional syntax required to make the distinction in the definition of the rules, rather than of the language, might diminish the freedom of a user to choose the syntactic forms for a system.

Other rules included in the system are taken from the classical system of §2.8.6 on page 57.

Example derivations

A frequent example used in the discussion of non-monotonic systems relates to a bird called *tweety* [KLM90]. It is common knowledge that birds can fly '$b \rightarrow f$', and also that penguins cannot '$p \rightarrow (\neg f)$'. The idea that a bird can fly unless it is a penguin can be expressed as '$(b \rightarrow f) \downarrow p$'. Given that tweety is a bird, then without any other knowledge about tweety it is reasonable to assume that tweety can fly 'f'. However if the additional fact that tweety is a penguin is also available then it is clear that the conclusion should be that tweety cannot fly '$\neg f$'. Naturally, when this is the case 'f' should not be derivable.

The system permits the following derivations for the first two of these three problems.

1. It is not known that tweety is a penguin, and since tweety is a bird, it can fly (by default): $b,(b\rightarrow f)\downarrow p,p\rightarrow(\neg f)\vdash f$

$$\dfrac{\dfrac{\dfrac{\dfrac{f\vdash f \qquad b\vdash b}{b\rightarrow f,b\vdash p,f}\rightarrow\vdash}{b\nvdash f,p \qquad b\rightarrow f,b\vdash p,f}\downarrow\vdash}{b,(b\rightarrow f)\downarrow p\vdash f}\neg\vdash}{b,(b\rightarrow f)\downarrow p,\neg f\vdash f} \qquad \dfrac{p\rightarrow\neg f,b\nvdash f,p \qquad \dfrac{f\vdash f \qquad b\vdash b}{b\rightarrow f,p\rightarrow\neg f,b\vdash f,p}\rightarrow\vdash}{p\rightarrow\neg f,b,(b\rightarrow f)\downarrow p\vdash f,p}\downarrow\vdash$$

$$\dfrac{b,(b\rightarrow f)\downarrow p,\neg f\vdash f \qquad p\rightarrow\neg f,b,(b\rightarrow f)\downarrow p\vdash f,p}{b,(b\rightarrow f)\downarrow p,p\rightarrow\neg f\vdash f}\rightarrow\vdash$$

2. Adding the additional fact that tweety is a penguin gives a different outcome. Tweety is a bird and a penguin, entails tweety *cannot* fly: $b,(b\rightarrow f)\downarrow p,p,p\rightarrow(\neg f)\vdash\neg f$

$$\dfrac{\dfrac{\dfrac{f\vdash f}{b,(b\rightarrow f)\downarrow p,p,f,\neg f\vdash}\neg\vdash \qquad p\vdash p}{b,(b\rightarrow f)\downarrow p,p,p\rightarrow\neg f,f\vdash}\rightarrow\vdash}{b,(b\rightarrow f)\downarrow p,p,p\rightarrow\neg f\vdash\neg f}\vdash\neg$$

So $\vdash$ is a non-monotonic relation, and the rule

$$\dfrac{\Gamma\vdash\Delta}{\Gamma,\varphi\vdash\Delta}M$$

does not hold in this system.

3. It is also important that, when tweety is a bird and a penguin and can fly, $b,(b\rightarrow f)\downarrow p,p,p\rightarrow(\neg f)\vdash f$ is *not* derivable.

Chapter 4

Quantification

This chapter shows how quantification is treated within the environment. Quantification is represented using *binders* which provide scoping that enables distinctions to be made between free and bound variables. A simple extension to the syntax of inference rules is used to formalise the substitution of arbitrary terms and eigenvariables for the variables governed by binders.

4.1 Introduction

Predicate logics extend the propositional systems studied in previous chapters by including quantifiers among their logical connectives. Intuitively, quantifiers introduce a "universe of discourse" within which propositions become predicates. In this way, quantifiers may range over individuals denoted by *terms*, as in *first-order* calculi.

Predicate variables are introduced in *second-order* systems, as predicates themselves become objects in the universe of discourse, and may become the objects of quantifiers. Also interesting from a Computer Science perspective are type inference systems which lie between first and second-order logics. In these systems, which make assignments of types to programs, the types are described by propositions rather than predicates; these may similarly be the objects of quantifiers and an example of this is shown below. Types are also used in many-sorted logics where they annotate variables to restrict the individuals that a variable may denote.

First-order predicate calculus illustrates the additional syntactic machinery required to add quantification to propositional logic. In particular the following are needed:

1. *quantifiers*, given as operators $[\forall - .-, \exists - .-, \ldots]$ over *variables* and *formulas*;

2. *variables*, $[x, y, \ldots]$ over which the quantification is performed;
3. *predicates*, $[P, Q, \ldots]$ are propositions parameterised by a sequence of terms; and
4. *terms*, $[t, t_1, \ldots]$ the syntactic structures which denote elements in some appropriate "universe of discourse"

   A *term* is defined as follows:

   - a *constant* symbol $[a, b, c, \ldots]$ is a term;
   - a *variable* is a term; and
   - if $t_1 \cdots t_n$ are terms and f is an n-place *function* symbol then $f(t_1, \ldots, t_n)$ is a term.

For example '$\forall x.P(a, f(x))$' is a sentence when: P is a predicate of arity two, a is a constant, f is a one-place function, and x is a variable whose occurrence in f is *bound* by the quantifier $\forall$. (An example of a second-order quantification would be '$\forall x.\forall P.P(x, a) \rightarrow R(x)$' in which P is quantified over.)

A variable is said to be *bound* when it lies within the scope of a quantifier which refers to it. For example $\forall x.P(x) \wedge Q(y)$ binds the occurrence of x in the predicate P. A variable is said to be *free* when there is no such containing quantifier - as is the case for y in the example. When a variable lies within the influence of several quantifiers, the closest one is responsible for the binding.

4.1.1 The semantic perspective

The treatment of first-order systems using models mirrors the syntactic elements above in terms of sets and functions. Here, valuations are extended from the propositional case by adding a (non-empty) set D called the *domain*, or "universe of discourse". Constant symbols are assigned denotations in the domain using a map C. Similarly, each function symbol is associated with a function by F which maps its arguments to an element in the domain. Predicates are associated with their characteristic functions by P so that the resulting characteristic function determines for which tuples of elements in the domain the predicate holds.

These relationships are packaged as a structure $M = \langle D, C, F, P \rangle$. A valuation is used to specify which individual a term designates in an expression. The valuation, v, is a map from variables to an individuals in the domain. A first-order sentence

is satisfied with respect to a structure M and a valuation v, written as $M, v \models \varphi$, precisely when:

$$\begin{aligned}
M, v \models p(t_1, \ldots, t_n) &\text{ iff } \langle t_1(v), \ldots, t_n(v) \rangle \in P(p) \\
M, v \models \neg\varphi &\text{ iff it is not the case that } M \models \varphi \\
M, v \models \varphi \wedge \psi &\text{ iff } M, v \models \varphi \text{ and } M, v \models \psi \\
M, v \models \varphi \vee \psi &\text{ iff either } M, v \models \varphi \text{ or } M, v \models \psi \\
M, v \models \varphi \rightarrow \psi &\text{ iff when } M, v \models \varphi \text{ then } M, v \models \psi \\
M, v \models \forall x.\varphi &\text{ iff } M, v[x/d] \models \varphi \text{ for every } d \in D \\
M, v \models \exists x.\varphi &\text{ iff } M, v[x/d] \models \varphi \text{ for some } d \in D
\end{aligned}$$

where $v[x/d]$ stands for a map which takes x to the element d but otherwise behaves like v. The effect of the valuation on terms is given by:

$$x(v) = v(x) \qquad c(v) = C(c)$$

$$f(t_1, \ldots, t_n)(v) = F(f)(t_1(v), \ldots, t_n(v))$$

If the formula φ contains no free variables, $M \models \varphi$ suffices since the initial contents of v has no bearing on the outcome. The validity of such a closed sentence is given by its truth in all structures: $\models \varphi$ iff $M \models \varphi$ for all M.

4.1.2 The syntactic perspective

Computer based theorem provers for predicate systems use the results of Gentzen and Herbrand to reduce first-order sentences to propositional form by a systematic elimination of the quantifiers. Herbrand's insight was to regard combinations of constants and function symbols as 'names' for their denotations in the universe of discourse. The validity of a quantified formula could then be translated into a collection of essentially propositional formulas. Early attempts to do this on a computer by Davis and Putnam[DP60] and Prawitz et al[PHV60] enumerated these combinations of 'names' directly with all the combinatorial inefficiencies that this gave rise to.

Robinson's use of the *unification algorithm* [Rob65] greatly improved the efficacy of a search procedure by removing the need to enumerate or guess the names: it constructs the most general name that is suitable. The introduction of the unification algorithm went hand in hand with the introduction of *resolution*, which may be viewed as a return to axiomatic presentations using a single rule analogous to *modus ponens*.

For systems such as classical predicate logic, in which a prenex normal form is available, the deduction can be divided into two stages. Gentzen's *mid-sequent*

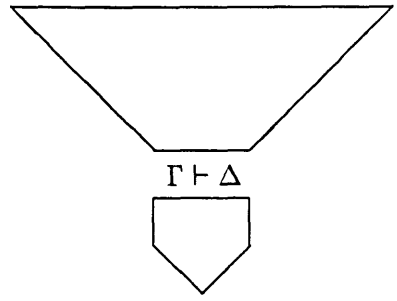


Figure 4.1: Mid-sequent of a proof

theorem shows that a proof of a sequent involving quantifiers can be transformed into a proof divided into two parts. The lower part contains rules concerned only with the quantifiers and structural properties such as contraction; the upper part is free of quantifiers and uses only propositional and structural rules. The dividing line between these two regions is the mid-sequent ($\Gamma \vdash \Delta$ in figure 4.1).

For other systems, such as Intuitionistic logic, this clean separation of quantifiers from the other logical connectives is not possible. In such systems, all rules must be considered together during the search for the proof.

To illustrate a typical set of quantifier rules, consider the following rules taken from a first-order all-introduction presentation of the intuitionistic predicate calculus in [Dum77].

All-introduction Rules for Quantifiers

$$\frac{\Gamma, \varphi(t) \vdash \psi}{\Gamma, \forall x.\varphi(x) \vdash \psi}\ \forall\vdash \qquad \frac{\Gamma \vdash \varphi(y)}{\Gamma \vdash \forall x.\varphi(x)}\ \vdash\forall$$

$$\frac{\Gamma, \varphi(y) \vdash \psi}{\Gamma, \exists x.\varphi(x) \vdash \psi}\ \exists\vdash \qquad \frac{\Gamma \vdash \varphi(t)}{\Gamma \vdash \exists x.\varphi(x)}\ \vdash\exists$$

Conditions

1. y is a variable and t is a term and both are free for x in $\varphi(x)$;
2. $\varphi(y)$ and $\varphi(t)$ result from $\varphi(x)$ by replacing every free occurrence of x by y and t respectively;
3. in the rules $\vdash\forall$ and $\exists\vdash$ the variable y does not occur free in '$\Gamma, \forall x.\varphi(x), \psi$' or in or '$\Gamma, \exists x.\varphi(x)$' respectively.

Remarks

The first condition ensures that the variable y cannot fall into the scope of a quantifier within which an occurrence of x may lie. This condition can be achieved by systematically renaming bound variables. The condition on t seeks to ensure that none of the variables in t can be captured in the same way.

The second condition formalises the notion of substitution; only the free occurrences can be changed by the substitution.

The final condition requires that the variable y be distinct from any free variable already occurring in the sequent. This ensures that the variable y is truly *arbitrary* and prevents invalid deductions in which a variable is used more than once.

When the structural rules of the system include the *contraction* rule quantifiers can be reused, *e.g.*

$$\dfrac{\dfrac{\Gamma, \forall x.\varphi(x), \varphi(t) \vdash \psi}{\Gamma, \forall x.\varphi(x), \forall x.\varphi(x) \vdash \psi}\,\forall\vdash}{\Gamma, \forall x.\varphi(x) \vdash \psi}\,\textit{contraction}$$

This is a source of great combinatorial complexity, as no upper bound can be placed on the size of the mid-sequent (when one exists).

4.2 Representation of quantification

This section provides details of the mechanisms used to represent quantification in the framework. In order to support the observations made in previous sections, a method must be supplied for defining quantifiers and using them in inference rules.

4.2.1 Extensions to language declaration

Variables which are subject to quantification are introduced to the framework by declaring a new syntactic category of variables. The variables are then used as the objects of *binders* which are in turn used to construct the quantifiers.

In order to use binders it is necessary to extend the technique used to specify languages which was described in Chapter 1. Two new operations are introduced in the language part:

- an operation 'Variable(–)' denotes a category of variables over a specified category. Variables constructed in this way can be scoped using a binder.

For example, the declaration Variable(Term) introduces variables that range over the category of terms. Metavariables are often also required, the following example

Category: Variable(Term)[xyz];

declares that bound variables are to be used over the category of Terms and defines the names of the metavariables [xyz] which will be used in the description of rules. An example of a rule using these metavariables is:

$$\frac{\Gamma, \varphi(t) \vdash \psi}{\Gamma, \forall x.\varphi(x) \vdash \psi}\ \forall\vdash$$

where t is a meta-variable over the category Term and $x(=\mathsf{x})$ is a meta-variable in the category of Variable(Term).

In the rule

$$\frac{\Gamma, \varphi(y) \vdash \psi}{\Gamma, \exists x.\varphi(x) \vdash \psi}\ \exists\vdash$$

both x and y are (distinct) meta-variables in the category of Variable(Term).

A means of forming object-level bound variables is also required for which the following example declaration can be used:

t:"[uvw]" → VAR(t):Variable(Term)

This indicates that variables over Terms are written as u, v, w, $u1$, $v1$, $w1$, ... This information is used to produce a concrete syntax for the user.

- The colon operator constructs the binding. Suppose v is a variable and $t_1, \ldots, t_n$ are terms to be bound by v, then the expression '$v : (t_1, \ldots, t_n)$' introduces the appropriate binder. When there is just one t, this can be written '$v : t$'. The following examples illustrate how binders are formed:

  Quantifiers The quantifiers given in the introduction for predicate logic can be defined as follows:

  '∀' a:Variable(Term) '.' t:Formula → FORALL(a:t):Formula
  '∃' a:Variable(Term) '.' t:Formula → EXISTS(a:t):Formula

  Abstraction (as in the lambda calculus)

  'λ' a:Variable(Term) '.' t:Term → ABS(a:t):Term

  declares an abstraction operator of the form $\lambda x.t$ assuming that x is in the category of Variable(Term) defined elsewhere and that t is a term. In the element 'ABS(a:t)' the 'a:t' part declares the binder. The notation also allows us to define (perhaps perversely) a infix operator *e.g.* $t\lambda x$ given by

t:Term 'λ' a:Variable(Term) → ABS(a:t):Term

Language for Predicate Calculus

The following example brings together the elements described above to define the language required for classical predicate logic. The boxed parts show the extensions required over the propositional logic used earlier.

Category: Formula[$\varphi\psi$] Sequent **Term[t] Variable(Term)[xyz]** SetOf(Formula)[$\Gamma\Delta$];
Judgement: Sequent;
a:SetOf(Formula) '⊢' b:SetOf(Formula) → SEQ(a,b):Sequent
'∀' a:Variable(Term) '.' t:Formula → FORALL(a:t):Formula
'∃' a:Variable(Term) '.' t:Formula → EXISTS(a:t):Formula
a:Formula '∧' b:Formula → AND(a,b):Formula
a:Formula '∨' b:Formula → OR(a,b):Formula
a:Formula '→' b:Formula → IMPLIES(a,b):Formula
'¬' a:Formula → NOT(a):Formula
p:"[A-Z][a-z0-9]*" '(' a:ListOf(Term) ')' → PRED(p,a):Formula
c:"[a-z][a-z]*" → CONST(c):Term
f:"[a-z][a-z]*" '(' a:ListOf(Term) ')' → FUN(f,a):Term
t:"[uvw]" → VAR(t):Variable(Term)

The boxed regions deal with, from the top: the declaration of the additional syntactic categories, the formation rules for the binders, and the formation rules for terms and their variables.

4.2.2 Rules that use binders

The goal is to provide a familiar notation for rules that introduce binders. This is done to make the framework intuitive to use. The notation used is similar to that used in the presentation of rules given for intuitionistic system in §4.1.2 above. Recall that a rule may have zero or more *upper sequents* and a single *lower sequent*:

$$\frac{\textit{upper sequent}(s)}{\textit{lower sequent}}$$

4.2.3 Binder in lower sequent

When a binder for a variable x occurs in the lower sequent of a rule, its *use* is indicated by writing '(x)' after the occurrence of the binder *e.g.* '$\forall x.\varphi(x)$'. The

substituands are indicated in upper sequents by writing '(u)' after occurrences of φ. Suppose that x is in the category Variable(c), then the mechanism used to interpret the formula depends on the category of 'u'.

- If u is a metavariable over the category c, then a brand-new meta-variable in the category c is substituted for x in φ.

$$\frac{\Gamma \vdash \varphi(t)}{\Gamma \vdash \exists x.\varphi(x)}$$

- If u is a metavariable over the same category as x, then a new name for x is selected and marked as depending on any existing variables in the lower sequent[1].

$$\frac{\Gamma \vdash \varphi(y)}{\Gamma \vdash \forall x.\varphi(x)}$$

  Supposing that metavariables a, b, c in the category over which x is a variable, are already used in a derivation, and that the new name for x is u, then the dependency of u on a, b, c is written as $u^{\{a,b,c\}}$. When no variables have been introduced, $u^{\{\}}$ is written.

4.2.4 Binder in an upper sequent

A binder can also be introduced in an upper sequent (even if it does not occur in the lower sequent). This is useful in forming the closure of an expression with respect to free occurrences of variables within it.

$$\frac{\Gamma : \forall x.u : v}{\Gamma : u(t) : v}\ Ga$$

The rule is only admissible when there are instances of the free variable t in the expression u that do not occur in other parts of the judgement *e.g.* Γ. The expression '$\forall x.u$' is the expression $u(t)$ where all occurrences of such a variable, t, have been replaced by the new bound variable x.

4.3 Deduction using binders

The interpretation of the use of binders in rules is enforced as follows. The condition (3) in §4.1.2 states that y does not occur free in the lower sequent of the rules for $\vdash\forall$ and $\exists\vdash$. The environment enforces this by forming a new constant which is

[1]This is similar to the approach used by early versions of the Isabelle system[Pau89]

indexed by the free variables in the lower sequent, *i.e.* $u^{\{a,b,c\}}$ above. This amounts to constructing a *Skolem function* from the current free variables to the arbitrary individual in question. The variable 'x' which is replaced is sometimes referred to as the *eigenvariable* of the rule.

The arbitrary nature of the t used in $\vdash\exists$ is represented by a *logical variable* in the implementation.

Unification Algorithm

It may be useful to state the unification algorithm here for reference. The unification algorithm takes a set of *disagreement pairs* S and tries to reduce them. Unify(u, v) is given as follows:

set S to $\{\langle u, v\rangle\}$
while S is not empty do
    remove a pair $\langle u, v\rangle$ from S
    if u is a variable then
        if u does not occur in v then let u be bound to v
        else report failure
    else if u is a function $f(s_1, \ldots, s_n)$ and v is a function $f(t_1, \ldots, t_n)$
        then add $\{\langle s_1, t_1\rangle, \ldots \langle s_n, t_n\rangle\}$ to S
    else if v is a variable then add $\langle v, u\rangle$ to S
    else report failure

The algorithm will report failure if it is not possible to reconcile all the disagreement pairs. If it succeeds, a sequence of *bindings* will have been made that have the effect of making u and v syntactically identical.

4.4 Examples of quantification rules

The use of the rules shown above is now illustrated with a few examples.

Example 1

Consider the two formulas:

$$\exists x.\forall y.P(x,y) \rightarrow \forall y.\exists x.P(x,y) \qquad \text{and} \qquad \forall y.\exists x.P(x,y) \rightarrow \exists x.\forall y.P(x,y)$$

The former is valid since, if there exists some x such that for all y the property holds, then certainly there is an x that satisfies the conclusion. However, the second formula is not valid as the xs may vary as a function of y in the antecedent. To see how the quantifier rules impose this restriction, consider the following proof trees:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P(u^{\{\}},a) \vdash P(b,v^{\{\}})}\, a\mapsto u^{\{\}}, b\mapsto v^{\{\}}}{P(u^{\{\}},a) \vdash \exists x.P(x,v^{\{\}})}\vdash\exists}{\forall y.P(u^{\{\}},y) \vdash \exists x.P(x,v^{\{\}})}\forall\vdash}{\forall y.P(u^{\{\}},y) \vdash \forall y.\exists x.P(x,y)}\vdash\forall}{\exists x.\forall y.P(x,y) \vdash \forall y.\exists x.P(x,y)}\exists\vdash}{\exists x.\forall y.P(x,y) \rightarrow \forall y.\exists x.P(x,y)}\vdash\rightarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{stuck!}}{P(u^{\{a,b\}},a) \vdash P(b,v^{\{a,b\}})}}{P(u^{\{a,b\}},a) \vdash \forall y.P(b,y)}\vdash\forall}{\exists x.P(x,a) \vdash \forall y.P(b,y)}\exists\vdash}{\exists x.P(x,a) \vdash \exists x.\forall y.P(x,y)}\vdash\exists}{\forall y.\exists x.P(x,y) \vdash \exists x.\forall y.P(x,y)}\forall\vdash}{\forall y.\exists x.P(x,y) \rightarrow \exists x.\forall y.P(x,y)}\vdash\rightarrow$$

The derivation of the first proposition is shown on the left. As the terms a, b are arbitrary, they are represented as *logical variables* and bound to their corresponding eigenvariables $v^{\{\}}, u^{\{\}}$. This is indicated by an appropriate label on the right of the basic rule, *e.g.*

$$\dfrac{}{P(u^{\{\}},a) \vdash P(b,v^{\{\}})}\, a\mapsto u^{\{\}}, b\mapsto v^{\{\}}$$

The derivation of the second proposition shown on the right is not successful. As the eigenvariables are eliminated by $\exists\vdash$ and $\vdash\forall$, they are indexed by the terms in use that the time they are created. In this case, $x \mapsto u^{\{a,b\}}$ and $y \mapsto v^{\{a,b\}}$ by the rules $\exists\vdash$ and $\vdash\forall$ respectively. This gives rise to the quantifier free sequent

$$P(u^{\{a,b\}},a) \vdash P(b,v^{\{a,b\}})$$

However, the unification of $v^{\{a,b\}}$ with a is prohibited, as is the unification of $u^{\{a,b\}}$ with b. Each would produce a *circular* term, and would therefore break the condition (3) of §4.1.2. This is precisely the role of the *occurs check* in the unification algorithm.

Example 2

Suppose a prenex intuitionistic formula is represented by the sequent

$$\vdash \forall x.\exists y.\forall x'.\exists y'.A(x',y',x,y)$$

where A is a quantifier-free formula. Applying the quantifier rules produces the following derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash A(v^{\{a\}}, b, u^{\{\}}, a)}{\vdash \exists y'.A(v^{\{a\}}, y', u^{\{\}}, a)}\vdash\exists}{\vdash \forall x'.\exists y'.A(x', y', u^{\{\}}, a)}\vdash\forall}{\vdash \exists y.\forall x'.\exists y'.A(x', y', u^{\{\}}, y)}\vdash\exists}{\vdash \forall x.\exists y.\forall x'.\exists y'.A(x', y', x, y)}\vdash\forall$$

at the top of which is an entirely propositional formula, $A(v^{\{a\}}, b, u^{\{\}}, a)$, which admits the following bindings:

	1	2	3	4
a	$u^{\{\}}$	$u^{\{\}}$	$v^{\{a\}}$	$v^{\{a\}}$
b	$u^{\{\}}$	$v^{\{a\}}$	$u^{\{\}}$	$v^{\{a\}}$

Only the first two bindings are valid assignments, since the others produce circular terms (consider $a \mapsto v^{\{a\}}$ in 3 and 4). Consequently, only the following two sentential forms are possible:

1. $A(v^{\{a\}}, u^{\{\}}, u^{\{\}}, u^{\{\}})$, and
2. $A(v^{\{a\}}, v^{\{a\}}, u^{\{\}}, u^{\{\}})$.

Only derivations of the quantifier-free part that use these assignments will be acceptable. As Dummett points out (in [Dum77], p. 150), this observation gives rise to a decision procedure for prenex formulas of intuitionistic predicate logic without function symbols, since the quantifiers cannot be reused.

Example 3

When several quantifier rules can be introduced at the same time, as in this example from classical logic (in [Gal87], p. 349), the order of introduction becomes significant. Consider the following sequent:

$$\vdash \exists x_1.\forall y_1.\neg P(x_1, y_1), \exists x_2.\forall y_2.\neg Q(z_2, y_2), \forall x_3.\exists y_3.\exists z_3.(P(x_3, y_3) \wedge Q(y_3, z_3))$$

Initially there is a choice of three candidate applications of quantifier rules out of a total of $\frac{7!}{2!2!3!} = 210$ different reduction paths; of these, just $\frac{4!}{2!2!} = 6$ lead to a

successful outcome, one of which is shown below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P(a,v^{\{a\}}) \vdash P(u^{\{\}},c)}\,{\scriptstyle a\mapsto u^{\{\}},c\mapsto v^{\{a\}}} \quad \dfrac{}{Q(b,w^{\{a,b\}}) \vdash Q(c,d)}\,{\scriptstyle b\mapsto c,d\mapsto w^{\{a,b\}}}}{P(a,v^{\{a\}}),Q(b,w^{\{a,b\}}) \vdash P(u^{\{\}},c)\wedge Q(c,d)}\,\vdash\wedge}{\vdash \neg P(a,v^{\{a\}}),\neg Q(b,w^{\{a,b\}}),P(u^{\{\}},c)\wedge Q(c,d)}\,\vdash\neg\times 2}{\vdash \neg P(a,v^{\{a\}}),\neg Q(b,w^{\{a,b\}}),\exists y_3.\exists z_3.P(u^{\{\}},y_3)\wedge Q(y_3,z_3)}\,\vdash\exists\times 2}{\vdash \neg P(a,v^{\{a\}}),\exists x_2.\forall y_2.\neg Q(x_2,y_2),\exists y_3.\exists z_3.P(u^{\{\}},y_3)\wedge Q(y_3,z_3)}\,\vdash\exists;\vdash\forall}{\vdash \exists x_1.\forall y_1.\neg P(x_1,y_1),\exists x_2.\forall y_2.\neg Q(z_2,y_2),\forall x_3.\exists y_3.\exists z_3.(P(u^{\{\}},y_3)\wedge Q(y_3,z_3))}\,\vdash\exists;\vdash\forall}{\vdash \exists x_1.\forall y_1.\neg P(x_1,y_1),\exists x_2.\forall y_2.\neg Q(z_2,y_2),\forall x_3.\exists y_3.\exists z_3.(P(x_3,y_3)\wedge Q(y_3,z_3))}\,\vdash\forall$$

Example 4

This example illustrates how the technique treats function symbols when they are present in a system. The language shown for first-order predicate calculus in §4.2.1 used the formation rule:

f:"[a-z][a-z]*" '(' a:ListOf(Term) ')' → FUN(f,a):Term

for function symbols. The following example illustrates a case where a, b are arbitrary terms which unify successfully:

$$\dfrac{\dfrac{\dfrac{}{P(f(a)) \vdash P(b)}\,{\scriptstyle b\mapsto f(a)}}{P(f(a)) \vdash \exists x.P(x)}\,\vdash\exists}{\forall x.P(f(x)) \vdash \exists x.P(x)}\,\forall\vdash$$

But, when $\exists$ is replaced with $\forall$, the following situation arises:

$$\dfrac{\dfrac{\text{stuck!}}{P(f(a)) \vdash P(u^{\{\}})}}{\forall x.P(f(x)) \vdash P(u^{\{\}})}\,\forall\vdash}{\forall x.P(f(x)) \vdash \forall x.P(x)}\,\vdash\forall$$

The derivation does not go through, as $f(a)$ and $u^{\{\}}$ fail to unify. (Recall that u is represented by a constant.) Intuitively, this should be expected as f might otherwise map all elements in the domain to a single element, and this would be excluded.

Example 5

In some derivations, intermediate bindings of terms to terms can occur. These bindings cause a variable to inherit dependencies on terms that may not have existed

at the point of the variable's introduction in the proof. Consider the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P(a)\vdash P(b)}\,{\scriptstyle a\mapsto b} \quad \dfrac{}{Q(u^{\{a\}})\vdash Q(b))}\,{\scriptstyle b\mapsto u^{\{a\}}}}{\vdash P(a)\to P(b)}\vdash\to \quad \dfrac{}{\vdash Q(u^{\{a\}})\to Q(b)}\vdash\to}{\vdash (P(a)\to P(b))\wedge(Q(u^{\{a\}})\to Q(b))}\vdash\wedge}{\vdash \exists z.(P(a)\to P(z))\wedge(Q(u^{\{a\}})\to Q(z))}\vdash\exists}{\vdash \forall y.\exists z.(P(a)\to P(z))\wedge(Q(y)\to Q(z))}\vdash\forall}{\vdash \exists x.\forall y.\exists z.(P(x)\to P(z))\wedge(Q(y)\to Q(z))}\vdash\exists$$

In this example, each of the two branches when examined separately appears to be consistent. The set of bindings when taken as a whole however, produces $\{a \mapsto b, b \mapsto u^{\{a\}}\}$, which is not unifiable, and so the derivation shown above is invalid. Note that the unification algorithm shown above addresses this transitivity issue. This ensures that all the bindings throughout the derivation are consistent.

4.5 A type inference system

The type inference system described in Damas and Milner's paper[DM82] provides a good example of a type inference system suitable for this environment. The system is second-order propositional, as it allows quantification over types, which can be regarded as propositions. The role of quantification in the type inference system is to allow reuse of program fragments with different types of data. Instead of quantifying over formulas and predicates, quantification is carried out over types within type expressions.

The t in the following example is a bound variable over the types in the system and '$\forall t.\cdots t\cdots$' indicates that the type of t is arbitrary. The following are examples of quantified type expressions: '$\forall t.t\to t$' for the identity function; or '$\forall t.t \times (t \times t\to t) \times \text{list}(t)\to t$' for the type of a function, *reduce*, that applies a function to the elements in a list, accumulating the partial results.

```
reduce b f [] = b
       b f a:as = f a (reduce b f as)
```

The type system consists of three categories: Expression, Type, HasType; the last category is the association of a type with an expression. Of the two judgement categories, Sequent and Generalise, the first is similar to the intuitionistic sequent except that it uses *lists* rather than *sets*; the second category is used to generalise a type by forming its closure with respect to any free type variables occurring within it. This is accomplished using the rules *Ga* and *Gb* below. The language for the syntactic categories is as follows:

Syntactic categories are: Expr, HasType, Sequent, Type, Identifier, Generalise;

Judgements are: Sequent, Generalise;

Formation Rules are

Expr

'λ' *v*:Identifier '.' *e*:Expr	→	ABS(*v*,*e*)
e:Expr *f*:Expr	→	APPL(*e*,*f*)
'fix' *e*:Expr	→	FIX(*e*)
'let' *i*:Identifier '=' *e*:Expr 'in' *f*:Expr	→	LET(*i*, *e*, *f*)
'if' *e*:Expr 'then' *f*:Expr 'else' *g*:Expr 'fi'	→	IF(*e*,*f*,*g*)
e:Expr '=' *f*:Expr	→	EQ(*e*,*f*)
e:Expr '+' *f*:Expr	→	PLUS(*e*,*f*)
e:Expr '-' *f*:Expr	→	MINUS(*e*,*f*)
e:Expr '*' *f*:Expr	→	TIMES(*e*,*f*)
a:number	→	NUM(*a*)
i:Identifier	→	ID(*i*)

Sequent

a:ListOf(HasType) '⊢' *b*:HasType	→	SEQ(*a*,*b*)

HasType

a:Expr ':' *b*:Type	→	HT(a,b)

Type

'∀' *a*:Variable(Type) '.' *u*:Type	→	GEN(*a*:*u*)
u:Type '→' *v*:Type	→	IMP(*u*,*v*)
'int'	→	INT
'bool'	→	BOOL

Identifier

i:`^[abcijk][0-9]*$`	→	ID(*i*)

Generalise

e:ListOf(HasType) *u*:Type '::' *v*:Type	→	GEN(*e*,*u*,*v*)

The metavariables used are as follows:

ListOf(HasType)	ΓΔ
Expr	*efg*
Type	*uvw*
Variable(Type)	*st*
Identifier	*xyz*

The binder GEN in the category of types is used to allow generalisation over types, *e.g.* '$\forall t.u(t)$'. Types may thereby be reused with different instances in different situations *i.e.* polymorphism. The *let* construct illustrates this, for example:

$$let\ i\ =\ \lambda x.x\ in\ i\ i$$

Consider the two occurrences of 'i' in the body of the *let* construct. Each occurrence of 'i' has a different type and the overall type of the expression is '$t \rightarrow t$'. This is deduced by assigning 'i' the polymorphic type of '$\forall t.t \rightarrow t$' which allows the different instantiations of $(a \rightarrow a) \rightarrow (a \rightarrow a)$ and $b \rightarrow b$ which are shown in figure 4.3.

$$\frac{}{\Gamma \vdash e : int}\ number\ e \quad \text{INT-BASIC} \qquad \frac{}{\Gamma \vdash x : u}\ lookup\ x{:}u\ \Gamma \quad \text{VAR-BASIC}$$

$$\frac{\Gamma \vdash e : bool \quad \Gamma \vdash f : u \quad \Gamma \vdash g : u}{\Gamma \vdash if\ e\ then\ f\ else\ g\ fi : u}\ IF \qquad \frac{\Gamma \vdash e : u \rightarrow v \quad \Gamma \vdash f : u}{\Gamma \vdash ef : v}\ APPL$$

$$\frac{x : u, \Gamma \vdash f : v}{\Gamma \vdash \lambda x.f : u \rightarrow v}\ ABS \qquad \frac{\Gamma \vdash e : u \rightarrow u}{\Gamma \vdash fix\ e : u}\ FIX$$

$$\frac{\Gamma \vdash e : u \quad \Gamma\ u :: u' \quad i : u', \Gamma \vdash f : v}{\Gamma \vdash let\ i = e\ in\ f : v}\ LET \qquad \frac{\Gamma, e : u(v), \Gamma' \vdash f : w}{\Gamma, e : \forall x.u(x), \Gamma' \vdash f : w}\ INST$$

$$\frac{\Gamma \vdash e : u \quad \Gamma \vdash f : u}{\Gamma \vdash e = f : bool}\ EQ \qquad \frac{\Gamma \vdash e : int \quad \Gamma \vdash f : int}{\Gamma \vdash e + f : int}\ PLUS$$

$$\frac{\Gamma \vdash e : int \quad \Gamma \vdash f : int}{\Gamma \vdash e - f : int}\ MINUS \qquad \frac{\Gamma \vdash e : int \quad \Gamma \vdash f : int}{\Gamma \vdash e * f : int}\ TIMES$$

$$\frac{\Gamma\ \forall x.u :: v}{\Gamma\ u(t) :: v}\ Ga \qquad \frac{}{\Gamma\ v :: v}\ Gb$$

Figure 4.2: Rules for a type inference system

The rules used by the system are shown in figure 4.2. The derivation of the type $b \rightarrow b$ for '$let\ i = \lambda x.x\ in\ i\ i$' is shown below in figure 4.3. The structure of the proof reflects the syntactic form of the expression. The rightmost branch involves two instantiations of the closed type deduced for i namely '$\forall t.t \rightarrow t$', one for each occurrence of i in the body. The rules *Ga* and *Gb* perform the required closure (in the example the 'context', Γ, is empty).

The example in figure 4.4 shows the role of the occurs check in type inference. The sets of bindings $\{u \mapsto a, v \mapsto a\}$ and $\{v \mapsto a \rightarrow a\}$ are inconsistent since the first aliases v with a and this gives rise to $a \mapsto a \rightarrow a$ which is a circular term.

$$\dfrac{\dfrac{\dfrac{}{i : c{\to}c \vdash i : (b{\to}b){\to}(b{\to}b)}\ {\scriptstyle c\mapsto(b\to b)}}{i : \forall t.t{\to}t \vdash i : (b{\to}b){\to}(b{\to}b)}\ INST \quad \dfrac{\dfrac{}{i : d{\to}d \vdash i : b{\to}b}\ {\scriptstyle d\mapsto b}}{i : \forall t.t{\to}t \vdash i : b{\to}b}\ INST}{\boxed{\text{A}}}\ APPL$$

$$\dfrac{\dfrac{}{\vdash \lambda x.x : a{\to}a}\ ABS \quad \dfrac{\dfrac{}{\forall t.t{\to}t :: \forall t.t{\to}t}\ Gb}{a{\to}a :: \forall t.t{\to}t}\ Ga \quad \boxed{\text{A}}}{\vdash let\ i = \lambda x.x\ in\ i\ i : b{\to}b}\ LET$$

Figure 4.3: Type inference involving the *let* construct

$$\dfrac{\dfrac{}{i : a{\to}a \vdash i : v{\to}u}\ {\scriptstyle u\mapsto a, v\mapsto a} \quad \dfrac{}{i : a{\to}a \vdash i : v}\ {\scriptstyle v\mapsto a{\to}a}}{i : a{\to}a \vdash i\ i : u}\ APPL$$

Figure 4.4: Type inference involving circular term

Chapter 5

Derived Rules

This chapter discusses derived rules. The notion of a derived rule is defined. Derived rules are split into two classes: simple and recursive. The technique for checking whether a rule falls into one of these classes is described. Some applications of derived rules are presented.

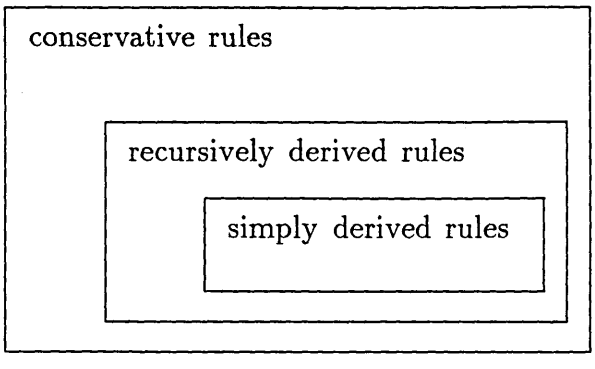
5.1 Introduction

A rule of inference is said to be *derived* with respect to some presentation of a system when its addition to the other rules of that system does not allow the derivation of any new judgements. This definition does not give any insight into how a rule can be checked to see if it is 'derived'. A more appropriate, and stronger, definition can be found by focusing on the way in which a rule can be used in a derivation.

> Any proof containing a *derived* rule can be algorithmically transformed into a proof having the same conclusion[1] but containing no occurrences of the derived rule.

This observation can be used to show that a rule is derived, and hence it can also be said that the rule is *conservative* with respect to the other rules of the system.

[1]The conclusion of a proof is taken to be the judgement with which it is rooted; its assumptions are any unsolved judgements - or proof obligations - on which the proof as a whole depends.



To demonstrate this, it must be shown that the rule can be eliminated from the proof wherever it occurs, to be replaced by other rules in the system; this may be carried out in one step, or in stages. This latter point is the basis for the distinction drawn between simply and recursively derived rules in the above diagram. In general, the algorithmic approach cannot be expected to identify all conservative rules, but using the techniques described below, a significant proportion can be identified.

In order to make the formulation of derived rules more precise, definitions must be provided for a proof, both as a final completed object and as it undergoes construction. A way of comparing proofs is needed, such that one proof is *similar* to another when it achieves a similar conclusion from similar assumptions. In this way, the internal structure of proofs can be disregarded when necessary.

Once the notion of a proof with respect to a system has been established, a notation for proof substitution is introduced that allows occurrences of specified rules within a proof to be replaced. With the aid of substitutions on proofs, transformations which rewrite proofs can be stated.

It is important to be able to show that a sequence of transformations will terminate in a finite number of steps. This is done by introducing a measure of the size of a proof, relative to the candidate derived rule; this should always be reduced by an application of a transformation. The distinction between simply and recursively derived rules is formalised using these definitions.

The remainder of the chapter is concerned with the type of analysis required to verify that a candidate rule is either simply or recursively derived.

The analysis of simply derived rules selectively and progressively combines forwards and backwards use of inference rules together with the judicious introduction of cut rules. The inference rules are combined schematically.

The analysis of recursively derived rules is more involved. When an overall proof strategy is available for the type rule in question, the transformation needed to remove occurrences of the rule from a proof can be subdivided into a number of simpler transformations. These subcases can be enumerated and verified by the environment. Two example rules are provided, and it is shown how the subcases are enumerated and validated for each of them. The examples are of a general weakening rule, and the cut rule. Gentzen's use of the cut rule to show consistency is discussed briefly.

5.2 Definitions

The definitions which follow will be used later in the analysis of derived rules.

5.2.1 Partial proofs

The *partial proofs* PP with respect to the language and rules of a system are defined as follows:

1. the judgements, whether valid or not, of the system are partial proofs, and
2. $R(P_1, \ldots, P_n, J)$ is a partial proof, when J is the judgement which results in the application of the inference rule R to the partial proofs $P_1, \ldots, P_n$.

Partial proofs are intended to represent proofs under construction. The conclusion of a partial proof is the conclusion of its outermost rule.

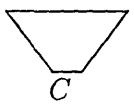
Partial proofs are written as $\dfrac{J_1 \cdots J_n}{C}$, and distinguished from single rules by their thicker horizontal lines. The J_i are the unsolved judgements and C is the conclusion.

5.2.2 Completed proofs

A *completed proof*, $p \in CP$, is defined as follows:

1. $R(P_1, \ldots, P_n, J)$ is a completed proof, when J is the judgement which results in the application of the inference rule R to the completed proofs $P_1, \ldots, P_n$.

A completed proof is a partial proof with no unsolved judgements. The notation



is used to represent the completed proof whose conclusion is the judgement C. A completed proof may contain a label to help distinguish it from other completed proofs in the transformation schemas later in the chapter.

5.2.3 Similar proofs

When studying proofs, it is often convenient to concentrate on their conclusions rather than their internal structure - which may vary according to the strategy used during their construction. Proofs which achieve similar conclusions from similar assumptions are called *similar* proofs. The relation symbol used for similarity is: '$\sim$'.

5.2.3.1 Similar completed proofs

Two completed proofs are said to be *similar*, written 

, when their conclusions are similar, $A \sim B$. Similarity between judgements is defined by the matching algorithm described in §5.2.12, but it can be stated briefly as: A and B can be made equal, given appropriate substitutions, and using any (structural) properties that are available by virtue of the categories they are formed from.

Example

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a,b \vdash a,c} \quad \dfrac{}{a,b \vdash b,c}}{a,b \vdash a \wedge b,c}\,{\vdash}\wedge}{a,b \vdash (a \wedge b) \vee c}\,{\vdash}\vee}{a \wedge b \vdash (a \wedge b) \vee c}\,{\wedge}\vdash \quad \sim \quad \dfrac{\dfrac{}{a \wedge b \vdash a \wedge b,c}}{a \wedge b \vdash (a \wedge b) \vee c}\,{\vdash}\vee$$

5.2.3.2 Similar partial proofs

It is useful to extend the notion of similarity to partial proofs that have similar unsolved judgements, called assumptions here. Proofs that make similar assumptions should be considered similar. The definition of similarity between partial proofs must account for the possibility that an assumption will occur more frequently in one partial proof than the other. There is no harm in identifying such cases, since once a proof is found for one occurrence of the assumption, it can be replicated for the other occurrences. This leads to the following definition.

Two partial proofs, $\dfrac{A_1 \cdots A_n}{A}$ and $\dfrac{B_1 \cdots B_m}{B}$, are *similar* when they have similar conclusions, $A \sim B$, as before, and when

$$(\forall i \in n.\exists j \in m.A_i \sim B_j) \wedge (\forall j \in m.\exists i \in n.B_j \sim A_i)$$

Examples

Partial proofs are similar modulo reordering of rules.

$$\dfrac{\dfrac{\Gamma,\varphi,\psi,\theta\vdash\Delta}{\Gamma,\varphi,\psi\vdash\neg\theta,\Delta}\vdash\neg}{\Gamma,\varphi\wedge\psi\vdash\neg\theta,\Delta}\wedge\vdash \quad\sim\quad \dfrac{\dfrac{\Gamma,\varphi,\psi,\theta\vdash\Delta}{\Gamma,\varphi\wedge\psi,\theta\vdash\Delta}\wedge\vdash}{\Gamma,\varphi\wedge\psi\vdash\neg\theta,\Delta}\vdash\neg$$

In the following example, the proofs are similar since, although the second partial proof does not use the unsolved judgement, '$\Gamma,\psi\vdash\theta$', the judgement is similar to '$\Gamma,\varphi\vdash\theta$', with the substitution $\psi\mapsto\varphi$.

$$\dfrac{\dfrac{\Gamma\vdash\varphi}{\Gamma\vdash\varphi\vee\psi}\vdash\vee a \quad \dfrac{\Gamma,\varphi\vdash\theta \quad \Gamma,\psi\vdash\theta}{\Gamma,\varphi\vee\psi\vdash\theta}\vee\vdash}{\Gamma\vdash\theta}\,cut \quad\sim\quad \dfrac{\Gamma\vdash\varphi \quad \Gamma,\varphi\vdash\theta}{\Gamma\vdash\theta}\,cut$$

5.2.4 Minimal Occurrences

An inference rule, R, has a minimal occurrence in a proof when all of its subproofs do not contain occurrences of R.

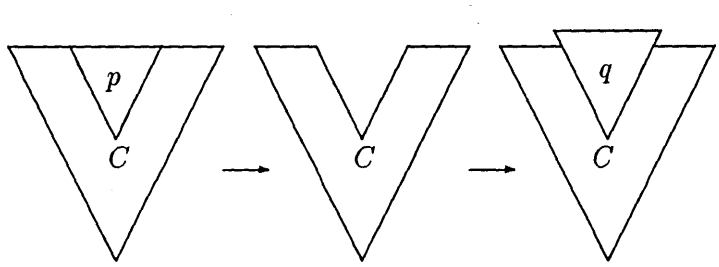
Example

In the following completed proof the uppermost occurrence of $\vdash\wedge$ is minimal:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a,b,c\vdash a} \quad \dfrac{}{a,b,c\vdash b}}{a,b,c\vdash a\wedge b}\vdash\wedge \quad \dfrac{}{a,b,c\vdash c}}{a,b,c\vdash(a\wedge b)\wedge c}\vdash\wedge}{a\wedge b,c\vdash(a\wedge b)\wedge c}\wedge\vdash$$

5.2.5 Proof Replacement

The following diagram illustrates the replacement of a subproof within a larger proof.



where $\overset{p}{\underset{C}{\triangledown}}$ and $\overset{q}{\underset{C}{\triangledown}}$ are two similar completed proofs.

The subproof rooted at C labelled p is selected from the proof P. The result is the same as replacing the unsolved judgement C in a partial proof with the proof labelled q provided the selected proof p and replacement proof q are similar. The similarity of the whole is preserved by the replacement.

Example

The following example illustrates the replacement of a subproof with a shorter similar proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{}\quad\overline{}}{a,b\vdash a,c \quad a,b\vdash b,c}\vdash\wedge}{a,b\vdash a\wedge b,c}\wedge\vdash}{a\wedge b\vdash a\wedge b,c}\vdash\vee}{a\wedge b\vdash (a\wedge b)\vee c}\left[\dfrac{\dfrac{\overline{}\quad\overline{}}{a,b\vdash a,c \quad a,b\vdash b,c}\vdash\wedge}{a,b\vdash a\wedge b,c}\wedge\vdash}{a\wedge b\vdash a\wedge b,c}\right]$$

$$= \dfrac{\dfrac{\overline{a\wedge b\vdash a\wedge b,c}}{a\wedge b\vdash (a\wedge b)\vee c}\vdash\vee}{}\left[\overline{a\wedge b\vdash a\wedge b,c}\right]$$

$$= \dfrac{\overline{a\wedge b\vdash a\wedge b,c}}{a\wedge b\vdash (a\wedge b)\vee c}\vdash\vee$$

Since

$$\dfrac{\dfrac{\dfrac{\overline{}\quad\overline{}}{a,b\vdash a,c \quad a,b\vdash b,c}\vdash\wedge}{a,b\vdash a\wedge b,c}\wedge\vdash}{a\wedge b\vdash a\wedge b,c} \sim \overline{a\wedge b\vdash a\wedge b,c}$$

5.2.6 Eliminating rules in a proof

Suppose, for the moment, that there is a method of eliminating a single *minimal* occurrence of a rule, R, from a proof. Then if a proof is given in which R occurs n times it is clear that:

1. while $n > 0$ do

   (a) there is a least one minimal occurrence of R.

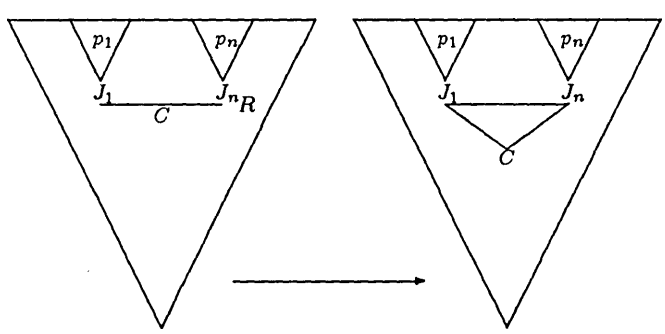
   (b) applying the method to the occurrence removes it from the proof.

2. all occurrences have been eliminated.

This indicates that all occurrences of R can be eliminated from a proof given that there is a means of eliminating the minimal occurrences of R.

5.2.7 Simply Derived Rules

The notion of a *simply derived rule* is illustrated by the following figure. The rule R shown on the left occurring within a larger proof is replaced with an equivalent subproof (the inner triangle) on the right:



Irrespective of the context in which the proof rule is used, there is a single partial proof $\dfrac{A_1 \cdots A_n}{A}$, that behaves in the same way as the rule R, so that $C \sim A$ and $J_i \sim A_i$, $1 \leq i \leq n$. This permits occurrences of R to be removed from the proof in a single step, in favour of other rules in the system.

To show that a rule is simply derived with respect to other rules, a construction must be found which satisfies this condition. This is done in §5.3.

Example

To illustrate this, consider the derived rule

$$\frac{\Gamma, \varphi \vdash \Delta}{\Gamma, \neg\neg\varphi \vdash \Delta}\ \neg\neg\vdash$$

and its equivalent partial proof

$$\dfrac{\dfrac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \neg\varphi, \Delta}\ \vdash\neg}{\Gamma, \neg\neg\varphi \vdash \Delta}\ \neg\vdash$$

An occurrence of $\neg\neg\vdash$ can be removed from a proof *e.g.*

$$\dfrac{\dfrac{\dfrac{}{a \vdash a} \quad \dfrac{}{b \vdash b}}{a, a{\to}b \vdash b}\ {\to}\vdash}{a, \neg\neg(a{\to}b) \vdash b}\ \neg\neg\vdash \quad \longrightarrow \quad \dfrac{\dfrac{\dfrac{}{a \vdash a} \quad \dfrac{}{b \vdash b}}{a, a{\to}b \vdash b}\ {\to}\vdash}{a \vdash \neg(a{\to}b), b}\ \vdash\neg}{a, \neg\neg(a{\to}b) \vdash b}\ \neg\vdash$$

5.2.8 Sizing proofs

The method in §5.2.6 for removing minimal occurrences depended on the ability to remove such an occurrence entirely. For simply derived rules this can be done in a single step - as the construction is independent of the context - but for some rules, the occurrences proliferate before they can be eliminated. For these rules, known here as recursively derived rules, an additional measure of size is needed that will ensure that the occurrences can eventually be eliminated.

Consider a minimal occurrence of R in a proof. From the definition of minimality, it is clear that the subproofs of R do not contain any further occurrences of R. A transformation applied to such an occurrence will either:

- eliminate the rule immediately;
- move the rule *higher* in the proof; or,
- replace the rule by several further rules, but any new occurrence of R will be *smaller* than the original in some way.

The notion of size defined here encodes these cases.

The *size* of a *minimal* occurrence of R in p is given by a pair $\langle C_R(p), s(p)\rangle$ where

- $s(p)$ is the number of rules in p:

$$s(R(p_1, \ldots, p_n, J)) = \sum_{i=1}^{n} s(p_i) + 1$$

- $C_R(p)$ is a measure of the complexity of the occurrence as a function of the rule. In the case of a cut rule for example, this is a measure of the complexity of the cut formula. For other rules, such as weakening, it is constant and can be taken to be zero.

The sizes of occurrences are compared as follows:

$$\langle c, s\rangle < \langle c', s'\rangle \text{ if } \begin{cases} c = c' \text{ and } s < s' \\ c < c' \end{cases}$$

The *size* of a completed proof with respect to R can now be written as $S_R(p)$, where $S_R(p)$ is the maximum of the weights assigned to *minimal* occurrences of R in p.

Examples

1. R is eliminated.

$$\frac{\dfrac{A_1 \quad A_2}{B_1}\,R_1}{C_1} \longrightarrow \frac{C_1 \quad C_2}{B_1}$$

$$\langle C_R(R_1), 0\rangle > \langle 0, 0\rangle$$

2. R is moved up, and perhaps copied.
$C_R(R_1) = C_R(R_2) = C_R(R_3) = r$

$$\frac{\dfrac{A_1 \quad A_2}{B_1}\quad B_2}{C_1}\,R_1 \longrightarrow \frac{\dfrac{A_1 \quad A_2}{B_1}\,R_2 \quad \dfrac{A_3 \quad A_4}{B_2}\,R_3}{C_1}$$

$$\langle r, 1\rangle > (\langle r, 0\rangle = max\{\langle r, 0\rangle, \langle r, 0\rangle\})$$

3. R is reduced in complexity, but further occurrences may be introduced.
$C_R(R_1) = r_1 > r_2 = C_R(R_2) = C_R(R_3)$

$$\frac{\dfrac{A_1 \quad A_2}{B_1}\quad B_2}{C}\,R_1 \longrightarrow \frac{\dfrac{A_1}{E}\quad B_2}{D_1}\,R_2 \quad \frac{A_2 \quad B_2}{D_2}\,R_3}{C}$$

$$\langle r_1, 1\rangle > (\langle r_2, 1\rangle = max\{\langle r_2, 1\rangle, \langle r_2, 0\rangle\})$$

5.2.9 Proof Transformations

It is now possible to define a transformation which can assist the elimination of a rule R.

Define an R-transformation on completed proofs, $T_R : CP \rightarrow CP$, such that if $p, q \in CP$, the occurrence of R is outermost in p (so p has the form $R(p_1, \ldots, p_n, J)$), and $T_R(p) = q$ then $p \sim q$ and $S_R(p) > S_R(q)$.

5.2.10 Recursively Derived Rules

A rule R is called *recursively derived* if, for any minimal occurrence of R, say p, there exists a R-transformation applicable to p such that $S_R(p) > S_R(T_R(p))$.

The procedure given in §5.2.6 can now be modified to make use of T_R.

If there is a proof in which R occurs n times, it is clear that:

1. while $n > 0$ do
   (a) there is a least one minimal occurrence of R called p.
   (b) applying T_R to p may introduce additional occurrences of R but they will all be smaller than the original occurrence. In particular they can be eliminated by recursively applying the procedure to $T_R(p)$. This eliminates R from p.
2. all occurrences have been eliminated.

5.2.11 Typed Internal Form

The *typed internal form* is the representation of inference rules that is used within the prototype environment. It provides local information about the names of categories and elements which form the judgements of an inference rule. The language of the system is often used to assign structural properties to collections. Judgements may also contain *patterns* (§3.2.8.1), binders and substitutions. These properties - which are implicit in the user's view of a judgement - must be made explicit so that rules may be combined with each other. The object level use of rules is determined by a translation which is described in §7.5.4. The following section describes the matching algorithm that is used to combine uninstantiated rules at the "meta-level."

The language component of a system defines a convenient mapping between the concrete syntax and the abstract syntax of the categories which are manipulated by the rules of the system. Parse trees in the abstract syntax are annotated to reflect the categories to which they belong, producing what is known here as *typed internal form* (*TIF*). Typed internal form has an untyped counterpart called *concise internal form*. Untyped form may be converted to typed internal form by inferring types by reference to the structure of the language of the system. The reverse conversion may be made by discarding any type information which is present, see §7.4.10 below.

Typed internal form is:

1. $\mathcal{E}_c^f(t_1, \ldots, t_n)$ for elements;
2. $\mathcal{C}_c(t_1, \ldots, t_n)$ for collections;
3. $\mathcal{V}_c(v)$ for metavariables;
4. $\mathcal{B}_c^v(t_1, \ldots, t_n)$ for binders; and,
5. $\mathcal{A}_c(t_1, t_2)$ for application of a binder to an argument.

where f is the name of a constructor, t_i is a *TIF* expression, c is the name of a category, and v is the name of a variable.

Typed internal form is very verbose as the example below illustrates. Its main benefit is that it localises type information and thus simplifies the description of algorithms that manipulate expressions.

Example

Consider the language of §4.2.1 and the sequent '$\Gamma \vdash \forall x.\varphi(x)$' which has the *TIF* equivalent of:

$$\mathcal{E}^{SEQ}_{Sequent}\Big(\begin{array}{l} \mathcal{V}_{SetOf(Formula)}(\Gamma), \\ \mathcal{C}_{SetOf(Formula)}(\mathcal{A}_{Formula}(\mathcal{E}^{FORALL}_{Formula}(\mathcal{V}_{Variable(Term)}(x), \mathcal{V}_{Formula}(\varphi)), \mathcal{V}_{Variable(Term)}(x))) \end{array} \Big)$$

The matching algorithm uses the typed internal form to combine judgements in their meta-level representations.

5.2.12 Similar Judgements

Given two *TIF* expressions, J_1 and J_2, denoting judgements, they are *similar*, $J_1 \sim J_2$, when they can be successfully matched.

The matching process can be visualised as

$$\frac{J_1}{J_2}$$

since the proof is constructed vertically. The algorithm works in symmetric and asymmetric modes. In the symmetric mode, J_1 and J_2 are treated equally, whereas in the asymmetric mode (written $J_2 :\sim J_1$) variables in J_2 are protected against substitution. This is useful if the conclusion of a proof is to remain unaffected by extensions made to its leaves.

The objective of the algorithm is to find some substitution to the variables in J_1 and, optionally, J_2 so that the resulting two expressions are syntactically equal modulo the properties that are present through the collections they contain.

For the case of SetOf, the algorithm implements commutative, associative and idempotent matching if the mode is asymmetric, or unification if the mode is symmetric. The case of BagOf is the same as SetOf except that idempotence is lost. For the case of ListOf, associative matching or unification is used.

$$\begin{aligned} J_1 \sim J_2 &= \mathit{Similar}(J_1,\ J_2,\ \text{symmetric}) \\ J_1 :\sim J_2 &= \mathit{Similar}(J_1,\ J_2,\ \text{asymmetric}) \end{aligned}$$

Algorithm

In the following the *mode* indicates whether the terms are treated symmetrically (with unification) or asymmetrically (with matching). A variable is only bound to a term when there are no free occurrences of the variable within the term. The algorithm can give rise to several unifiers when choices are available, *e.g.* for ListOf in the selection of prefixes.

Similar(J_1, J_2, *mode*) holds if
Case $\langle J_1, J_2 \rangle$ in

$\langle \mathcal{C}_{l(c)}(S), \mathcal{C}_{l(c)}(T) \rangle$ where l is the type of the collection: SetOf, BagOf, ListOf.

Case l in

ListOf It is not permitted to change the order of the sequence, or to duplicate elements or collection metavariables. However collection metavariables can be bound to appropriate sequences, including the empty-sequence. Variables in S can only be bound when the *mode* is **symmetric**.

While S and T are non-empty do

$T = t, T'$ and $S = s, S'$

Case $\langle t, s \rangle$ in

$\langle \mathcal{V}_{\mathsf{ListOf}(c)}(u), s \rangle$ then bind u to a (possibly empty) prefix of S and call the remaining elements S'.

$\langle \mathcal{V}_c(u), \mathcal{E}_c^f(P) \rangle$ bind u to the element provided that u does not occur free within the element.

$\langle \mathcal{V}_c(u), \mathcal{V}_c(v) \rangle$ bind u to v.

$\langle \mathcal{E}_c^f(P), \mathcal{E}_c^f(Q) \rangle$ apply the algorithm recursively to the arguments pairwise $\langle p_i, q_i \rangle$, $1 \leq i \leq n$.

$\langle t, \mathcal{V}_{\mathsf{ListOf}(c)}(v) \rangle$ (**symmetric** only) bind v to a (possibly empty) prefix of T and call the remaining elements T'.

$\langle \mathcal{E}_c^f(P), \mathcal{V}_c(v) \rangle$ (**symmetric** only) bind u to v.

Let $S \leftarrow S'$ and $T \leftarrow T'$

BagOf, SetOf Proceed as follows:

1. Split each of the components of S and T into elements and unbound collection metavariables, $\langle E, M \rangle$ and $\langle F, N \rangle$ respectively.
2. Find matches for the elements of E and F by the procedure recursively. If the l is SetOf, allow elements to be duplicated. Otherwise find a match for the elements allowing reordering when required.
3. Let $E - F$ (resp. $F - E$) be the elements of E (resp. F) that fail to find matches in F (resp. E).
4. Try to find assignments of collection metavariables to unmatched elements as follows.

   If N and $E - F$ are both non-empty then choose a $\mathcal{V}_c(u)$ from N and bind it to the collection $\mathcal{C}_{l(c)}(E - F, \mathcal{V}_{l(c)}(v))$ where v is a new collection variable, and add v to N. Otherwise fail.

   If the *mode* is **symmetric** and M and $E - F$ are both non-empty then choose a $\mathcal{V}_{l(c)}(u)$ from M and bind it to the collection $\mathcal{C}_{l(c)}(F - E, \mathcal{V}_{l(c)}(v))$ where v is a new collection variable, and add v to M. Otherwise fail.
5. After the elements have been reconciled, match the collection metavariables, M and N:

If $|M| > |N|$ and the *mode* is **asymmetric** then, by the pigeon hole principle, two m's must be identified, so fail.

Otherwise construct a binding from N to M as follows:

Case l in

BagOf bind each n_i to a m_j, and bind remaining m's or n's to the empty collection.

SetOf if $|M| \neq |N|$ then make copies of which ever has the fewer and bind each n_i to a m_j.

$\langle \mathcal{E}_c^f(S), \mathcal{E}_c^f(T) \rangle$ Match the arguments pairwise $\langle s_i, t_i \rangle$, $1 \leq i \leq n$.

$\langle \mathcal{E}_c^f(T), \mathcal{V}_c(v) \rangle$ Fail if the mode is **asymmetric** otherwise bind the variable v to the element $\mathcal{E}_c^f(T)$, provided v does not occur free in $\mathcal{E}_c^f(T)$.

$\langle \mathcal{V}_c(v), \mathcal{E}_c^f(T) \rangle$ Bind the variable v to the element $\mathcal{E}_c^f(T)$, provided v does not occur free in $\mathcal{E}_c^f(T)$.

$\langle \mathcal{V}_c(u), \mathcal{V}_c(v) \rangle$ Bind u to v.

Examples

1. Suppose Γ, Δ are **ListOf(formula)** and φ, ψ is a **formula**, and $\varphi \sim \psi$. The strategy for lists applies in this case.

$J_1 = \Gamma_1, \varphi, \Gamma_2$	$J_2 = \psi, \Delta$
1. $\Gamma_1 \mapsto \varnothing; \varphi \mapsto \psi; \Gamma_2 \mapsto \Delta$	
$J_1 = \psi, \Delta$	$J_2 = \psi, \Delta$

but when $\varphi \not\sim \psi$ and the mode is **symmetric**, *e.g.*

$J_1 = \Gamma_1, \psi \wedge \varphi, \Gamma_2$	$J_2 = \psi' \vee \varphi', \Delta$
1. $\Gamma_1 \mapsto \psi' \vee \varphi'$	$\Delta \mapsto \psi \wedge \varphi, \Gamma_2$
$J_1 = \psi' \vee \varphi', \psi \wedge \varphi, \Gamma_2$	$J_2 = \psi' \vee \varphi', \psi \wedge \varphi, \Gamma_2$

2. In the following example $\Gamma_1, \Gamma_2, \Delta$ are **SetOf(formula)** and φ is a **formula**. In **symmetric** mode:

$J_1 = \Gamma_1, \varphi, \Gamma_2$	$J_2 = \Delta$
$\langle E_1, M_1 \rangle = \langle \{\varphi\}; \Gamma_1, \Gamma_2 \rangle$	$\langle E_2, M_2 \rangle = \langle \varnothing; \Delta \rangle$
$E_1 - E_2 = \{\varphi\}$	$E_2 - E_1 = \varnothing$
	$\Delta \mapsto \varphi, \Gamma_1, \Gamma_2$

In **asymmetric** mode the algorithm would report failure as J_2 is protected against substitution, and J_1 is more specific that J_2.

5.2.13 Subformula property

The definition of *TIF* above can be used to present a more general formulation of the subformula property given in §2.7.1. There, the definition was in terms of sequents; it is now possible to give a definition in terms of judgements. The subformulas of a judgement J are its *TIF* constituents $SF(J)$, defined below:

$$SF'(t) = \{t\} \cup SF(t)$$

$$\left.\begin{array}{l} SF(\mathcal{E}^f_j(t_1,\ldots,t_n)) \\ SF(\mathcal{C}_c(t_1,\ldots,t_n)) \\ SF(\mathcal{B}^v_c(t_1,\ldots,t_n)) \end{array}\right\} = \bigcup_{i=1}^{n} SF'(t_i)$$

$$SF(\mathcal{V}_c(v)) = \varnothing$$

$$SF(\mathcal{A}_c(s,t)) = SF'(s) \cup SF'(t)$$

A rule has the subformula property when all of the syntactic elements that make up the judgements above the horizontal line can be found in the judgement that is beneath the line.

$$\frac{J_1 \quad \cdots \quad J_n}{J_0}$$

has the subformula property when

$$\bigcup_{i=1}^{n} SF(J_i) \subset SF(J_0)$$

Example

The rule $\dfrac{\Gamma \vdash \varphi}{\Box\Gamma, \Gamma' \vdash \Box\varphi}$ can be expressed in *TIF* as:

$$\frac{\mathcal{E}^{SEQ}_{Sequent}(\mathcal{V}_{SetOf(Formula)}(\Gamma), \mathcal{V}_{Formula}(\varphi))}{\mathcal{E}^{SEQ}_{Sequent}(\mathcal{C}_{SetOf(Formula)}(\mathcal{E}^{BOX}_{Formula}(\mathcal{V}_{SetOf(Formula)}(\Gamma)), \mathcal{V}_{SetOf(Formula)}(\Gamma')), \mathcal{E}^{BOX}_{Formula}(\mathcal{V}_{Formula}(\varphi)))}$$

Applying $SF(-)$ to the uppermost judgement gives

$$\{\ \mathcal{V}_{SetOf(Formula)}(\Gamma), \mathcal{V}_{Formula}(\varphi)\ \}$$

The result of applying $SF(-)$ to the conclusion is:

$$\{\ \mathcal{E}^{BOX}_{Formula}(\mathcal{V}_{SetOf(Formula)}(\Gamma)), \mathcal{V}_{SetOf(Formula)}(\Gamma), \mathcal{V}_{SetOf(Formula)}(\Gamma'), \mathcal{E}^{BOX}_{Formula}(\mathcal{V}_{Formula}(\varphi)), \mathcal{V}_{Formula}(\varphi)\ \}$$

Clearly, the former is included in the latter, and the rule has the sub-formula property as required.

5.2.14 Classification of Rules

A number of rule characteristics may be distinguished algorithmically. Questions which may be answered in this way include:

1. Does the rule have the subformula property? The subformula property requires that all the syntactic components in the antecedents have occurrences in the conclusion.

2. Does the rule contain metavariables that have no occurrences in the conclusion? If metavariables are present in the antecedents but not in the conclusion the rule might cause the theorem prover to diverge.

3. Is the rule branching? What is its degree? This can be useful when ordering application of rules.

4. If the judgement type of the conclusion is appropriate, does the rule introduce a connective?

   (a) on the left-hand side, *e.g.*

   $$\frac{\varphi, \psi, \Gamma \vdash \Delta}{\varphi \wedge \psi \vdash \Delta} \wedge\vdash$$

   (b) on the right-hand side, *e.g.*

   $$\frac{\Gamma \vdash \varphi, \psi}{\Gamma \vdash \varphi \vee \psi} \vee\vdash$$

   (c) both together, *e.g.*

   $$\frac{\Gamma \vdash \varphi}{\Box\Gamma \vdash \Box\varphi}\ \textit{normal}$$

   (d) Does it have other modal patterns?

In the environment, this information is determined as the rules are *digested* (§7.5.3) and is stored with the internal representation of the rule.

5.3 Verifying Simply Derived Rules

Suppose that $\dfrac{J_1 \cdots J_n}{C} R$ is a candidate simply derived rule. The objective is to construct a partial proof $\dfrac{A_1 \cdots A_n}{A}$ using other rules in the system such that

$$\frac{A_1 \cdots A_n}{A} \quad \sim \quad \frac{J_1 \cdots J_n}{C} R$$

To ensure that the partial proof can be used in any context, it must be at least as general as the rule, hence $C :\sim A$ and $J_i :\sim A_i$, $1 \leq i \leq n$.

The simply derived rules discussed here are simple with respect to an all-introduction presentation of a system which includes an appropriate form of cut rule.

If the cut rule is not accepted, the strategy given here is restricted to rules that enjoy the subformula property.

The following examples of simply derived rules are used to motivate the discussion of our approach to verification.

5.3.1 Examples

1. Consider the following rule which would appear likely to be simply derived.

$$\frac{\Gamma, \psi \wedge \varphi \vdash \Delta}{\Gamma, \varphi \wedge \psi \vdash \Delta}$$

The justification for this is the following proof:

$$\dfrac{\Gamma, \psi \wedge \varphi \vdash \Delta \qquad \dfrac{\dfrac{\dfrac{}{\varphi, \psi \vdash \psi} \quad \dfrac{}{\varphi, \psi \vdash \varphi}}{\varphi, \psi \vdash \psi \wedge \varphi} \vdash\wedge}{\varphi \wedge \psi \vdash \psi \wedge \varphi} \wedge\vdash}{\Gamma, \varphi \wedge \psi \vdash \Delta} \; cut$$

The example contains an application of a cut rule whose cut formula is the main formula of the upper sequent of the derived rule. A cut rule is required here as the formula in the upper sequent is not a simple subformula of the lower sequent.

2. However, consider the simply derived rule

$$\frac{\Gamma \vdash \varphi \rightarrow \psi, \Delta \quad \Gamma \vdash \varphi \rightarrow \theta, \Delta}{\Gamma \vdash \varphi \rightarrow (\psi \wedge \theta), \Delta}$$

and its derivation

$$\dfrac{\dfrac{\Gamma \vdash \varphi \rightarrow \psi, \Delta \qquad \dfrac{\overline{\varphi \vdash \varphi} \quad \overline{\psi \vdash \psi}}{\varphi \rightarrow \psi, \varphi \vdash \psi}\rightarrow\vdash}{\Gamma, \varphi \vdash \psi, \Delta}\,cut \qquad \dfrac{\Gamma \vdash \varphi \rightarrow \theta, \Delta \qquad \dfrac{\overline{\varphi \vdash \varphi} \quad \overline{\theta \vdash \theta}}{\varphi \rightarrow \theta, \varphi \vdash \theta}\rightarrow\vdash}{\Gamma, \varphi \vdash \theta, \Delta}\,cut}{\dfrac{\Gamma, \varphi \vdash \psi \wedge \theta, \Delta}{\Gamma \vdash \varphi \rightarrow (\psi \wedge \theta), \Delta}\vdash\rightarrow}\vdash\wedge$$

containing the partial proof

$$\frac{\Gamma, \varphi \vdash \psi, \Delta \quad \Gamma, \varphi \vdash \theta, \Delta}{\Gamma \vdash \varphi \rightarrow (\psi \wedge \theta), \Delta} \quad \sim \quad \dfrac{\dfrac{\Gamma, \varphi \vdash \psi, \Delta \quad \Gamma, \varphi \vdash \theta, \Delta}{\Gamma, \varphi \vdash \psi \wedge \theta, \Delta}\vdash\wedge}{\Gamma \vdash \varphi \rightarrow (\psi \wedge \theta), \Delta}\vdash\rightarrow$$

whose unsolved judgements become the targets of the cut rules as shown.

3. The sequentised elimination rules of the Natural Deduction presentation developed in Chapter 2 are also examples of simply derived rules, *e.g.* $\rightarrow\vdash\mathcal{E}$, $\wedge\vdash\mathcal{E}a$, $\wedge\vdash\mathcal{E}b$, $\vee\vdash\mathcal{E}$, $\neg\neg\vdash\mathcal{E}$, $\neg\vdash\mathcal{E}$.

$$\frac{\Gamma \vdash \varphi \quad \Gamma \vdash \varphi \rightarrow \psi}{\Gamma \vdash \psi}\rightarrow\vdash\mathcal{E} \quad \sim \quad \dfrac{\Gamma \vdash \varphi \rightarrow \psi \qquad \dfrac{\Gamma \vdash \varphi \quad \overline{\psi \vdash \psi}}{\Gamma, \varphi \rightarrow \psi \vdash \psi}\rightarrow\vdash}{\Gamma \vdash \psi}\,cut$$

$$\frac{\Gamma, \varphi \vdash \theta \quad \Gamma, \psi \vdash \theta \quad \Gamma \vdash \varphi \vee \psi}{\Gamma \vdash \theta}\vee\vdash\mathcal{E} \quad \sim \quad \dfrac{\Gamma \vdash \varphi \vee \psi \qquad \dfrac{\Gamma, \varphi \vdash \theta \quad \Gamma, \psi \vdash \theta}{\Gamma, \varphi \vee \psi \vdash \theta}\vee\vdash}{\Gamma \vdash \theta}\,cut$$

In these rules, there are occurrences of metavariables contained in the upper judgements of the rule that have no corresponding occurrences in the conclusion of the rule. As before, the derivation of the rule requires the insertion of a cut rule. In each case, the subproof that results from the cut rule is a partial proof, and its unsolved judgements are similar to judgements in the candidate rule.

These observations lead to the following strategy for checking simply derived rules.

5.3.2 Method

The examples given above illustrate aspects of the problem. If the candidate rule enjoys the subformula property, the task of verifying its status is simplified: a

construction that replaces the rule can be found using the all-introduction rules alone. When a candidate rule does not have the subformula property, there are some subformulas of its upper judgements that do not belong to the conclusion of the rule. In this case, it is necessary to proceed *indirectly*, using appropriately formed cut rules.

The strategy is to work backwards from the conclusion of the candidate rule, C in the figure below. The backwards expansion using all-introduction rules in the system produces a number of target judgements, $\mathcal{T}$. Any upper judgement of the rule J_i that is similar to a T_j enables the pair $\langle J_i, T_j \rangle$ to be eliminated from the search, *e.g.* $\langle J_1, T_1 \rangle$ in the figure below. After all such pairs have been eliminated, the remaining $\mathcal{J}$ and $\mathcal{T}$ can be paired using a cut rule, *e.g.* the pair $\langle J_i, T_j \rangle$ are cut with a new judgement T'_j as follows:

$$\dfrac{J_i \qquad T'_j}{T_j}\ cut$$

The introduction of a cut between J_i and T_j is only possible when the J_i contains a suitable subformula that can act as a cut formula, and lead to the reduction of the resulting T'_j. This is expanded in the same fashion as the original judgement C. (The judgements that result are shown as Ss in the figure below.) The number of candidate judgements are increased in this way. The backwards expansion of C and $\mathcal{T}$ must be interleaved with the pairing and introduction of cut formulas. This is because order of application of inference rules is significant: two or more rules may be applicable at a given point, and different sets of target judgements can result.

$$\dfrac{\begin{array}{ccccc} J_1 & & & & J_n \\ \vdots & & \dfrac{S_1 \cdots S_p}{\dfrac{J_i \qquad T'_j}{T_j}\ cut} & & \vdots \\ T_1 & \cdots & & \cdots & T_m \end{array}}{C}$$

The examples revisited

The correspondences between this approach and the justifications offered for the examples above can be seen when their judgements are labelled:

Example 1.

$$\dfrac{\dfrac{\overbrace{\Gamma, \varphi \wedge \psi \vdash \Delta}^{J} \qquad \dfrac{\overline{\Gamma, \varphi, \psi \vdash \varphi, \Delta} \qquad \overline{\Gamma, \varphi, \psi \vdash \psi, \Delta}}{\Gamma, \varphi, \psi \vdash \varphi \wedge \psi, \Delta}}{\overbrace{\Gamma, \varphi, \psi \vdash \Delta}^{T}}\ cut}{\underbrace{\Gamma, \varphi \wedge \psi \vdash \Delta}_{C}}\ \wedge\vdash$$

Expand a target judgement A target judgement T in $\mathcal{T}$ is replaced by the judgements that result from the application of an inference rule, R:

$$\frac{S_1 \cdots S_p}{S}\,R \qquad T :\sim S$$

$$\mathcal{T} \leftarrow \mathcal{T} - \{T\} + \{S_1, \ldots, S_p\}$$

The expansion may lead to a reduction in the size of $\mathcal{T}$ when $p = 0$.

A bound may be placed on the size of $\mathcal{T}$ so that, after all expansion has been performed, $|\mathcal{T}| \leq |\mathcal{J}|$.

Introduce a cut rule Form an application of the cut rule for a pair of judgements $\langle T, J\rangle \in \mathcal{T} \times \mathcal{J}$ as follows:

$$\frac{A \qquad T'}{B}\,cut \qquad J :\sim A,\ T :\sim B$$

$$\mathcal{T} \leftarrow \mathcal{T} - \{T\} + \{T'\}$$

$$\mathcal{J} \leftarrow \mathcal{J} - \{J\}$$

The new target judgement T' is constructed from the other two judgements: a *cut formula*, f, is chosen from A. This is inserted, in the opposite side of a new sequent T'. The sequent is formed by combining of the elements of $A - f$ and B, *e.g.*

$$J = A = \Gamma, \varphi \wedge \psi \vdash \Delta \quad T = B = \Gamma\varphi, \psi \vdash \Delta \quad f = \varphi \wedge \psi \quad T' = \Gamma, \varphi, \psi \vdash \varphi \wedge \psi, \Delta$$

The introduction of the cut rule may allow further expansion of T' or its pairing with another $J \in \mathcal{J}$

Example 2.

$$\dfrac{\dfrac{\overbrace{\Gamma \vdash \varphi \to \psi, \Delta}^{J_1} \quad \dfrac{\overline{\Gamma, \varphi \vdash \varphi, \Delta} \quad \overline{\Gamma, \psi \vdash \psi, \Delta}}{\Gamma, \varphi, \varphi \to \psi \vdash \psi, \Delta}\,{\to}\vdash}{\overbrace{\Gamma, \varphi \vdash \psi, \Delta}^{T_1}}\,cut \quad \dfrac{\overbrace{\Gamma \vdash \varphi \to \theta, \Delta}^{J_2} \quad \dfrac{\overline{\Gamma, \varphi \vdash \varphi, \Delta} \quad \overline{\Gamma, \theta \vdash \theta, \Delta}}{\Gamma, \varphi, \varphi \to \theta \vdash \theta, \Delta}\,{\to}\vdash}{\overbrace{\Gamma, \varphi \vdash \theta, \Delta}^{T_2}}\,cut}{\dfrac{\Gamma, \varphi \vdash \psi \wedge \theta, \Delta}{\underbrace{\Gamma \vdash \varphi \to (\psi \wedge \theta), \Delta}_{C}}\,\vdash\to}\,\vdash\wedge$$

Example 3.

$$\dfrac{\overbrace{\Gamma \vdash \varphi \to \psi}^{J_1} \quad \dfrac{\overbrace{\Gamma \vdash \varphi}^{J_2 :\sim T_2} \quad \overline{\psi \vdash \psi}}{\Gamma, \varphi \to \psi \vdash \psi}\,{\to}\vdash}{\underbrace{\Gamma \vdash \psi}_{C :\sim T_1}}\,cut$$

$$\dfrac{\overbrace{\Gamma \vdash \varphi \vee \psi}^{J_1} \quad \dfrac{\overbrace{\Gamma, \varphi \vdash \theta}^{J_2 :\sim T_2} \quad \overbrace{\Gamma, \psi \vdash \theta}^{J_3 :\sim T_3}}{\Gamma, \varphi \vee \psi \vdash \theta}\,\vee\vdash}{\underbrace{\Gamma \vdash \theta}_{C :\sim T_1}}\,cut$$

5.3.3 Algorithm

Initial values of $\mathcal{J}$ (unsolved judgements) and $\mathcal{T}$ (target judgements) are assigned as follows:

$$\begin{aligned} \mathcal{J} &= \{J_1, \ldots, J_n\} \\ \mathcal{T} &= \{C\} \end{aligned}$$

The search-space is then given by the non-deterministic application of three rules. A solution is found when $\mathcal{J}$ and $\mathcal{T}$ are both empty. The result is a partial proof that justifies the candidate rule's status as a simply derived rule.

Remove a similar pair The components of any pair $\langle T, J \rangle \in \mathcal{T} \times \mathcal{J}$ for which $J :\sim: T$ are removed from $\mathcal{T}$ and $\mathcal{J}$ respectively:

$$\mathcal{T} \leftarrow \mathcal{T} - \{T\}$$

$$\mathcal{J} \leftarrow \mathcal{J} - \{J\}$$

Such a pair means that T is at least as general as J and the unsolved judgement J, is required by the partial proof being constructed.

5.4 Verifying Recursively Derived Rules

A recursively derived rule was defined earlier in terms of the application of an R-transformation to a completed proof. This section concentrates on identifying examples of such transformations, and shows how to construct the internal structure of a transformation from an analysis of the other rules in the system.

5.4.1 Defining T_{weaken}

Suppose it is wished to show that the rule

$$\frac{\Gamma \vdash \Delta}{\varphi, \Gamma \vdash \Delta}\ \textit{weaken}$$

is recursively derived in a system containing just basic sequents of the form

$$\frac{}{\Gamma, \varphi \vdash \varphi, \Delta}\ \textit{basic}$$

and the rules $\vdash\wedge$ and $\wedge\vdash$ (taken from §2.8.6).

An appropriate transformation T_{weaken} must be applicable in arbitrary proof contexts. The contexts that are of concern for this rule have the form:

$$\frac{\begin{array}{c}\triangledown\\ P\\ \Gamma \vdash \Delta\end{array}}{\varphi, \Gamma \vdash \Delta}$$

The function T_{weaken} splits into cases depending on the outermost rule in the proof labelled P. There are three possibilities *basic*, $\vdash\wedge$ and $\wedge\vdash$.

For the first case, suppose an application has the form, $\psi \neq \varphi$:

$$\frac{\dfrac{}{\psi, \Gamma \vdash \psi, \Delta}\ \textit{basic}}{\varphi, \psi, \Gamma \vdash \psi, \Delta}\ \textit{weaken}$$

then *weaken* can be eliminated immediately and replaced with a new proof:

$$\frac{}{\varphi, \psi, \Gamma \vdash \psi, \Delta}\ \textit{basic}$$

If $\psi = \varphi$ then the weakening is redundant

$$\frac{\dfrac{}{\varphi, \Gamma \vdash \varphi, \Delta}\ \textit{basic}}{\varphi, \varphi, \Gamma \vdash \varphi, \Delta}\ \textit{weaken}$$

and the new proof becomes $\dfrac{}{\varphi, \Gamma \vdash \varphi, \Delta}\ \textit{basic}$

There are two other cases to consider, one each for the logical rules $\wedge\vdash$ and $\vdash\wedge$, respectively. For $\wedge\vdash$ the proof is:

$$\dfrac{\dfrac{\Gamma\vdash\varphi,\Delta \quad \Gamma\vdash\psi,\Delta}{\Gamma\vdash\varphi\wedge\psi,\Delta}\vdash\wedge}{\theta,\Gamma\vdash\varphi\wedge\psi,\Delta}\,weaken \quad\longrightarrow\quad \dfrac{\dfrac{\Gamma\vdash\varphi,\Delta}{\theta,\Gamma\vdash\varphi,\Delta}\,weaken \quad \dfrac{\Gamma\vdash\psi,\Delta}{\theta,\Gamma\vdash\varphi,\Delta}\,weaken}{\theta,\Gamma\vdash\varphi\wedge\psi,\Delta}\vdash\wedge$$

and for $\wedge\vdash$:

$$\dfrac{\dfrac{\Gamma,\varphi,\psi\vdash\Delta}{\Gamma,\varphi\wedge\psi\vdash\Delta}\wedge\vdash}{\theta,\Gamma,\varphi\wedge\psi\vdash\Delta}\,weaken \quad\longrightarrow\quad \dfrac{\dfrac{\Gamma,\varphi,\psi\vdash\Delta}{\theta,\Gamma,\varphi,\psi\vdash\Delta}\,weaken}{\theta,\Gamma,\varphi\wedge\psi\vdash\Delta}\wedge\vdash$$

Each of the four proof transformations reduces the measure S_{weaken} when $C_{\text{weaken}}(p)=0$. Using the algorithm of §5.2.10 with T_{weaken}, all occurrences of *weaken* will be eliminated from the proof in favour of broader basic sequents.

5.4.2 Defining T_{cut}

The next example considers the transformations needed for the cut rule. The significance of the cut rule lies in its description of the transitivity of the consequence relation of a system. If the cut rule can be been shown to be a derived rule then there are many important corollaries.

5.5 Cut Elimination

The cut rule (or Modus Ponens) allows an assumption φ to be derived and then used subsequently in the proof of some sequent

$$\dfrac{\Gamma\vdash\Delta,\varphi \quad \varphi,\Gamma'\vdash\Delta'}{\Gamma,\Gamma'\vdash\Delta,\Delta'}\,M.P.$$

or the single conclusioned version

$$\dfrac{\Gamma\vdash\varphi \quad \varphi,\Gamma'\vdash\theta}{\Gamma,\Gamma'\vdash\theta}$$

The cut rule allows the proof to proceed 'indirectly' through the cut formula φ. In systems with weakening and contraction, the following simpler forms are sufficient[2]:

$$\dfrac{\Gamma\vdash\Delta,\varphi \quad \varphi,\Gamma\vdash\Delta}{\Gamma\vdash\Delta} \qquad \dfrac{\Gamma\vdash\varphi \quad \varphi,\Gamma\vdash\theta}{\Gamma\vdash\theta}$$

The argument given to T_{cut} will be a minimal occurrence of the cut rule in a proof

[2] Assumptions, or conclusions, can be merged with contraction and introduced when required with weakening.

and will have the form

$$\dfrac{\overset{a}{\triangledown}\quad\overset{b}{\triangledown}}{\dfrac{\Gamma_1 \vdash \Delta_1, \varphi \quad \varphi, \Gamma_2 \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut}$$

where φ is the cut formula of the rule.

The argument is minimal, since the proofs a and b are guaranteed to be free of occurrences of the cut rule. The transformation must reduce the size of the proof according to the measure S_{cut}, whilst preserving similarity.

A few further definitions are needed. The *rank* of a cut formula φ is the lifetime of that formula - but not its subformulas - in the proof. The cut formula φ will have been introduced on the left- and right-hand sides at some point higher in the proof by rules that introduce the outmost connective of the formula. The rank is taken to be the sum of the maximum lengths of the left and right branches. The *degree* of a cut formula is the number of logical constants in the formula. Define the complexity of the cut rule, C_{cut} such that $C_{\text{cut}}(p) = \langle \text{rank of } \varphi, \text{degree of } \varphi \rangle$ and $C_{\text{cut}}(p_1) < C_{\text{cut}}(p_2)$ when

$$\langle r_1, d_1 \rangle < \langle r_2, d_2 \rangle \text{ iff } \begin{cases} d_1 < d_2 \\ d_1 = d_2 \text{ and } r_1 < r_2 \end{cases}$$

The action of T_{cut} depends on the rank and degree of the cut formula, φ, in the following way.

Case $\langle rank, degree \rangle$ of φ in

$\langle 0, 0 \rangle$ The proof will have the following form, when the structural rules are absorbed into the language.

$$\dfrac{\dfrac{}{\Gamma_1 \vdash \Delta_1, \varphi} \quad \dfrac{}{\varphi, \Gamma_2 \vdash \Delta_2}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut$$

This can be replaced with

$$\dfrac{}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, basic$$

For an intuitionistic system, the cut formula occupying the right-hand side of the sequent can be introduced by a rule for explicit weakening, *e.g.*

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma_1 \vdash \bot\end{array}}{\Gamma_1 \vdash \varphi}\vdash\bot \qquad \dfrac{}{\varphi, \Gamma_2 \vdash \varphi}\,basic}{\Gamma_1, \Gamma_2 \vdash \varphi}\,cut$$

which becomes

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma_1 \vdash \bot\end{array}}{\Gamma_1 \vdash \varphi}\vdash\bot}{\Gamma_1, \Gamma_2 \vdash \varphi}\,weakening$$

The implications of structural properties are discussed in §5.6.

$\langle r, d\rangle, r > 0$ Reduce the *rank* of the cut formula. The following proof illustrates this for the formula φ where the first rule on the branch is an introduction of $a \wedge b$ on the right. Assume that φ was introduced somewhere in the body of the subproof numbered 1.

$$\dfrac{\dfrac{\begin{array}{c}\boxed{1}\\ \Gamma_1 \vdash a, \varphi, \Delta_1\end{array} \quad \begin{array}{c}\boxed{2}\\ \Gamma_2 \vdash b, \Delta_2\end{array}}{\Gamma_1, \Gamma_2 \vdash a \wedge b, \varphi, \Delta_1, \Delta_2}\vdash\wedge \qquad \begin{array}{c}\boxed{3}\\ \varphi, \Gamma_3 \vdash \Delta_3\end{array}}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash a \wedge b, \Delta_1, \Delta_2, \Delta_3}\,cut$$

This becomes (by exchanging the rules and some assumptions)

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\boxed{1}\\ \Gamma_1 \vdash a, \varphi, \Delta_1\end{array} \quad \begin{array}{c}\boxed{3}\\ \varphi, \Gamma_3 \vdash \Delta_3\end{array}}{\Gamma_1, \Gamma_3 \vdash a, \Delta_1, \Delta_3}\,cut \qquad \begin{array}{c}\boxed{2}\\ \Gamma_2 \vdash b, \Delta_2\end{array}}{\Gamma_1, \Gamma_3, \Gamma_2 \vdash a \wedge b, \Delta_1, \Delta_3, \Delta_2}\vdash\wedge}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash a \wedge b, \Delta_1, \Delta_2, \Delta_3}\,exch.$$

The second proof reduces the lifetime of φ by consuming it before the introduction of $a \wedge b$. The instance of the cut rule has been pushed one step out of the proof.

$\langle 0, d\rangle, d > 0$ Reduce the *degree* of the cut formula. The example below illustrates this

when the cut formula is $\varphi \wedge \psi$.

$$\dfrac{\dfrac{\overset{1}{\triangledown}\ \Gamma_1 \vdash \Delta_1, \varphi, \psi, \Delta_2}{\Gamma_1 \vdash \Delta_1, \varphi \wedge \psi, \Delta_2}\vdash\wedge \qquad \dfrac{\overset{2}{\triangledown}\ \Gamma_2, \varphi, \Gamma_3 \vdash \Delta_3 \quad \overset{3}{\triangledown}\ \Theta_1, \psi, \Theta_2 \vdash \Delta_4}{\Gamma_2, \Theta_1, \varphi \wedge \psi, \Gamma_3, \Theta_2 \vdash \Delta_3, \Delta_4}\wedge\vdash}{\Gamma_1, \Gamma_2, \Theta_1, \Gamma_3, \Theta_2 \vdash \Delta_1, \Delta_2, \Delta_3, \Delta_4}\ cut$$

When $\wedge\vdash$ and $\vdash\wedge$ is removed in favour of a second application of cut, this becomes

$$\dfrac{\dfrac{\dfrac{\overset{1}{\triangledown}\ \Gamma_1 \vdash \Delta_1, \varphi, \psi, \Delta_2 \quad \overset{2}{\triangledown}\ \Gamma_2, \varphi, \Gamma_3 \vdash \Delta_3}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_2, \psi, \Delta_3}\ cut \qquad \overset{3}{\triangledown}\ \Theta_1, \psi, \Theta_2 \vdash \Delta_4}{\Gamma_1, \Gamma_2, \Gamma_3, \Theta_1, \Theta_2 \vdash \Delta_1, \Delta_2, \Delta_3, \Delta_4}\ cut}{\Gamma_1, \Gamma_2, \Theta_1, \Gamma_3, \Theta_1 \vdash \Delta_1, \Delta_2, \Delta_3, \Delta_4}\ exch.$$

5.5.1 Reducing the rank

In the example that was given above, the cut rule was exchanged with the branching rule that introduced $a \wedge b$ on the right-hand side.

Now consider the following generalisation of this situation, in which a proof rule has the form:

$$\dfrac{\Gamma_1, \varphi_1, \ldots, \varphi_m, \Delta_1 \quad \cdots \quad \Gamma_n, \varphi_1, \ldots, \varphi_m, \Delta_n}{\Gamma_1, \ldots, \Gamma_n, f(\varphi_1, \ldots, \varphi_m), \Delta_1, \ldots, \Delta_n}\ f(\text{-}introduction)$$

where Γ_i, Δ_i are collection metavariables over the category, c; $f(\varphi_1, \ldots, \varphi_m)$ represents the element of c being introduced by the rule; and, $\varphi_1, \ldots, \varphi_m$ are the subformulas that occur in the antecedents of the rule.

For example, in the illustration given above take $c = \mathsf{Formula}$, $f = \wedge$, $m = 2$, $\varphi_1 = a$ and $\varphi_2 = b$, so $f(\varphi_1, \varphi_2) = a \wedge b$.

The $\vdash$'s have been omitted from the sequents to indicate that it does not matter on which side of the sequent the individual subformulas, or the introduced formula, fall. This reduces the number cases that have to be considered. A rule of this form can be combined with the cut formula on either the left or right of a cut rule. Without loss of generality, suppose that the cut formula occurs on the right:

$$(5.1)\qquad \dfrac{\dfrac{\overset{1}{\triangledown}\ \Gamma_1,\varphi_1,\ldots,\varphi_m,\Delta_1,\varphi \quad\cdots\quad \overset{n}{\triangledown}\ \Gamma_n,\varphi_1,\ldots,\varphi_m,\Delta_n}{\Gamma_1,\ldots,\Gamma_n,f(\varphi_1,\ldots,\varphi_m),\Delta_1,\ldots,\Delta_n}\,f \qquad \overset{n+1}{\triangledown}\ \varphi,\Gamma_{n+1}\vdash\Delta_{n+1}}{\Gamma_1,\ldots,\Gamma_{n+1},f(\varphi_1,\ldots,\varphi_m),\Delta_1,\ldots,\Delta_{n+1}}\,cut$$

This can be rearranged to become:

$$(5.2)\qquad \dfrac{\dfrac{\overset{1}{\triangledown}\ \Gamma_1,\varphi_1,\ldots,\varphi_m,\Delta_1,\varphi \qquad \overset{n+1}{\triangledown}\ \varphi,\Gamma_{n+1}\vdash\Delta_{n+1}}{\Gamma_1,\Gamma_{n+1},\varphi_1\cdots\varphi_m,\Delta_1,\Delta_{n+1}}\,cut \qquad\cdots\qquad \overset{n}{\triangledown}\ \Gamma_n,\varphi_1,\ldots,\varphi_m,\Delta_n}{\Gamma_1,\ldots,\Gamma_{n+1},f(\varphi_1,\ldots,\varphi_m),\Delta_1,\ldots,\Delta_{n+1}}\,f$$

The occurrence of the cut rule is now higher in the proof, thereby reducing the lifetime of φ, as required. The definition of S_{cut} gives $S_{\text{cut}}(5.2) < S_{\text{cut}}(5.1)$.

The construction above assumes that it is possible to rearrange the order of assumptions. For this *exchange* is needed for the collections involved, so SetOf and BagOf are sufficient here.

Notice that if φ occurs in more than one antecedent of f, the transformation will be required to introduce more than one occurrence of the cut rule, with consequent replication of the assumptions Γ_{n+1} and Δ_{n+1}. In order to recover the original conclusion of the proof, several steps of *contraction* are needed, which is implicit if the collection concerned is SetOf.

For systems which use BagOf, because the number of occurrences of an assumption is an important factor, the rule f shown above must be responsible for the contraction itself. Consequently, the problem of replication does not arise.

5.5.2 Reducing the Degree

When the lifetime of a cut formula has been reduced to zero, it stands in the conclusion of two rules with its subformulas in their antecedents. Thinking of the cut formula as an intermediate 'lemma,' it is clear that one of the rules introduces it and the other consumes, or eliminates it. This exposes a fundamental relationship between the introduction rules acting on the left and right-hand side of a sequent, and leads to a large number of subcases.

Determining the subcases

Using the information obtained from the analysis of rules (§5.2.14), and knowledge about the formation rules for the category, a number of subcases may immediately be formed. Suppose that for each connective, or pattern, occurring in the conclusion of rules, there is a matrix representing the introduction of the connective on the left- and right-hand sides of the sequent; the tick marks indicate the subcases to consider:

f	*right*
left	

$\wedge$	$\vdash\wedge_1$	$\vdash\wedge_2$
$\wedge\vdash$	$\surd$	$\surd$

$\vee$	$\vdash\vee$
$\vee\vdash_1$	$\surd$
$\vee\vdash_2$	$\surd$

$\Box$	*normal*
normal	$\surd$

There may be rules that introduce patterns on one side but not on the other. In this case, the connective can be considered as a *structural* rather than logical connective, since it can never occur in a cut formula. An example is the intuitionistic operator $\perp$ which is used to represent the absence of a formula on the right-hand side of an intuitionistic sequent[3]. (See §5.6 below.) When there is a potential for overlapping among rules, there is a proliferation of subcases, *e.g.*

$\neg$	$\vdash\neg$
$\neg\vdash$	$\surd$

$\neg\wedge$	$\vdash\neg\wedge$
$\neg\wedge\vdash$	$\surd$

which gives the following possible subcases:

$\neg$	$\vdash\neg$	$\vdash\neg\wedge$
$\neg\vdash$	$\surd$	$\surd$
$\neg\wedge\vdash$	$\surd$	$\surd$

and so forth. Each matrix gives rise to a number of possible combinations of rules for introducing a cut formula which is syntactically distinct from other matrices.

Transforming the subcases

Suppose the initial proof has the form:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ p_1\end{array}\quad\begin{array}{c}\vdots\\ p_n\end{array}}{\Gamma_1,\varphi_1,\ldots,\varphi_m,\Delta_1\cdots\Gamma_n,\varphi_1,\ldots,\varphi_m,\Delta_n\vdash f}\quad\dfrac{\begin{array}{c}\vdots\\ q_1\end{array}\quad\begin{array}{c}\vdots\\ q_l\end{array}}{\Pi_1,\varphi_1,\ldots,\varphi_m,\Theta_1\cdots\Pi_l,\varphi_1,\ldots,\varphi_m,\Theta_l\vdash f}}{\Gamma_1,\ldots,\Gamma_n,\Pi_1,\ldots,\Pi_l\vdash\Delta_1,\ldots,\Delta_n,\Theta_1,\ldots,\Theta_l}\ cut$$

For each subcase there must be a transformation that preserves the similarity of the proofs and reduces S_{cut}. This is accomplished by reducing the complexity of the cut

[3]Although $\perp$ is often used to define $\neg\varphi$ as $\varphi\rightarrow\perp$.

rule, as measured by C_{cut}. This technique is used to remove the original cut rule, introducing new cut rules acting on the constituents $\varphi_1, \ldots, \varphi_m$ of the cut formula $f(\varphi_1, \ldots, \varphi_m)$.

The strategy used for obtaining a transformation does not affect any of the subproofs, $p_1, \ldots, p_n, q_1, \ldots, q_l$. Instead, a new derivation is constructed from them without using the logical rules $\vdash f$ and $f \vdash$, but using cut rules and structural rearrangements of the proof. The size of all the cut rules used in the new derivation will be less than the original cut being eliminated because, as the new cut formulas are subformulas of the original cut formula, they will each have smaller degree.

To see how to construct the transformations required for each of the subcases, consider the distribution of the subformulas $\varphi_1, \ldots, \varphi_m$ in the antecedents of the rules $\vdash f$ and $f \vdash$. Let these be called $A_1, \ldots, A_n$ and $B_1, \ldots, B_l$ respectively.

Let φ' be a *counterpart* of φ if φ' occurs on the opposite side of a sequent to φ and $\varphi = \varphi'$. For the elimination to be successful, all occurrences of $\varphi_1, \ldots, \varphi_m$ in $A_1, \ldots, A_n$ must have counterparts in $B_1, \ldots, B_l$, and vice versa.

If this were not the case there would be some φ_i that could not be cut from the proof, and the construction of the transformation would fail.

Suppose there are more occurrences of φ_i in $A_1, \ldots, A_n$ (resp. $B_1, \ldots, B_l$) than there are in $B_1, \ldots, B_l$ (resp. $A_1, \ldots, A_n$) then, provided *contraction* is available - *i.e.* with SetOf, we can make appropriately more occurrences of φ_i in $B_1, \ldots, B_l$ (resp. $A_1, \ldots, A_n$).

If the collection metavariables are ignored for the moment, then concentrating on the subformulas in the derivation, it is possible to construct a skeletal proof as follows:

1. Form a list of candidate subgoals $G = [A_1; \ldots; A_n; B_1; \ldots; B_l]$, adding additional copies to balance the occurrences of counterparts appropriately. If this is not possible for structural reasons, report failure.

2. Remove a pair of subgoals, $\langle A_j; B_k \rangle$, containing a pair of counterparts, from the candidates available.

3. Form an application of a cut rule from them:

$$\frac{A_j \quad B_k}{C}\, cut$$

4. Add the conclusion of the cut rule to the list of candidate subgoals.

5. Stop when G has the form $\{\vdash\}$. Otherwise go to step 2.

This algorithm assumes that where choices are possible, the correct choice is made - daemonic non-determinism. In practice, this can be achieved through the less efficient non-determinism supplied by PROLOG's ability to enumerate the (finite) search space through backtracking.

As an example of the algorithm, consider the rules given in the example on page 153. In that example the subproofs concluded in:

$$\Gamma_1 \vdash \Delta_1, e, f, \Delta_2 \quad \Gamma_2, e, \Gamma_3 \vdash \Delta_3 \quad \Theta_1, f, \Theta_2 \vdash \Delta_4$$

when collection metavariables are ignored, give the initial candidates for G:

$$\vdash e, f \quad e \vdash \quad f \vdash$$

Applying the algorithm described above produces the following steps:

step	G	$\langle A; B \rangle$	*skeleton proof*
1.	$\{\vdash e, f; e \vdash; f \vdash\}$	$\langle \vdash e, f; e \vdash \rangle$	$\dfrac{\vdash e, f \quad e \vdash}{\vdash f}$
2.	$\{\vdash f; f \vdash\}$	$\langle \vdash f; f \vdash \rangle$	$\dfrac{\dfrac{\vdash e, f \quad e \vdash}{\vdash f} \qquad f \vdash}{\vdash}$
3.	$\{\vdash\}$	finished.	

To replace the original subgoals, it is possible to use implicit structural rules, the form of the cut rule, and the matching algorithm, to recover the full schematic version of the proof:

$$\dfrac{\dfrac{\overset{1}{\triangledown} \qquad \overset{2}{\triangledown}}{\dfrac{\Gamma_1 \vdash \Delta_1, e, f, \Delta_2 \quad \Gamma_2, e, \Gamma_3 \vdash \Delta_3}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_2, f, \Delta_3} \qquad \overset{3}{\triangledown}}{\Theta_1, f, \Theta_2 \vdash \Delta_4}}{\Gamma_1, \Gamma_2, \Gamma_3, \Theta_1, \Theta_2 \vdash \Delta_1, \Delta_2, \Delta_3, \Delta_4}}{\Gamma_1, \Gamma_2, \Theta_1, \Gamma_3, \Theta_1 \vdash \Delta_1, \Delta_2, \Delta_3, \Delta_4}$$

Exchange is needed to achieve this. Other systems may require some contraction. The example was taken from Linear Logic which does not have contraction.

5.6 Explicit Structural Rules

The environment favours the formulation of systems with implicit structural rules. However, it is sometimes convenient to make a structural rule explicit.

5.6.1 Weakening

As noted above (§5.4.1), weakening rules are derived rules in systems with 'generalised' basic sequents. In intuitionistic systems, weakening on the right-hand side of a sequent is only possible when there is no formula present; $\bot$ denotes 'no formula'.

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}\vdash thin \qquad \frac{\varphi, \Gamma \vdash \bot}{\Gamma \vdash \neg\varphi}\vdash\neg \qquad \frac{\Gamma \vdash \varphi}{\neg\varphi, \Gamma \vdash \bot}\neg\vdash$$

In such systems, there are rules that interact with $\bot$ and $\neg$: The rule $\vdash thin$ weakens the right-hand side by introducing an arbitrary formula in place of $\bot$. A negated formula may be introduced on the right-hand side using an assumption on the left, provided that there is no formula present. Finally, a $\bot$ is introduced on the right by $\neg\vdash$. Note that $\bot$ is not introduced on the left.

There is clear indication that the role of $\bot$ is structural, rather than logical, as it may only be introduced the right-hand side, and therefore cannot be eliminated from a cut formula.

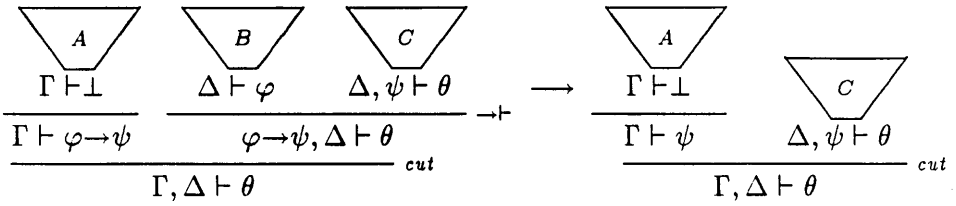
Consider the elimination of a formula introduced by the weakening rule $\vdash thin$, taking the cut formula to be $\varphi\wedge\psi$:

$$\dfrac{\dfrac{\begin{array}{c}A\\ \Gamma \vdash \bot\end{array}}{\Gamma \vdash \varphi\wedge\psi}\vdash thin \qquad \begin{array}{c}B\\ \varphi\wedge\psi, \Delta \vdash \theta\end{array}}{\Gamma, \Delta \vdash \theta}\, cut$$

The left rank of $\varphi\wedge\psi$ is zero, and transformation to reduce the right rank is straightforward, using techniques discussed earlier. Once the right rank is reduced to zero, the proof transformation required to reduce the degree of the cut formula depends on its structure. In this case, the formula is $\varphi\wedge\psi$ and the transformation can be taken as follows:

$$\dfrac{\dfrac{\begin{array}{c}A\\ \Gamma \vdash \bot\end{array}}{\Gamma \vdash \varphi\wedge\psi}\vdash thin \qquad \dfrac{\begin{array}{c}B\\ \varphi, \psi, \Delta \vdash \theta\end{array}}{\varphi\wedge\psi, \Delta \vdash \theta}\wedge\vdash}{\Gamma, \Delta \vdash \theta}\, cut \quad \longrightarrow \quad \dfrac{\dfrac{\begin{array}{c}A\\ \Gamma \vdash \bot\end{array}}{\Gamma \vdash \varphi}\vdash thin \qquad \dfrac{\dfrac{\begin{array}{c}A\\ \Gamma \vdash \bot\end{array}}{\Gamma \vdash \psi}\vdash thin \qquad \begin{array}{c}B\\ \varphi, \psi, \Delta \vdash \theta\end{array}}{\varphi, \Gamma, \Delta \vdash \theta}\, cut}{\Gamma, \Delta \vdash \theta}\, cut$$

This transformation suggests an extension to the algorithm outlined above. Specifically, the rule $\vdash thin$ can be used to obtain an unlimited source of correspondents for formulas occurring on the left-hand side, and as the following example shows, derivations of correspondents on the right-hand side can be dropped, *e.g.* B in the following:



The effect on the procedure described on page 156 is that when a proof concludes with a rule that introduces some formula, say $f(\varphi_1, \ldots, \varphi_m)$, with constituents, φ_i, that do not occur in the antecedents of the rule, the transformation can make arbitrary instantiations to the φ_i without affecting the proof above the conclusion.

5.6.2 Contraction and Exchange

The two structural properties, if not coded implicitly through collections, can be stated explicitly over ListOf collections:

$$\frac{\Gamma, \varphi, \psi, \Gamma' \vdash \Delta}{\Gamma, \psi, \varphi, \Gamma' \vdash \Delta}\, exch\vdash \qquad \frac{\Gamma \vdash \Delta, \varphi, \psi, \Delta'}{\Gamma \vdash \Delta, \psi, \varphi, \Delta'}\, \vdash exch \qquad \frac{\Gamma, \varphi, \varphi, \Gamma' \vdash \Delta}{\Gamma, \varphi, \Gamma' \vdash \Delta}\, contr\vdash \qquad \frac{\Gamma \vdash \Delta, \varphi, \varphi, \Delta'}{\Gamma \vdash \Delta, \varphi, \Delta'}\, \vdash contr$$

However, when this is done, the transformations become complicated by the need to treat these structural rules as rules in their own right. Since these rules may occur at any point in a proof, they hugely increase the number of cases to consider when constructing transformations which reduce the rank and degree of a cut formula. Moreover, appropriate combinations of such rules need to be constructed as 'glue', composed of permutations and contractions, depending on the precise structure of the proof. These two disadvantages are avoided with implicit formulations.

5.7 Rule involving patterns

Rules containing patterns can also be treated; such rules are mostly used in the presentations of modal logics. The modal rules presented in chapter 3 often include weakening in their formulation *e.g.* Γ', Δ in the rule

$$\frac{\Gamma \vdash \varphi}{\Gamma', \Box\Gamma \vdash \Box\varphi, \Delta}\, normal$$

This could be restated with explicit weakening as:

$$\dfrac{\dfrac{\Gamma \vdash \varphi}{\Box\Gamma \vdash \Box\varphi}\,\textit{normal}'}{\Gamma', \Box\Gamma \vdash \Box\varphi, \Delta}\,\textit{weaken}$$

In the context of the environment, the first approach is preferable to the presentation including explicit weakening. Although this increases the number of cases to consider, the arguments used in §5.6.1 may be applied to make the task more manageable.

As an example, consider the modal system formed by extending propositional classical logic with a single rule *normal* (page 143). For '$\Box$' to be treated as a logical operator occurring within a cut formula, it must be introduced on both sides of the sequent. The only rule available to do this is *normal*. Observe that this rule *necessarily* introduces '$\Box\varphi$' on the right-hand side, and may *possibly* introduce a number of $\Box$'ed formulas as part of the modal pattern on the left-hand side. A cut formula can be pushed through an application of a pair of normal rules as the following example illustrates:

$$\dfrac{\dfrac{\overset{A}{\Gamma_1 \vdash \varphi}}{\Box\Gamma_1, \Gamma_2 \vdash \Box\varphi, \Delta_1}\,\textit{normal} \qquad \dfrac{\overset{B}{\varphi, \Gamma_3 \vdash \psi}}{\Box\varphi, \Box\Gamma_3, \Gamma_4, \vdash \Box\psi, \Delta_2}\,\textit{normal}}{\Box\Gamma_1, \Gamma_2, \Box\Gamma_3, \Gamma_4 \vdash \Delta_1, \Delta_2, \Box\psi}\,\textit{cut}$$

is transformed to

$$\dfrac{\dfrac{\overset{A}{\Gamma_1 \vdash \varphi} \quad \overset{B}{\varphi, \Gamma_3 \vdash \psi}}{\Gamma_1, \Gamma_3 \vdash \psi}\,\textit{cut}}{\Box\Gamma_1, \Gamma_2, \Box\Gamma_3, \Gamma_4 \vdash \Delta_1, \Delta_2, \Box\psi}\,\textit{normal}$$

5.7.1 Rank reduction

A cut formula cannot pass through the *normal* modal rule unchanged, as the rule forms the upper sequent by removing the outermost $\Box$ from all assumptions and the conclusion of the lower sequent.

As an example, consider the following derivation of $\Box\varphi\wedge\Box\psi \vdash \Box\varphi$ involving the cut formula $\Box(\varphi\wedge\psi)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\varphi \vdash \varphi \quad \psi \vdash \psi}{\varphi, \psi \vdash \varphi\wedge\psi}\,\vdash\wedge}{\Box\varphi, \Box\psi \vdash \Box(\varphi\wedge\psi)}\,K}{\Box\varphi\wedge\Box\psi \vdash \Box(\varphi\wedge\psi)}\,\wedge\vdash \qquad \dfrac{\dfrac{\varphi, \psi \vdash \varphi}{\varphi\wedge\psi \vdash \varphi}\,\wedge\vdash}{\Box(\varphi\wedge\psi) \vdash \Box\varphi}\,K}{\Box\varphi\wedge\Box\psi \vdash \Box\varphi}\,\textit{cut}$$

The rank of the right occurrence of the cut formula, $\Box(\varphi\wedge\psi)$, is zero, but the rank

of the left occurrence of the cut formula is one. This is due to the presence of the $\wedge\vdash$ rule between *cut* and K. Here, the rank of the right-hand occurrence is reduced by pushing the rule $\wedge\vdash$ beneath the cut rule:

$$\dfrac{\dfrac{\dfrac{\dfrac{\varphi \vdash \varphi \quad \psi \vdash \psi}{\varphi, \psi \vdash \varphi \wedge \psi}\vdash\wedge}{\Box\varphi, \Box\psi \vdash \Box(\varphi\wedge\psi)}K \qquad \dfrac{\dfrac{\varphi, \psi \vdash \varphi}{\varphi\wedge\psi \vdash \varphi}\wedge\vdash}{\Box(\varphi\wedge\psi) \vdash \Box\varphi}K}{\Box\varphi, \Box\psi \vdash \Box\varphi}cut}{\Box\varphi\wedge\Box\psi \vdash \Box\varphi}\wedge\vdash$$

Now the degree can be reduced using the strategy given above. The transformation needed for a rule like $K4$ that carries $\Box$'d assumptions forward is similar.

$$\dfrac{\Gamma, \Box\Gamma \vdash \varphi}{\Box\Gamma, \Gamma' \vdash \Box\varphi, \Delta}K4$$

An analogous example to the one just given, but for the rule $K4$ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\varphi \vdash \varphi \quad \psi \vdash \psi}{\varphi, \Box\varphi, \psi, \Box\psi \vdash \varphi\wedge\psi}\vdash\wedge}{\Box\varphi, \Box\psi \vdash \Box(\varphi\wedge\psi)}K4 \qquad \dfrac{\dfrac{\dfrac{\varphi, \psi, \Box(\varphi\wedge\psi) \vdash \varphi}{\varphi\wedge\psi, \Box(\varphi\wedge\psi) \vdash \varphi}\wedge\vdash}{\varphi\wedge\psi, \Box(\varphi\wedge\psi) \vdash \Box\varphi}K4}{\Box(\varphi\wedge\psi) \vdash \Box\Box\varphi}K4}{\Box\varphi, \Box\psi \vdash \Box\Box\varphi}cut$$

This becomes

$$\dfrac{\dfrac{\dfrac{\dfrac{\varphi \vdash \varphi \quad \psi \vdash \psi}{\varphi, \Box\varphi, \psi, \Box\psi \vdash \varphi\wedge\psi}\vdash\wedge}{\Box\varphi, \Box\psi \vdash \Box(\varphi\wedge\psi)}K4 \qquad \dfrac{\dfrac{\varphi, \psi, \Box(\varphi\wedge\psi) \vdash \varphi}{\varphi\wedge\psi, \Box(\varphi\wedge\psi) \vdash \varphi}\wedge\vdash}{\Box(\varphi\wedge\psi) \vdash \Box\varphi}K4}{\varphi, \Box\varphi, \psi, \Box\psi \vdash \Box\varphi}cut}{\Box\varphi, \Box\psi \vdash \Box\Box\varphi}K4$$

and then

$$\dfrac{\dfrac{\dfrac{\dfrac{\varphi \vdash \varphi \quad \psi \vdash \psi}{\varphi, \Box\varphi, \psi, \Box\psi \vdash \varphi\wedge\psi}\vdash\wedge \qquad \dfrac{\varphi, \psi \vdash \varphi}{\varphi\wedge\psi \vdash \varphi}\wedge\vdash}{\varphi, \Box\varphi, \psi, \Box\psi \vdash \varphi}cut}{\varphi, \Box\varphi, \psi, \Box\psi \vdash \Box\varphi}K4}{\Box\varphi, \Box\psi \vdash \Box\Box\varphi}K4$$

Note that the rank of the right occurrence of $\Box(\varphi\wedge\psi)$ was zero, although it occurred throughout the branch. This allows *cut* to be pushed safely through the second $K4$ rule.

5.7.2 Subformula Property

When patterns are interpreted according to the revised subformula property given above, it is clear that not all the modal rules enjoy the subformula property, *e.g.*

those involving symmetry such as the modal system B. The offending formulas are shown in boxes in the following table. The use of a strategy limiting the number of rules not having the subformula property is helpful in such cases. This can be achieved by adding a global condition to a strategy, for example:

```
{b=5}<B,left,right>[BASIC]
```

sets the maximum number of occurrences of rules not having the subformula property to 5 in each branch.

Systems	$\Box$	$\Diamond$
DB	$\dfrac{\Gamma, \Diamond\Gamma', \boxed{\Diamond\Gamma''} \vdash \Delta, \Box\Delta', \boxed{\Box\Delta''}}{\Box\Gamma, \Diamond\Gamma', \Gamma'' \vdash \Diamond\Delta, \Box\Delta', \Delta''}$	
KB, DB, B	$\dfrac{\Gamma, \Diamond\Gamma', \boxed{\Diamond\Gamma''} \vdash \varphi, \Delta, \Box\Delta', \boxed{\Box\Delta''}}{\Box\Gamma, \Diamond\Gamma', \Gamma'' \vdash \Box\varphi, \Box\Delta, \Diamond\Delta', \Delta''}$	$\dfrac{\varphi, \Gamma, \Diamond\Gamma', \boxed{\Diamond\Gamma''} \vdash \Delta, \Diamond\Delta', \boxed{\Box\Delta''}}{\Diamond\varphi, \Box\Gamma, \Diamond\Gamma', \Gamma'' \vdash \Box\Delta, \Diamond\Delta', \Delta''}$

5.8 Consistency of a System

The result of the analysis of cut given above amounts to showing Gentzen's Haupstatz holds for the system in question. Gentzen's objective in developing his Haupstatz was to show the consistency of arithmetic. In the same way it is possible to deduce the consistency of a system presented in the environment as follows.

Consider a system in which the derivability of the empty sequent '$\vdash$' can be interpreted as the inconsistency of the system as a whole. This is the case for a system with weakening

$$\dfrac{\vdash}{\Gamma \vdash \Delta}\ \textit{weakening}*$$

If a system includes a formulation of the cut rule, then the derivation of the empty sequent must conclude either in one of the stated all-introduction rules of the system or otherwise through the cut rule.

In the first case, simple inspection of the rules in the system can ascertain whether this is possible, otherwise it is an occurrence of a cut rule. If the cut rule is known to be a derived rule, there is a transformation which can be applied to the proof that will eliminate all occurrences of the cut rule from the derivation of the empty sequent. This reduces the problem to the first case.

Thus, an inspection of the non-derived rules suffices to show whether it is possible to construct such a derivation. If it is not possible, then the system can be seen to be consistent.

5.9 Expressing Properties

If the system contains a judgement encoding a two-place relation, '$\vdash$', over formulas or the like, it may be desired to verify some of the properties proposed in §1.3.1. In particular if '$\vdash$' holds between left and right, the following 'types' of the relation may be expected:

		right			
		set	bag	list	singleton
	set	1			2
left	bag		3		4
	list			5	6

The typical uses of these are:

1. classical logics and classical extensions such as modal systems;
2. intuitionistic logics and their modal extensions;
3. relevant and classical linear logics;
4. intuitionistic linear logics;
5. multi-conclusioned logics with explicit structural rules;
6. single-conclusioned logics with explicit structural rules, type inference systems.

Other points in the matrix give rise to intermediate forms.

Reflexivity	1	2	3	4	5	6
$\dfrac{}{\varphi \vdash \varphi}$	√	√	√	√	√	√

Monotonicity	1	2	3	4	5	6
$\dfrac{\Gamma \vdash \Delta}{\varphi, \Gamma \vdash \Delta}$ $\dfrac{\Gamma \vdash \Delta}{\vdash \Delta, \varphi}$	√		√		√	
$\dfrac{\Gamma \vdash \psi}{\varphi, \Gamma \vdash \psi}$ $\dfrac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}$		√		√		√

Transitivity	1	2	3	4	5	6
$\dfrac{\Gamma \vdash \Delta, \varphi \quad \varphi, \Gamma \vdash \Delta}{\Gamma \vdash \Delta}$	√		√			
$\dfrac{\Gamma_1 \vdash \Delta_1, \varphi \quad \varphi, \Gamma_2 \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}$	√		√		√	
$\dfrac{\Gamma \vdash \varphi \quad \varphi, \Gamma \vdash \psi}{\Gamma \vdash \psi}$		√[a]				
$\dfrac{\Gamma_1 \vdash \varphi \quad \varphi, \Gamma_2 \vdash \psi}{\Gamma_1, \Gamma_2 \vdash \psi}$		√		√		

[a] with monotonicity

5.10 Conclusions

This chapter has provided an approach which can be formulated as part of the prototype environment and strategies have been described for verifying whether rules are derived with respect to a system. The distinction has been made between simply derived rules and recursively derived rules, also, it has been shown that a rule may be simply derived in the presence or absence of a cut rule in a system. A more complex, context dependent, analysis was required to verify the status of recursively derived rules.

Chapter 6

The Environment

This chapter gives an outline of the interface provided by the environment to a prospective user. Aspects of the interaction provided between the environment and the user are discussed. Details of some mechanisms, in particular, the specialised widgets that are used in the interface are presented.

6.1 Motivation

The realisation of the framework outlined in earlier chapters is the prototype environment described here. The environment is written in a version of the PROLOG language[DP89] that allows applications to be constructed using the X window system[GSN89] and its principal 'toolkit' [MAS89]. The portability of the *ω-prolog* and the X window system, allows the environment to be used on workstations from a variety of different computer manufacturers.

The interface makes use of the X toolkit graphical objects - known as *widgets* - provided by the MIT Athena widget set[Pet89]. Some of the author's own widgets (see §6.3) are also used.

One of the major design goals of the environment was that it should be usable by a wide class of users. The motivation for this came from the author's involvement in the GENESIS project, ESPRIT 1222(1041) [Har85], in particular the project's interface [Wra87].

The GENESIS project produced an environment that could be configured to manipulate a variety of formal systems. The environment was presented with a description of the concrete syntax of a formal system, in the form of a *projection scheme.*

It used this to produce a structure editor for the formal system. A user of the GENESIS environment is given the impression of manipulating a 'typeset' version of the text; abstract syntax is maintained internally and is equivalent to the concrete form presented to the user. Components of the environment have access to the internal abstract form, and can perform tasks related to semantic elements of the formal system, such as: type checking/inference, declaration before use, *etc.* The GENESIS environment is implemented in Sun's Common LISP.

A second source of motivation stems from the interface of the popular Apple Macintosh computer. The Macintosh supports a consistent and natural pattern of interaction - which is common to all its application programs. The philosophy behind the Macintosh user interface, and guidance for application builders, is given in detail in the "Inside Macintosh" books [Com85].

Using the X system to provide an interface with the quality of the Macintosh is difficult. This is partly because of the X consortium's objective that the system should be "policy free", and partly because of X system's comparative immaturity. Although the "policy free" approach has made the X system general, it is only in the last year or so that toolkit and widget writers have been able to achieve consistency in their interfaces approaching the Macintosh. There are now several competing approaches within the X community.

6.2 An Overview

The environment is presented to a user as a large workspace (see figure 6.1) surmounted by a title and a row of command buttons called the *control panel.* Three buttons allow the user to [Quit] from the environment, ask for [Help], or interrupt a computation by asking it to [Stop!]. The remaining two buttons invoke the [Output Window] and the [Special Characters] display. The Output Window, which is used by the system for making remarks to the user, is not shown initially and only makes an appearance when the user asks for it, or if it is needed to display text.

A pop-up menu contains a list of logical systems that are available to the user. The menu is structured as a number of overlapping 'cards' as shown in figure 6.2. The menu is activated by pressing the third mouse button[1] while the mouse cursor is directly over the main window of the workspace. Cards that are obscured can be made accessible by sliding the mouse over them, whereupon they pop to the top of the pack. A system is chosen by releasing the mouse button while its selection is

[1]This is usually the rightmost button, although X permits buttons numbers to be reassigned.

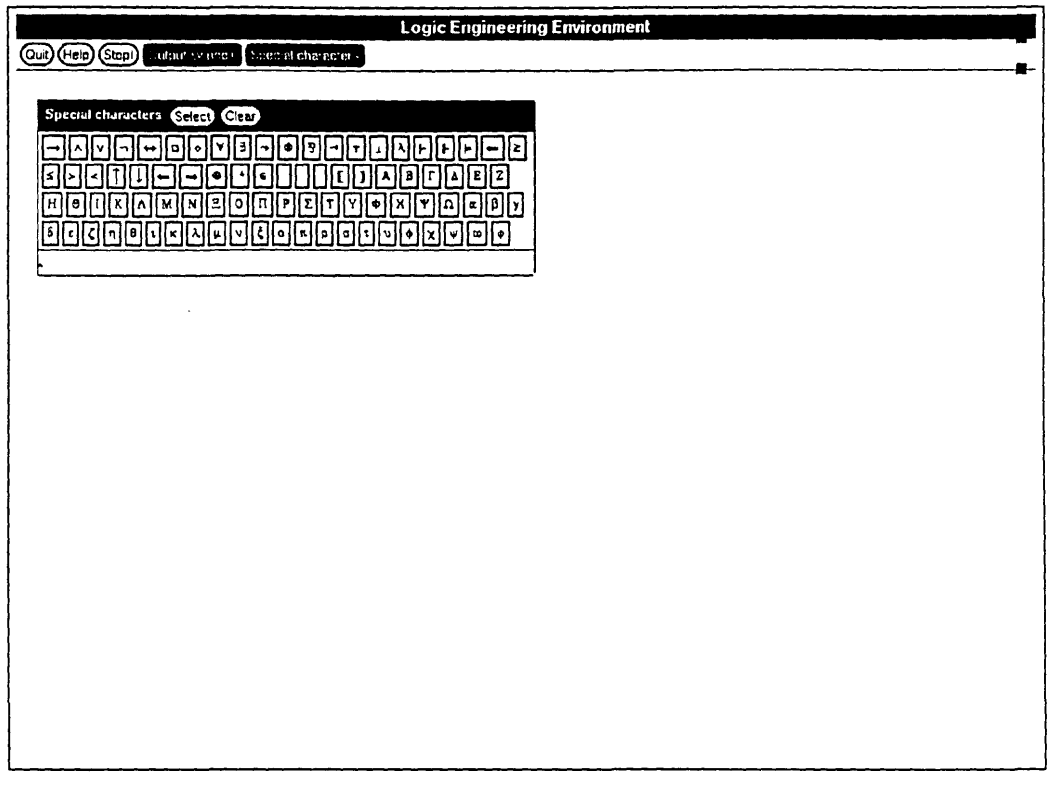


Figure 6.1: The Environment at Start-up

highlighted, as illustrated in the figure. The systems on the cards are located from libraries specified in the user's startup script.

Items such as the special character keyboard (shown in the figure) and the output window can be resized, moved, raised and lowered in the workspace. This is carried out using the mouse on a small titlebar which extends along the top of each object inhabiting the workspace, or on special square regions which are placed at its bottom left and bottom right-hand corners. (See figure 6.11.)

Objects can be hidden from view by clicking on the text in their titlebars *e.g.* "Special Characters" in figure 6.1. The object can be retrieved by pressing the corresponding button in the bar above the workspace.

When there are many objects of the same kind, such as proofs, these are given their own button on the control panel, *e.g.* Proofs. This controls a box of buttons referring to the individual proofs. The effect is to make the proliferation of buttons manageable. A user can easily navigate among the objects by locating the *type* of object that is of interest, and working from there.

Selecting an entry on the logics menu brings forth a presentation of the corresponding logic. A logic is presented as a window composed of at least five panes as shown

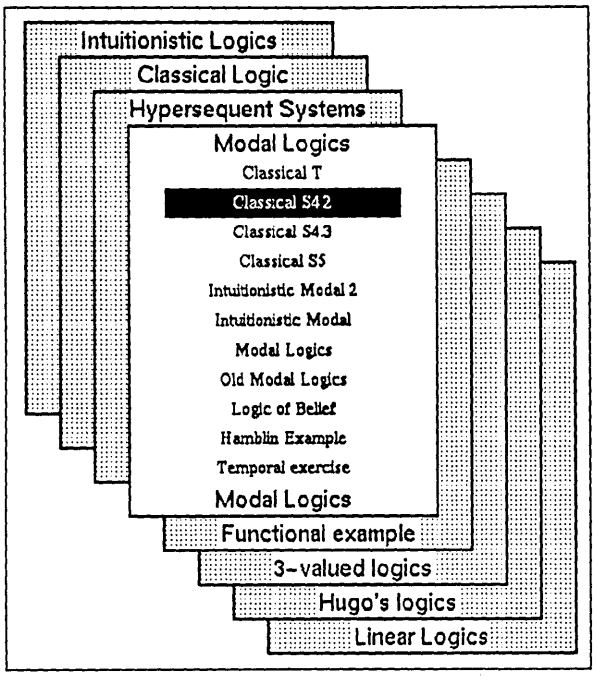


Figure 6.2: Example Logics Menu

by figure 6.3. The panes of the figure are usually coloured to make them easier to identify.

Each pane can be conveniently thought of as a pane in a sash window. The panes be made to slide up and down, by using the filled black squares shown down the right-hand side of the figure. However, the order of the panes cannot be changed.

Taking the panes in the figure from top to bottom the n^{th} pane contains:

1. a brief informal description of the system;
2. its language;
3. rules for the system;
4. examples to try out; and,
5. strategies to use.
6. (optionally) additional panes, *e.g.* side-conditions.

A pane's title may also offer various buttons which perform some computation over its contents. For example, the pane called "Rules" offers [Save], [Digest] and [Check]. Underneath all of these titled panes, at the bottom of the object, is a pane containing buttons which initiate actions dependent on the system as a whole.

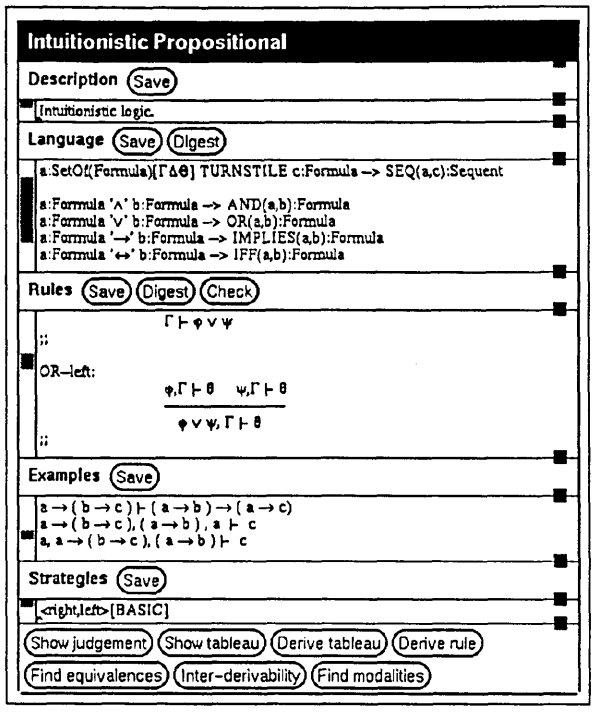


Figure 6.3: Logic Presentation

When a system is first opened, a button called Systems appears above the workspace and a new box of buttons titled as "Systems" is placed on the workspace. This box contains a button with the same name as the requested system. In the example in figure 6.4 this is called Intuitionistic Logic. As further logics are accessed further buttons are added to the "Systems" box.

Pressing the button Show Judgement invokes the dialog box shown in figure 6.5. The dialog consists of two questions and spaces for the answers to them. In this case they are: "Judgement?" and "Strategy?". As shown both are completed before the button Ok is pressed. In the figure, the judgement "$\vdash \neg\neg(((a \rightarrow b) \rightarrow a) \rightarrow a)$" has been chosen; together with a strategy "`<left,right>[BASIC]`". Dialogs in the environment have the same form: a list of questions to be completed, rather like a form.

The ease of supplying answers to the dialogs is enhanced by the ability to copy and paste text from one place to another using the workstation's mouse. Pressing the left mouse button at some position marks it as the start of a selection. Dragging[2] the mouse across an area of text selects the region covered. Similarly pressing the right button extends the start of the selection to the current position. The middle button is used to insert the current selection; this operation is known as 'pasting'. Copying the text is done implicitly when selection is made. Cutting a selected region

[2]usually taken to mean moving the mouse whilst keeping a button pressed down

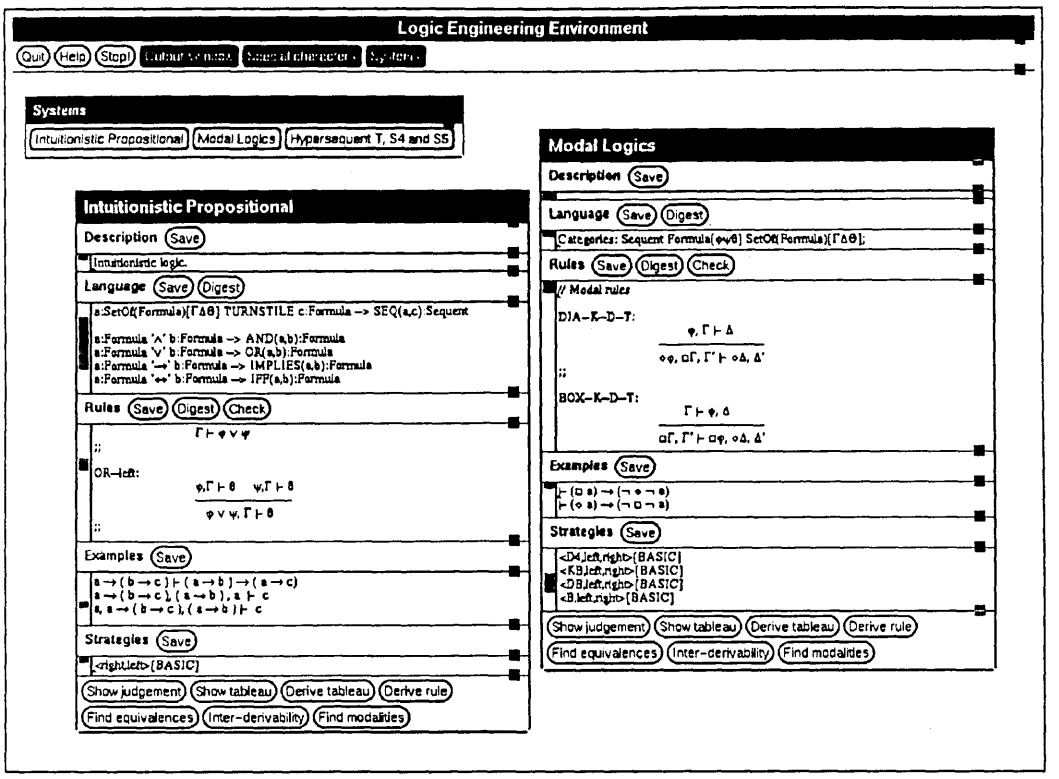


Figure 6.4: Systems Box

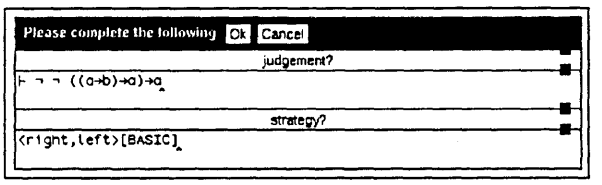


Figure 6.5: A Proof Dialog

is done with **cntrl-W** at the current position. The selected region is indicated by highlighting the characters within it.

The special character "keyboard" is also available for the entry of greek and mathematical characters not usually found on standard keyboards. The keyboard consists of two regions: the upper contains an array of buttons labeled with the

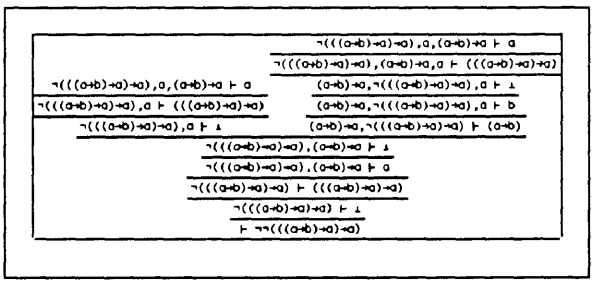


Figure 6.6: A Proof

special characters; below these, is a scratch area. When a button is pressed the character it represents is inserted into the scratch area. The keyboard's title includes a button Clear which deletes the contents of its scratch region, and a button Select that makes the contents of the scratch region the current selection.

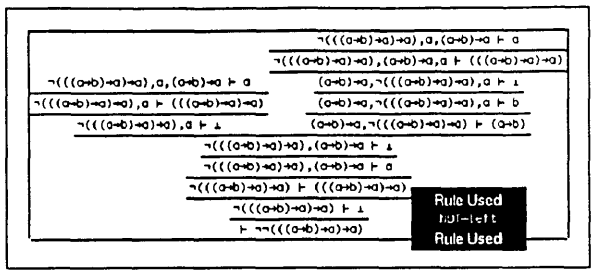


Figure 6.7: Identifying a Proof Rule

The completed proof is shown in figure 6.6. A proof contains additional information which is stored on pop-up menus associated with parts of it. Names of the inference rules used in the proof are placed on the pop-up menus. An inference rule's name is discovered by pressing the right button over its horizontal rule - see figure 6.7.

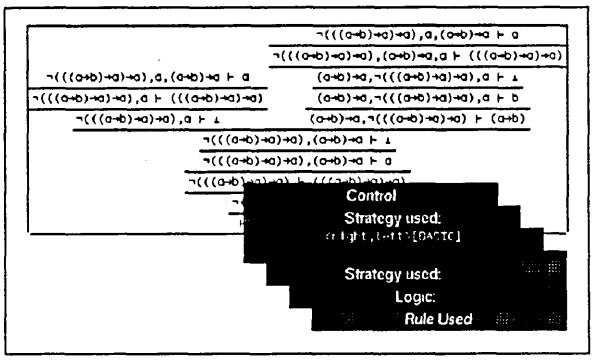


Figure 6.8: Inspecting a Proof

The pop-up menu associated with the outermost rule contains information about the logic and the strategy used, as well as providing a means of hiding or removing the proof - see figure 6.8.

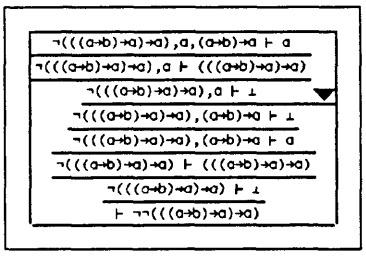


Figure 6.9: Proof Elision

Parts of a proof can be elided by pressing the left mouse button on their horizontal

rules - as shown in figure 6.9. Further details of the proof display widget are given below.

6.3 Widgets

It was necessary to add two widgets to the Athena widget set, to improve the interaction possible with the system. The first widget - called a "Mover" widget - is responsible for the placement of objects on the workspace. The second "Sequent" widget is responsible for the rendering of proof trees within the environment. The following sections give a brief description of these widgets.

6.3.1 Widgets

A *widget* in the terminology of the X toolkit is a combination of a rectangular graphical region with some specialised layout or rendering capability. Each of the objects visible to a user of the environment is a widget in this sense. A hierarchy of widgets is formed by the ability of some widgets to "manage" others. Both the widgets described below have this ability.

The PROLOG language in which the environment is constructed provides easy access to widgets written in the toolkit, and it is fairly straightforward to add further widgets if they are needed [DP89]. Some widgets are able to invoke capabilities of the PROLOG language by making *callbacks*. When a suitable event occurs, such as the pressing a mouse button while the mouse pointer is over a button, the widget can be programmed to make a callback. The callback is translated into a goal to be solved by the PROLOG system. A callback must be previously registered with a widget before it can be used. An application writer can obtain additional information by examining the contents of other widgets - typically widgets concerned with the manipulation of text. In this way there is a simple, controlled, interaction of the user with the environment.

A widget is an *object* in the sense of languages like C++. All widgets are subclasses of the "core" widget class. The core class defines the basic minimum of functions, or methods, that a widget must possess. Subclassing can increase the stock of functions available and, in this way the widgets form different lineages according to their inherited attributes. A widget also contains variables that define its current state. Some variables are local to the widget, others are made available for inspection through the toolkit. The toolkit provides mechanisms for setting and examining these public variables.

6.3.2 Proof displayer

The proof display widget - otherwise known as the sequent widget - constructs the layout of proof trees as shown in figure 6.6 above. Its objective is to format the proof in a readable way, so that horizontal lines are in their correct positions. It also provides a means of concealing parts of a large proof, and can be used with other widgets[3] to allow a large proof to be viewed comfortably when the display space available is restricted.

A sequent widget will create and manage other sequent widgets for any subproofs of a proof, such as that shown in figure 6.10.

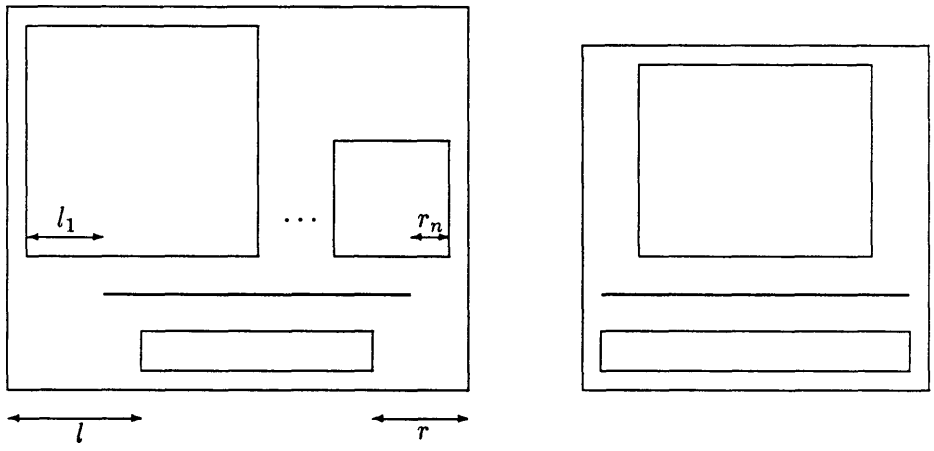


Figure 6.10: Sequent Widget Layout Strategy

A leaf of a proof tree is formatted according to the unparse scheme defined by the language of the system concerned. An internal node of the proof tree falls into one of the two cases illustrated in the figure: the subproofs are wider than the conclusion or they are not. If the subproofs are wider, the text of the conclusion, (formatted as in the same way as a leaf) is arranged so that is centered with respect to the representations of the subproofs. Otherwise the subproofs are centered with respect to the conclusion.

A sequent widget defines two publicly accessible attributes, which correspond to the dimensions of its left and right indents. These are the ls and rs in the figure. The indents are used to draw a horizontal line from the left indent of the leftmost subproof upto the right indent of the rightmost subproof. A sequent widget is used in this way for each application of an inference rule in the proof.

[3] most commonly the viewport widget

The display widget toggles its iconisation state when it receives a mouse press in the region of its horizontal rule, or its icon. Sequent widgets are constructed so that if one of them changes its size, the size change is propagated to its immediate neighbours. They in turn pass changes in their size to their neighbours, and so on. In this way the proof tree as whole, resizes itself when one of its constituents changes in size.

6.3.3 Workspace manager

The workspace management is performed by the "Mover" widget. This widget constructs a workspace in which its child widgets are placed. The children of the mover widget are each decorated with three (optional) mouse sensitive regions as shown in figure 6.11.

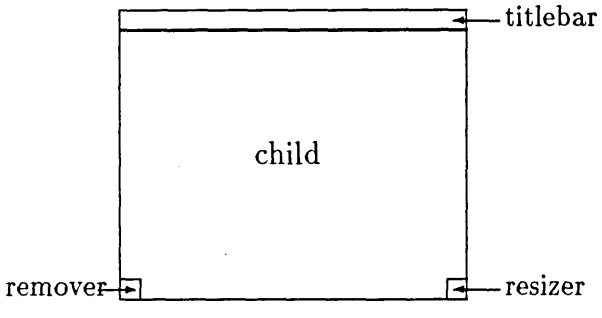


Figure 6.11: A Decorated Child of the Mover Widget

The regions are, from the top clockwise: a horizontal bar across the top of the child used to move it about the workspace; a square area on the bottom right of the child that is use to resize the child; and a second square area on the bottom left of the child that is usually used to remove - or destroy - the child permanently. The top region can also be used to raise of lower the child respect to the stacking order of its siblings. The stacking order indicates if a child lies above or below its siblings for purposes of display. It obscures siblings lower in the stacking order, while it is obscured by siblings higher in the stacking order. A child may not be visible if it is *unmanaged.* The selective management of widgets occupying the desktop is used to implement their conditional appearance and disappearance.

Chapter 7

Inside the Environment

This chapter gives details of the implementation of the environment. The prolog metalanguage in which the environment is written in is described and details are given of the internal mechanisms responsible for the construction of internal forms of a system's language and rules. These are used to show how, with the aid of heuristics, derivations of judgements are constructed.

7.1 Introduction

As illustrated in the previous chapter, the environment is constructed with a straightforward model of interaction that is simple for the user to use. From the users point of view the focus of attention is the system pane. If the user has permission, the contents of the system's panes can be edited directly. Several systems can be active at any one time. The environment performs actions when the user makes requests using buttons. Buttons are available at the top of the workspace, or within systems, they are attached either to the titles of panes or collected on a pane by themselves at the bottom of the system presentation.

The action of pressing a button in the interface forms a query in the metalanguage. The result of the query is reflected in the interface by creating a proof, or by writing text to a new or existing window. If the query arose from a button attached to a system then the query is given a context that describes the system concerned. When additional information is required it is collected using a dialogue box containing the relevant questions (*e.g.* figure 3.8 on page 81). Only when all the questions are answered satisfactorily does the action proceed.

The diagram in figure 7.1 shows how the environment can be thought of as divided into three parts. The environment constructs and manipulates objects existing to a greater or lesser extent in the *internal*, *interface* or *storage* part of this view. For example, the representation of a system consists of: (i) the external view presented by the interface to the user (see figure 6.3 on page 169); (ii) parts of this view providing a note-pad facility, which are mapped directly onto elements in the storage part; and (iii) other parts which are translated into prolog modules containing predicates that constitute a number of internal views of this information. This chapter is mostly devoted to a description of this last aspect.

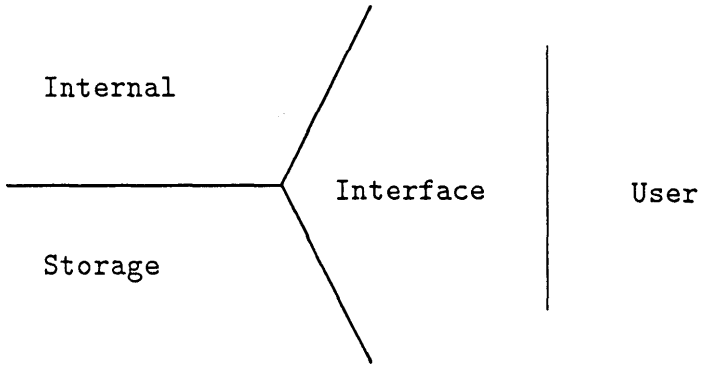


Figure 7.1: Schematic diagram of the environment

7.2 The Metalanguage

The environment is implemented in a variant of the prolog language [DP89] and, as such, this is the metalanguage of the environment. As some of this prolog code will be used in examples, a few remarks are required to explain how to read its slightly unusual syntax.

7.2.1 Clauses

A predicate is defined as a number of clauses and written as

```
def predicate ::
      argument ··· argument if
        goal argument ··· argument &
        ⋮
        goal argument ··· argument
    | argument ··· argument if
        goal argument ··· argument &
```

```
        ⋮
        goal argument ··· argument
    ;
```

where | and & are separators for clauses and goals respectively. The keyword **if** separates the arguments, which are terms, from the body of the clause when the body is not empty.

Example

The predicate **append/3**, expressing the relationship of concatenation between two lists, can be stated as

```
def append ::
      [] v v
    | e:es v e:fs if
        append es v fs
    ;
```

query	`append [1,2,3] [4,5] x`	`append x y [1,2]`
outcome(s)	x = `[1,2,3,4,5]`	x = `[]` y = `[1,2]` x = `[1]` y = `[2]` x = `[1,2]` y = `[]`

7.2.2 Predicates

Predicates are often distinguished by their arities. This is done by writing *predicate/arity*. When the name of a predicate appears in the body of a clause it is written as @*predicate/arity*. This allows the prolog compiler to replace the predicate with its fully qualified name (see below).

7.2.3 Variables

Logical variables are written in *italic* in the program texts here to distinguish them from their constant counterparts *e.g.* 'v' versus '*v*'. In the actual text of a prolog program, logical variables are declared explicitly using the `forall` construct in each clause.

7.2.4 Definite clause grammars

A useful means of eliding a particular form of an "accumulating parameter" in a predicate is provided by the so called "definite clause grammar extension" (DCGs). In this translation, the last two arguments of a predicate are constructed by the prolog compiler according to the following pattern:

```
def predicate ::=
      argument ··· argument if
        goal argument ··· argument &
        [ term ] &
        { goal' argument ··· argument } &
        goal argument ··· argument
    ;
```

becomes

```
def predicate ::
      argument ··· argument t0 t3 if
        goal argument ··· argument t0 t1 &
        $eq t1 term:t2 &
        goal' argument ··· argument &
        goal argument ··· argument t2 t3
    ;
```

in which $t_0, \ldots, t_3$ are new variables introduced by the translation. Any goals enclosed in {} are excluded from the translation (*e.g.* `goal'` above) and terms enclosed within [] are accumulated in the list (*e.g.* `term` above). Although this technique originated with the encoding of grammars in prolog, it can easily be used to construct or take apart the parameter list. This provides some of the convenience of higher-order functions in functional languages.

Example

The predicate nonEmptyCommaSeparatedList/3, shown below, parses a non-empty comma-separated list, represented by a list of tokens, and returns the underlying list without the commas. The list of tokens and the empty list are supplied as the initial and final values of the DCG's last two parameters.

```
def nonEmptyCommaSeparatedList ::=
      [t] if
        [ t ]
    | t:ts if
        [ t ',' ] & nonEmptyCommaSeparatedList ts
    ;
```

The predicate succeeds when given the following parameters

```
nonEmptyCommaSeparatedList [a,b,c] [a,",",b,",",c] []
```

succeeds

Note that the same predicate can be used to construct, rather than parse, the "unparsed" comma separated list. DCGs are used in the environment for the parsing and unparsing of language, rules and heuristic components of systems.

7.2.5 Modules

A module system for *ω-prolog* gives the predicate name space a structure. This is used to avoid name clashes between predicates in large programs.

The module system employed by this version of prolog operates by encoding the module hierarchy in the global name of a predicate. Distinct modules can thereby contain predicates with the same local name but they will not have identical global names.

Example

Suppose there is a module called 'a' with two sub-modules 'b' and 'c' each of which defines a predicate 'p' of arity 2. Then, the collapsing of the hierarchy means that 'p's name is 'a/b/p/2' and 'a/c/p/2' in the modules 'b' and 'c' respectively.

Visibility

The module system also restricts the visibility of predicates between modules, so that two modules having the same parent cannot use each other's predicates without explicitly making them available through the parent module.

Exporting predicates

A predicate may be exported by "promoting" its name in the name-space in the following way: suppose module 'a' exports predicate 'p' from its sub-module 'b', (this is quite legal in this system) then the name of 'p' changes from 'a/b/p/2' to 'a/p/2' - thereby giving it the appearance of being defined in 'a' rather than 'b'.

External representation

By convention, the module hierarchy is mirrored in the way the code is laid-out in the UNIX filesystem. Thus, the file containing the code for module prover would be called `prover.p`, and the subdirectory `prover` would contain code files for its sub-modules. Further details of the prolog system are given in [DP89].

7.3 Defining a System

The user's view of a system in the environment was described in the previous chapter (*e.g.* figure 6.3 on page 169). There, a system was presented as a collection of, at least, five panes: (i) a description, (ii) the language, (iii) its rules, (iv) examples, and (v) strategies. Of these (i) and (iv) are provided for the user's convenience and the others (ii), (iii) and (v) we shall describe here.

A system is regarded as a single entity, and this is reflected in its representation as a collection of files in a single directory whose corresponding internal representation is a prolog module.

7.3.1 Representation of a system

The environment represents a system as a module together with a directory containing a number of files. These files hold representations of the language and rules of the system together with the surface presentations of the panes.

```
family.p - family of systems
family/  - contents of the family
        system1.p - system module
        system1/  - directory
                  lang.p rules.p - language and rule submodules
                  desc etc.      - files holding the contents of the panes
        system2.p - another system
        system2/
        ⋮
```

Figure 7.2: Storage representation of a family of systems

7.3.2 System menu

Each of the "cards" making up the menu of systems (shown in figure 6.2 on page 168) corresponds to a module in the metalanguage. This module ('family.p' in figure 7.2) contains the systems shown on the card as submodules.

A user indicates the names of the systems which are of interest by defining an environment variable called LOGICS, before starting the environment. The deck of cards shown in the earlier example was obtained by making the following assignment to LOGICS:

```
setenv LOGICS in:classical:hyper:modal:functional:threeValued:hugo:linear
```

Thus, the internal name given to a system reflects its location in this hierarchy of modules, as it is structured according to the module's lineage (§7.2.5)[1].

Each module, whether it defines a family or a single system, defines three predicates called Description/1, Contents/1 and Location/1.

Description/1 The description predicate maps the internal name of the module to the external name allocated by its creator.

Contents/1 The contents predicate names the sub-logics/families if they exist. The module is taken to denote a logic when there are no contents defined by this predicate.

Location/1 The location predicate gives the location of auxiliary files that define the contents of the panes and other files such as any LaTeX presentations of the system.

Systems are invoked by constructing the names of these predicates, from the prefixes given by the LOGICS environment variable. The "autoloading" facility of the prolog system then finds the definitions of the predicates when these exist. Further details of the autoloader are given in [DP89].

7.3.3 Submodules of a system

A system contains two submodules corresponding its language and rules. The modules define a number of predicates as follows:

[1]There is no restriction in principle to having families of families of systems and so forth, however in practice this is not possible because of the two-level nature of the present pop-up menus.

lang	rules
tokens/3	preds/1
form/4	AND_right/3
metavariables/2	*etc.*
negativeJudgements/1	
judgements/1	
equality/2	
equality/3	

The module **rules** defines a predicate for each rule and the predicate rules/1 gives the names of these predicates. The translations from the surface syntaxes to these modules are described in the following sections.

7.4 Defining a Language

The purpose of the language is three-fold. Firstly it gives a relationship between the internal and external views of the object language by providing parsing and unparsing schemes. Secondly, it defines the "parts of speech" used, namely syntactic categories and judgements. Thirdly, it provides enough syntactic machinery to define rules over these linguistic structures which it does this by specifying the names of metavariables for the categories. Metavariables need only be defined for categories that are referred to directly by rules.

The parsing and unparsing strategies employed provide a simple operator grammar with the ability to define distfix operators. This has the extra flexibility provided by the structure of the syntactic categories, and built-in constructions for sets, lists, and bags, *etc.* The unparsing scheme makes use of parentheses to clarify the output. However, the main purpose of the language is to allow the structure of a system to be defined concisely and easily, and to allow such definitions to be manipulated in a convenient way. The abstract syntax provided by the language definition is used internally in either the *CIF* or *TIF* forms described elsewhere. Figure 7.5 shows an extended Backus Naur Form for the syntax of a language description.

The language will be illustrated with the example shown in figure 7.3. Next to the title of the language are two buttons Save and Digest, which permit the contents of the window to be saved to the language file, or converted to an internal form.

7.4.1 Digesting

The language of a system must be *digested* before it can be used. This is done by pressing the Digest button associated with the language pane. If the language is changed, it must be digested to reflect the changes.

As the language is *digested*, a LaTeX file is produced containing tables which can be included in documentation associated with the system.

7.4.2 Components of Language

The definition of a language has four parts:

- identification of *syntactic categories* and their *metavariables* (line 1);
- identification of *judgements* (line 3);
- *formation rules* for syntactic categories (lines 5–13); and,
- description of *lexemes* (line 15).

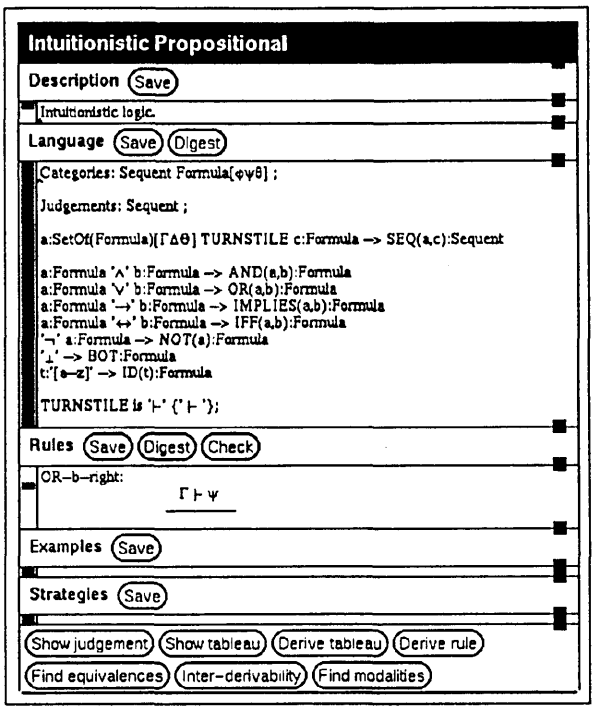


Figure 7.3: Presentation of a language

```
Categories: Sequent Formula[φψθ] ;                                    1

Judgements: Sequent ;                                                 3

a:SetOf(Formula)[ΓΔΘ] TURNSTILE c:Formula → SEQ(a,c):Sequent          5

a:Formula '∧' b:Formula → AND(a,b):Formula                            7
a:Formula '∨' b:Formula → OR(a,b):Formula                             8
a:Formula '→' b:Formula → IMPLIES(a,b):Formula                        9
a:Formula '↔' b:Formula → IFF(a,b):Formula                           10
'¬' a:Formula → NOT(a):Formula                                       11
'⊥' → BOT:Formula                                                    12
t:'[a-w]' → ID(t):Formula                                            13

TURNSTILE is '⊢' {'⊢'};                                              15
```

Figure 7.4: Language of intuitionistic logic

7.4.2.1 Syntactic categories

Line 1 We need two syntactic categories for the example of Intuitionistic Logic. The first is *Sequent* and the second *Formulas*.

7.4.2.2 Judgements

Line 3 Sequents are judgements here. Judgements are designated syntactic categories. Their role is to separate out those categories that are *external* from the remaining *internal* ones. Judgement categories have a special role as they are allowed to occur outermost in rules and to form queries, whereas ordinary categories are not.

Negative judgements can be declared; these are mostly used for modelling systems of defeasible reasoning §3.7. The declaration is similar to judgements. There are no negative judgements in the example.

7.4.2.3 Formation Rules

Lines 5–13 The formation rule for Sequent is shown first:

a:SetOf(Formula)[ΓΔΘ] TURNSTILE c:Formula → SEQ(a,c):Sequent

This is read as: if **a** is a set of *Formula* (with metavariables starting with ΓΔΘ) and it is followed by a lexeme called TURNSTILE and then by a formula **c**, then SEQ(a,c) is a *Sequent.* This defines an infix operator TURNSTILE, which maps a set of formulas and a formula to a sequent. The environment uses this information to parse and unparse the phrase, as well to extract its abstract syntax. In addition to sets, bags and lists are also provided: BagOf, ListOf. These are used to cope with various types of structural rules.

Formation rules for Formulas are similar. Line 13 `t:'[a-w]'` determines that identifiers start with a greek letter, ($\varphi\psi\theta$ being used for meta-variables).

7.4.2.4 Lexemes

Line 15 provides a map from names to character strings in logeng font - a font which includes many mathematical and logical characters.

TURNSTILE is '⊢' {' ⊢ '};

indicates that the lexeme TURNSTILE presents the string '⊢' for input and ' ⊢ ' for output. The { ··· } is optional and if omitted, input and output tokens are identical.

7.4.2.5 Metavariables

Metavariables may be declared either when the category is defined or when a collection is used. The example shows both of these. Line 5 contains:

a:SetOf(Formula)[ΓΔΘ]

which declares metavariables over a at that point. This is useful in that it makes it unnecessary to introduce an intermediary category just to record the metavariables.

7.4.3 Translation of Language

Within the prolog part of the environment, the language of a system is represented as a tuple consisting of the names of the six predicates defined by the system's language sub-module:

$$(\underbrace{\text{tokens/4}}_{\S 7.4.4}, \underbrace{\text{form/4}}_{\S 7.4.5}, \underbrace{\text{metavariables/2}}_{\S 7.4.6}, \underbrace{\text{judgements/1}, \text{negativejudgements/1}}_{\S 7.4.7}, \underbrace{\text{equality/2}}_{\S 7.4.8})$$

The predicates are formed from the language as follows. In the first part of the translation, the contents of the language window are parsed according to the grammar of figure 7.5. The parse tree so constructed is restructured and checked for consistency, before being formed into the definitions of the predicates listed above.

In particular, *lexemes* are checked for duplicate definition. An input lexeme must not contain any white space. *Categories* are collected and at least one category must be defined. Similarly for *judgements*, at least one judgement category must be defined, and this must be valid. *Negative judgements* can also be declared. Each *formation rule* is analysed to check that each variable appears exactly once on both sides of the definition. All categories that have been used must have been declared. In addition, there must be at least one formation rule for each category.

7.4.4 Token predicate

tokens/4 - are extracted and collated from the formation rules.

```
def tokens ::
        TURNSTILE "⊢" " ⊢ "
    ;
```

7.4.5 Formation Rules

All formation rules are collated into a single predicate, form/4. Each clause of the predicate corresponds to a single formation rule and consists of:

1. the category of the element the clause forms;
2. the class of the formation rule: prefix, infix, constant, *etc.*;
3. the *CIF* term generated by the formation rule;
4. a list of conditions that must be satisfied in order to form the element.

The first and third parameters constitute the right-hand side of the formation rule, The forth parameter corresponds to the left-hand side. The second parameter is used to guide the parse/unparse algorithm. A list of conditions may be formed from:

`token(`*c*`)` where *c* is a constant.

`lexeme(`*l*`)` where *l* is the name of a lexeme.

`regexp(`*re*`,`*t*`)` where *t* is a constant matching the regular expression *re*.

`number(`*n*`)` where *n* is a number.

`type(`*c*`,`*o*`,`*m*`,`*t*`)` where *t* is an element of the category *c*, supposing a local choice of metavariables *m* and display options *o*. The metavariables are denoted by a regular expression as before. The display options allow the element to be displayed on a named pop-up menu, or ignored. For collections a separator other than ',' can be specified.

For the example above form/4 is:

```
def form ::
      Sequent infix [SEQ,a,c] [type(SetOf(Formula),[],"^[ΓΔΘ][0-9']*$" ,a),
        lexeme(TURNSTILE),type(Formula,[],NONE,c)]
    | Formula infix [AND,a,b] [type(Formula,[],NONE,a),token("∧"),
          type(Formula,[],NONE,b)]
    | Formula infix [OR,a,b] [type(Formula,[],NONE,a),token("∨"),
          type(Formula,[],NONE,b)]
    | Formula infix [IMPLIES,a,b] [type(Formula,[],NONE,a),token("→"),
          type(Formula,[],NONE,b)]
    | Formula infix [IFF,a,b] [type(Formula,[],NONE,a),token("↔"),
          type(Formula,[],NONE,b)]
    | Formula prefix [NOT,a] [token("¬"),type(Formula,[],NONE,a)]
    | Formula const BOT [token("⊥")]
    | Formula const [ID,t] [regexp("^[a-w]$",t)]
    ;
```

7.4.6 Metavariable predicate

References to metavariables are extracted and collated from the declarations of categories to form the predicate **metavariables/2.** The first argument to this is the category name and the second is a regular expression constructed to match a metavariable over the category. When a category has no variables, NONE is written instead

```
def metavariables ::
      Sequent NONE
    | Formula "^[φψθ][0-9']*$"
    ;
```

7.4.7 Judgement predicate

judgements/1 - a single argument consisting of a list judgement categories.

```
def judgements ::
      [Sequent]
    ;
```

When negative judgements are present in a system, the predicate negativejudgements/1 holds for them.

```
def negativejudgements ::
      [NSequent]
    ;
```

7.4.8 Equality Predicate

Each category in a language induces an appropriate notion of equality between its elements. For example, consider the category of Formulas defined as:

```
a:Formula '∧' b:Formula  →  AND(a,b):Formula
a:Formula '∨' b:Formula  →  OR(a,b):Formula
a:Formula '→' b:Formula  →  IMPLIES(a,b):Formula
          '¬' a:Formula  →  NOT(a):Formula
              a:"[a-z]"  →  PROP(a):Formula
```

For this category, equality of two elements is determined by the following predicate:

```
def Equality ::
      AND [a, b] [a', b'] if
        Equality a a' &
        Equality b b'
    | OR [a, b] [a', b'] if
        Equality a a' &
        Equality b b'
    | IMPLIES [a,b] [a', b'] if
        Equality a a' &
        Equality b b'
    | NOT [a] [a'] if
        Equality a a'
    | PROP [a] [a]
    ;
```

This is essentially structural, as no laws are involved, and this predicate can be more efficiently implemented by using the unification of the prolog metalanguage directly. Thus, the above code need not be generated but can be replaced with:

```
def Equality ::
      any a a
    ;
```

Where laws do arise, care must be taken to ensure that equality in the category is correctly implemented:

a:SetOf(Formula) '⊢' b:Formula → SEQUENT(a,b):Sequent

The SetOf and BagOf constructions require the use of specialised equality predicates. In this case, the equality predicate generated looks like:

```
def Equality ::
      SEQUENT [a,b] [a', b] if
        EqualSets Formula a a'
    ;
```

Where EqualSets/3 is defined as:

```
def EqualSets ::
      category e e' if
        ContainedIn e e' category &
        ContainedIn e' e category
    ;
```

and ContainedIn/3 depends on the representation of Sets.

Removing explicit equality

The formation rules of a language give rise to a dependency graph that reflects the use of one category by another through any **type** conditions for the category.

The equality predicates are computed from an analysis of this dependency graph. Initially, categories are flagged depending on whether they require explicit or implicit equality. Categories that make use of SetOf or BagOf collections are marked as explicit, other categories are marked as implicit. The flags are then propagated through the graph so that categories that refer indirectly to explicit categories themselves become explicit. Only categories that remain implicit at the end of this process are eligible for direct unification.

The equality predicate for the language in figure 7.4 is:

```
def equality ::
      op:a op:b if
        $cut &
        equality op a b
    | a a
    ;


def equality ::
        SEQ [a,c] [$a,$b] if
        EqualSets a $a @equality/2 &
        equality c $b
    | AND $c $c
    | OR $d $d
    | IMPLIES $e $e
    | IFF $f $f
    | NOT $g $g
    | ID $h $h
    ;
```

7.4.9 Typed internal form

Typed internal form (*TIF*) was first introduced in §5.2.11. There, it was defined as:

1. $\mathcal{E}^f_c(t_1, \ldots, t_n)$ for elements;
2. $\mathcal{C}_c(t_1, \ldots, t_n)$ for collections;
3. $\mathcal{V}_c(v)$ for metavariables;
4. $\mathcal{B}^v_c(t_1, \ldots, t_n)$ for binders; and,
5. $\mathcal{A}_c(t_1, t_2)$ for application of a binders to an argument.

where f is the name of a constructor, t_i is a *TIF* expression, c is the name of a category, and v is the name of a variable.

In the concrete representation used within the environment, further distinctions are made. $\mathcal{E}_c^f(t_1,\ldots,t_n)$ becomes E(c, f, $[t_1,\ldots,t_n]$), $n > 0$ and when $n = 0$, we write T(c, f). Similarly, $\mathcal{V}_c(v)$ is split according to whether c is a collection category or not. CMV(l, c, v) is written when it has the form $l(c)$, and MV(c, v) otherwise.

7.4.10 Concise internal form

Concise internal form is similar to *TIF* except it omits the typing information. *CIF* is generated directly from the formation rules given for the system. We can define the translation from *TIF* to *CIF* as:

$$\begin{aligned} CIF(\mathcal{E}_c^f(t_1,\ldots,t_n)) &= f(CIF(t_1),\ldots,CIF(t_n)) \\ CIF(\mathcal{C}_c(t_1,\ldots,t_n)) &= [CIF(t_1),\ldots,CIF(t_n)] \\ CIF(\mathcal{V}_c(v)) &= Metavariable(v) \end{aligned}$$

where f is the name of a constructor, t_i is a *TIF* expression, c is the name of a category, and v is the name of variable.

A *TIF* expression can be obtained by annotating one in concise internal form using the language description of the system (in module rules/internal).

7.4.11 Typing concise internal forms

This section describes the conversion of concise internal form to typed internal form. The predicate InternaliseJudgement/5 takes three main parameters and, as it is a DCG, also accumulates occurrences of metavariables on its last two arguments. The first parameter is a judgement in *CIF* format (*judgement*). The resulting *TIF* equivalent is placed in the second parameter (*Tjudgement*). The language of the system is passed as the third parameter.

```
def InternaliseJudgement ::=
      judgement Tjudgement language if
        { JudgementsOf language judgements &
          member judgeCat judgements } &
        InternaliseCategory judgeCat judgement Tjudgement language
```

where JudgementsOf/2 selects the names of judgement categories from the language of the system. Note that the conversion starts from a judgement category. If a metavariable is found then its occurrence is recorded by the DCG, otherwise the formation rules of the system's language are used to discover the name of the category of the expression.

```
def InternaliseCategory ::=
      cat Metavariable(cat,id) MV(cat,id) language if
        [ id ] & // convert meta-variable and indicate occurrence
        { $con cat } // not a collection category
    | cat Metavariable(list(cat),id) CMV(list,cat,id) language if
        [ id ] // convert collection metavariable and indicate occurrence
    | cat value Tvalue' language if
        { FormationRulesOf language cat fixity value details } &
        InternaliseDetails details Tdetails terminal language &
        { FormationRulesOf language cat fixity Tvalue Tdetails &
          convert Tvalue Tvalue' terminal cat }
    ;
```

In the above code, the predicate FormationRulesOf/5 selects formation rules from the system's language, and the convert/4 distinguishes between terminal and non-terminal elements. A terminal element is one that does not refer to elements of another category. The variable *terminal* is bound to NonTerminal by InteraliseDetail/4 only if a type condition occurs in the condition of the selected formation rule.

```
def convert ::
      MV(cat, id) MV(cat, id) terminal cat
    | value T(cat, value) Terminal cat
    | value E(cat, value) NonTerminal cat
    ;
```

The main work is to determine how the element has been constructed. This is done by considering each of the conditions in the left-hand side of the formation rule in turn. The predicate InternaliseDetails/6 iterates over this list of conditions using InternaliseDetail/6 to interpret the meaning of each condition as shown below.

```
def InternaliseDetails ::=
      [] [] terminal language
    | d:ds Td:Tds terminal language if
        InternaliseDetail d Td terminal language &&
        InternaliseDetails ds Tds terminal language
    ;

def InternaliseDetail ::=
      lexeme(lex) lexeme(lex) terminal language
    | token(tok) token(tok) terminal language
    | regexp(re, val) regexp(re, val) terminal language
    | type(list(cat), misc, mvs, value) type(list(cat), misc, mvs, C(list,cat,Tvalue))
                NonTerminal language if
        { member list [SetOf, ListOf, BagOf] } &&
        InternaliseCollection value Tvalue list cat language
    | type(cat, misc, mvs, value) type(cat, misc, mvs, Tvalue) NonTerminal language if
        InternaliseCategory cat value Tvalue language
    ;
```

```
def InternaliseCollection ::=
      [] [] list cat language
    | e:es Te:Tes list cat language if
        InternaliseItem e Te list cat language &
        InternaliseCollection es Tes list cat language
    where {
    def InternaliseItem ::=
          Metavariable(list(cat), id) CMV(list, cat, id) list cat language if
            [ id ]
        | Metavariable(cat, id) MV(cat, id) list cat language if
            [ id ]
        | element Telement list cat language if
            InternaliseCategory cat element Telement language
        ;
    }
```

```
Language ::= { Category-decl | Judgement-decl | N-Judgement-decl
             | Formation-rule | Lexeme-decl }+

Category-decl ::= Categories:
             { Category-name { Metavariables-decl } }+ ;

Judgement-decl ::= Judgements: { Category-name }+ ;

N-Judgement-decl ::= Negative judgements: { Category-name }* ;

Category ::= SetOf( Category-name )
           | BagOf( Category-name )
           | ListOf( Category-name )
           | Variable( Category-name )
           | Category-name

Category-name ::= Identifier

Formation-rule ::= Lhs -> Rhs

Lhs ::= { Var : Number
        | Var : Category { Metavariables-decl } { Misc-decl }
        | Var : Regular-expression
        | Single-quoted-string
        | Lexeme-name }+

Rhs ::= Uppercase-identifier { ( Elt { , Elt }* ) } : Category-name

Elt ::= Var { : Binding }

Binding ::= Var
          | ( Var { , Var}* )

Lexeme-decl ::= Lexeme-name is Single-quoted-string { Single-quoted-string } ;

Lexeme-name ::= Uppercase-identifier

Metavariables-decl ::= [ letters ]

Misc-decl ::= < Misc-item { , Misc-item }* >
Misc-item ::= Menu= Single-quoted-identifier
            | Sep= Single-quoted-identifier
            | Ignore

Var ::= Lowercase-letter
```

Figure 7.5: Extended BNF description of language

7.5 Defining Rules

An inference rule has the following form:

rule-name :

$$\frac{J_1 \quad \ldots \quad J_n}{J} \quad : \quad \textit{side-condition}$$

;;

The precise syntax of the judgements naturally depends of the language defined for the system in question. Figure 7.7 shows the extended Backus Naur Form for rules and figure 7.6 gives an example of the format of rules taken from Intuitionistic logic.

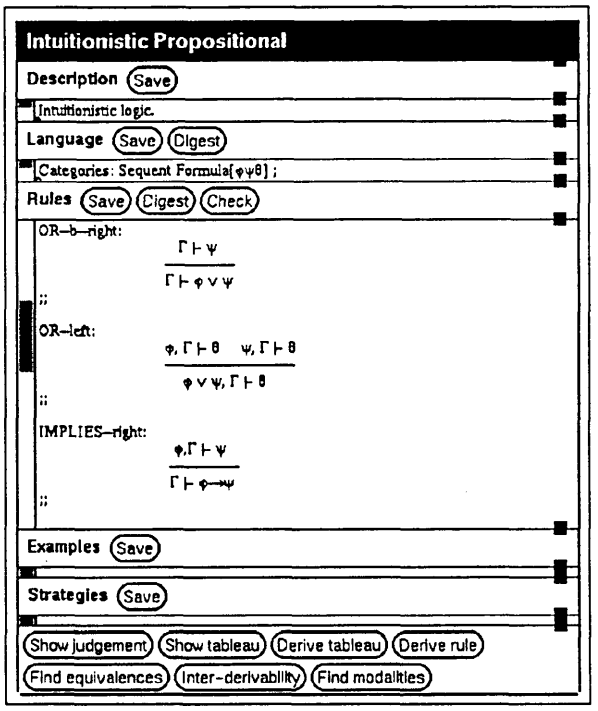


Figure 7.6: Intuitionistic Rules

7.5.1 Rule names

Rules are named using hyphen separated words. This allows rules with similar properties to be grouped together. For example, in the example system, all the rules which introduce connectives on the right-hand side of the turnstile are called *connective*-right, whereas those acting on the left are known as *connective*-left. This way of naming rules is useful when forming strategies. The names given to modal rules reflect the system in which they are applicable.

7.5.2 Side-conditions

Side-conditions can be attached to rules to restrict their applicability. This is useful when making an extension to the rule's semantics. Side-conditions can appear after the judgements as well as after the horizontal bar. They are written in the prolog metalanguage except that object language expressions must be enclosed in round brackets. Predicates intended to be used in side-conditions must be exported from the module **side**.

Rule ::= *Rule-name* :
Antecedents { *Side-condition* }
Horizontal-rule { *Side-condition* }
Consequent { *Side-condition* }
;;

Rule-name ::= *Identifier* { - *Identifier* }*

Consequent ::= *Judgement*

Antecedent ::= { *Judgement* }*

Side-condition ::= : ωp*goal* ;

Horizontal-rule ::= ----*

Figure 7.7: BNF for rules

7.5.3 Internal representation

The module **rules** for the system defines a number of predicates that represent the rules of the system. An individual rule is translated to a predicate using the following strategy. There is an additional predicate, **preds/1** which lists the names of the other predicates. The internal names of the rule predicates are derived from the names given to them by the creator of the system, and are unique between systems, as each has a fully qualified name which includes its parent module:

family/*system*/`rules`/*rule-name*/3

Each such predicate has three clauses indexed by the first parameter: RULE, PROP and META:

RULE provides a translation of the rule that is used by the prover. The second parameter is a *CIF* template for the conclusion of the rule and the third parameter is a list of *CIF* templates forming the antecedents of the rule. The body of the clause is a list of prolog predicates that must be satisfied in order to pass from the conclusion to the antecedents of the rule. The translation that constructs this clause is described in §7.5.4 below.

PROP gives a summary of the analysis of the rule. The second parameter is a list of properties:

- the arity of the rule;
- whether the rule has a subformula property;
- whether it diverges;
- what piece of syntax the rule is judged to introduce and
- if the judgement is two-sided indicates then on which side the syntax is introduced.

This clause also provides a list of the metavariables that occur in properties of the rule.

META gives the *TIF* representation of the judgements in the rule. This is used by parts of the environment that translate or manipulate rules at a "meta-level". The second parameter consists of a list of metavariables used by the rule, followed by a list of *TIF* expressions corresponding to judgements which starts with the conclusion followed by the antecedents of the rule from left-to-right. The third parameter contains a representation of side-conditions if they are present.

The following example illustrates the translation performed for the rule

$$\frac{\varphi, \psi, \Gamma \vdash \theta}{\varphi \wedge \psi, \Gamma \vdash \theta}$$

```
def AND_left ::
      RULE [SEQ,$a,θ] [[SEQ,$b,θ]] if
        permuteAssumptions [AND,φ,ψ] $a Γ &
        appendAssumption ψ Γ $c &
        appendAssumption φ $c $b
    | PROP [1,AND,left] [[φ,ψ],NODIV,SFP,LHS(E(Formula,[AND,φ,ψ]))]
    | META [[Γ,φ,ψ,θ],E(Sequent,[SEQ,C(SetOf,Formula,[E(Formula,
            [AND,MV(Formula,φ),MV(Formula,ψ)]),CMV(SetOf,Formula,Γ)]),
            MV(Formula,θ)]),E(Sequent,[SEQ,C(SetOf,Formula,[MV(Formula,φ),
            MV(Formula,ψ),CMV(SetOf,Formula,Γ)]),MV(Formula,θ)])] []
    ;
```

7.5.4 Translation from external to internal form

The META form of the rule is obtained by annotating the parse tree obtained for the rule, using the algorithm detailed in §7.4.11. Details are now provided of the translation from META to PROP which makes the presentation of the rule suitable for the theorem prover.

A PROP transformation performs the one-step extension of an object-level proof tree when the leaf judgement matches the rule's conclusion. The results of the match are the new sub-goals. The strategy used by the theorem prover, *e.g.* depth-first, breadth-first or iterative-deepening may depend on the contents of the PROP clause. For example, if *rules* denotes the names of rules to try at some point, then

```
def extendOnce ::
      rules trace goal rule trace' subGoals info if
        member rule rules & // choose a rule to use
        rule RULE goal subGoals & // apply the rule
        // adjust the tracing information
        chooseIndex subGoals index &
        adjustTrace trace rule index info trace'
    ;
```

finds the first rule that extends the *goal*, updating the *trace* information accordingly.

The overall translation is controlled by the predicate TranslateRule/3, which is a DCG accumulating the translation of the rule. TranslateRule/3 takes a *TIF* description of the rule and constructs its corresponding predicate. Most of the work for the PROP and META clauses has been done by this stage. The predicate TranslateRule/9, also a DCG, converts the *TIF* forms of the conclusion (*goal*) and antecedents (*subGoals*) of the rule into corresponding *CIF* templates (*Tgoal* and *TsubGoals* respectively). The DCG part is used to accumulate any additional predicates which are required to take apart the conclusion or to form the antecedents of the rule; it also contains any side-conditions. These predicates are bound to the variable *condition* in the following program.

```
def TranslateRule ::=
      ruleName(name,metaVars,PROP(pvs, properties),goal,
            subGoals,sideCondition) if
        [ // rule is translated into a 3-arity predicate:
          (ruleName, 3,
            [actualVars:[RULE,Tgoal,TsubGoals]:condition,
                [pvs, [PROP,numberSubGoals:name,pvs:properties]],
                [metaVars, [META,metaVars:goal:subGoals,sideCondition]]
            ]
          )
        ] &
        { TranslateRule goal Tgoal subGoals TsubGoals
```

```
                sideCondition actualVars metaVars condition [] &&
            length subGoals numberSubGoals }
    | ruleName(name,metaVars,goal,subGoals,sideCondition) if
        { Error ["unable to translate the rule for ", ruleName] }
    ;
```

TranslateRule/10 accumulates the clause's condition. It first translates the conclusion (*goal*), next the side-condition (*sideCondition*) and finally the antecedents of the rule (*subGoal*). As the condition is evaluated sequentially, this ensures that the predicate will fail if the conclusion is not of the appropriate form, or if the side-conditions are not satisfied. Only when these two guards have been satisfied are the *CIF* templates for the antecedents completed by the remainder of the condition.

```
def TranslateRule ::=
      goal Tgoal subGoals TsubGoals sideCondition actualVars metaVars language if
        { VariableNames vs } & // collection of possible variables to use
        // translate the bottom judgement in the IN sense
        Translate goal Tgoal IN vs vs' &
        TransSideCondition sideCondition & // translate the side-condition
        // translate the top judgements in the OUT sense
        TransSubGoals subGoals TsubGoals vs' vs" language &
        // determine which variables were used
        { VariablesUsed vs vs" actualVars metaVars }
    ;
```

Subgoals are translated in the 'OUT' sense by TransSubGoals/7. When constructing a derivation, negative judgements occurring as subgoals are labelled 'Neg(···)' to distinguish them from ordinary judgements. A list of the variables used is assembled by VariablesUsed/4.

```
def TransSubGoals ::=
      [] [] vs vs language
    | subGoal:subGoals TsubGoal:TsubGoals vs vs' language if
        Translate subGoal Tgoal OUT vs vs" &
        { LabelNegatedGoal subGoal Tgoal TsubGoal language } &&
        TransSubGoals subGoals TsubGoals vs" vs' language
    where {
    def LabelNegatedGoal ::
          E(category, value) goal Neg(goal) language if
            NegativeJudgementsOf language n:ns &
            member category n:ns &/
        | other goal goal language
        ;
    }
```

7.5.4.1 Judgements

The translation of a judgement is performed by Translate/7. The translation works in two senses, according to whether the judgement is being taken apart or constructed. These are referred to as IN and OUT respectively. The predicate TranslateSubGoals/6 iterates the predicate Translate/7 over the antecedents in the OUT sense.

Translation of a judgement proceeds by inspection of the structure of the *TIF* expression. The last two parameters *vs* and *vs'* are a difference list of names of available variables.

```
def Translate ::=
      E(cat, name:args) name:Targs sense vs vs' if // element
        TranslateList args Targs sense vs vs'
    | T(cat, name) name sense vs vs // terminal
    | MV(cat, name) name sense vs vs // metavariable
    | C(list, cat, elements) v sense vs vs' if // collection containing modal pattern §7.5.4.4
        { ListContainsModalPattern elements list cat } &&
        // translate the pattern using v as the communicating variable
        TranslatePattern elements v list cat sense vs vs'
    | C(list, cat, elements) value sense vs vs' if // plain collection §7.5.4.2
        TranslateCollection elements value list cat sense vs vs'
    | CMV(list, cat, name) name sense vs vs // collection metavariable
    ;
```

The translation of a simple metavariable is the metavariable itself. It acts as a placeholder and becomes a logical variable in the final clause. Similarly terminal elements stand for themselves `T(c,v)`. When an element has arguments, `E(c, name:args)`, then the arguments are translated by iterating Translate/7 over each of them in turn, TranslateList/7:

```
def TranslateList ::=
      [] [] sense vs vs
    | e:es Te:Tes sense vs vs' if
        Translate e Te sense vs vs" &
        TranslateList es Tes sense vs" vs'
    ;
```

7.5.4.2 Collections

The translation of collections and patterns is slightly different. Firstly, by using the predicate ListContainsModalPattern/3 the translation of collections is split into two cases depending on whether a pattern occurs inside the collection. If it does, TranslatePattern/9 is used (see below), otherwise TranslateCollection/9 does the work for these plain collections.

Plain collections are translated as follows. Empty collections become empty lists. Singleton collections become singleton lists unless the element is a collection metavariable; if this is the case, the name of the collection metavariable is the translation. These translations are independent of the *sense*. For more populated collections, the translation varies according to the sense.

```
def TranslateCollection ::=
      [] [] list cat sense vs vs
    | [e] Te list cat sense vs vs' if
        TranslateSingleton e Te sense vs vs'
    | e:f:fs nvs list cat IN vs vs' if
        TranslateGroupIn e f:fs nvs list cat vs vs'
    | e:f:fs nvs list cat OUT vs vs' if
        TranslateGroupOut e f:fs nvs list cat vs vs'
    ;
```

In the IN sense, predicates are added to the condition of the clause according to the type of the collection. This is done by Element/7 and an intermediate variable (*nv*) is introduced to hold the collection being taken apart.

```
def TranslateGroupIn ::=
      MV(cat, id) remainder nv list cat nv:vs vs' if
        Element IN list id rest nv &
        TranslateCollection remainder rest list cat IN vs vs'
    | T(cat, value) remainder nv list cat nv:vs vs' if
        Element IN list value rest nv &
        TranslateCollection remainder rest list cat IN vs vs'
    | E(cat, value) remainder nv list cat nv:vs vs' if
        Element IN list Te rest nv &
        Translate E(cat, value) Te IN vs vs" &
        TranslateCollection remainder rest list cat IN vs" vs'
    | CMV(list, cat, id) remainder nv list cat nv:vs vs' if
        AddCollection IN list id rest nv &
        TranslateCollection remainder rest list cat IN vs vs'
    ;
```

The predicate TranslateGroupOut/9 which acts for the opposite sense is similar. However, here the predicates in the clause's condition are formed in a slightly different order, so that the collection under construction is available when it is needed.

```
def TranslateGroupOut ::=
      MV(cat, id) remainder nv list cat nv:vs vs' if
        TranslateCollection remainder rest list cat OUT vs vs' &
        Element OUT list id rest nv
    | T(cat, value) remainder nv list cat nv:vs vs' if
        TranslateCollection remainder rest list cat OUT vs vs' &
        Element OUT list value rest nv
    | E(cat, value) remainder nv list cat nv:vs vs' if
        Translate E(cat, value) Te OUT vs vs" &
        TranslateCollection remainder rest list cat OUT vs" vs' &
        Element OUT list Te rest nv
    | CMV(list, cat, id) remainder nv list cat nv:vs vs' if
        TranslateCollection remainder rest list cat OUT vs vs' &
        AddCollection OUT list id rest nv
    ;
```

7.5.4.3 Collection management

The following predicates translate the operations of selection, injection, and union over collections SetOf, BagOf and ListOf. Predicates which finally appear in the condition are defined in the module **side**, which also contains the definitions of side-condition predicates. Other types of collection can be added easily. In addition, the data structure used to represent the collection can be changed if required.

```
def Element ::=
      OUT type element set result if
        ElementOut type element set result
    | IN type element set result if
        ElementIn type element set result
    where {
    def ElementOut ::=
          SetOf element set result if
            [ [@appendAssumption/3, element, set, result] ]
        | BagOf element set result if
            [ [@$eq/2, element:set, result] ]
        | ListOf element set result if
            [ [@$eq/2, element:set, result] ]
        ;
    def ElementIn ::=
          SetOf element result set if
            [ [@permuteAssumptions/3, element, set, result] ]
        | BagOf element result set if
            [ [@permuteAssumptions/3, element, set, result] ]
        | ListOf element result set if
            [ [@$eq/2, set, element:result] ]
        ;
    }

def AddCollection ::=
      OUT type e1 e2 result if
        AddCollectionOut type e1 e2 result
    | IN type e1 e2 result if
        AddCollectionIn type e1 e2 result
    where {
    def AddCollectionIn ::=
          SetOf set1 set2 result if
            [ [@append/3, set1, set2, result] ]
        | BagOf bag1 bag2 result if
            [ [@append/3, bag1, bag2, result] ]
        | ListOf list1 list2 result if
            [ [@append/3, list1, list2, result] ]
        ;
    def AddCollectionOut ::=
          SetOf set1 [element] result if
            [ [@addAssumption/3, element, set1, result] ]
        | SetOf set1 set2 result if
            [ [@appendAssumptions/3, set2, set1, result] ]
        | BagOf set1 set2 result if
            [ [@append/3, set2, set1, result] ]
        | ListOf set1 set2 result if
            [ [@append/3, set1, set2, result] ]
        ;
    }
```

7.5.4.4 Patterns

If a collection is found to contain a pattern, TranslatePattern/9 is used to perform the translation.

```
def TranslatePattern ::=
      elements v list cat sense vs vs' if
        { member list [SetOf, BagOf] &
          PatternConstants consts &
          ClassifyPattern elements list cat components consts consts'
        } &
        ProcessPattern sense components v list vs vs'
```

The elements in the collection are classified into three types:

1. plain collection metavariables;
2. patterns involving constructors of elements in the underlying category; and,
3. simple elements of the underlying category.

The first and last types can be translated using the predicates described in the previous section. The second type requires a modification to that strategy.

```
def Classify ::=
      CMV(list, cat, id) list cat (element, pattern, id:plain) (element, pattern, plain) if /
    | E(cat, name:args) list cat (element, P(vars, template):pattern, plain)
                (element, pattern, plain) if
        { ArgsContainCollectionMetavariable args list cat } &&
        MakeTemplate E(cat, name:args) template vars []
    | e list cat (e:element, pattern, plain) (element, pattern, plain)
    ;
```

The pattern is analysed by MakeTemplate/6. Any collection metavariables contained within it are identified and replaced with a constant in an appropriately formed template. At the same time, a table is constructed to record pairings of constants with collection metavariables. The template and table are recorded in the analysis of the pattern, P(*vars, template*) by Classify.

```
def MakeTemplate ::=
      E(cat, name:args) name:template vars vars' if
        MakeTemplateArgs args template vars vars'
    | T(cat, name) name vars vars
    | MV(cat, id) id vars vars
    | CMV(list, cat, v) const (const, v):vars vars if
        [ const ]
    where {
    def MakeTemplateArgs ::=
          [] [] vars vars
        | e:es t:ts vars vars' if
            MakeTemplate e t vars vars" &
            MakeTemplateArgs es ts vars" vars'
        ;
    }
```

Once the analysis of elements in the collection has been completed, the conditions and *CIF* templates can be generated. This is done by ProcessPattern/7 according to the sense in which the patterns occur.

IN Remove all elements from the collection. Then remove all elements matching a pattern (note that more than one pattern may be applicable for a given element). Finally, distribute the remaining elements to collection metavariables.

OUT Union the collections derived from plain collection metavariables. Add the elements. Construct new collections according to pattern templates and add these to form the final collection.

```
def ProcessPattern ::=
      IN (elements, patterns, plain) v list vs vs' if
        RemoveElements elements v v' list vs vs" &
        DestructPatterns patterns v' v" list vs" vs"' &
        PartitionPlain plain IN v" list vs"' vs'
    | OUT (elements, patterns, plain) v list vs vs' if
        PartitionPlain plain OUT v' list vs vs" &
        ConstructPatterns patterns v' v" list vs" vs"' &
        InsertElements elements v" v list vs"' vs'
    ;
```

The predicates {InsertElements, RemoveElements}, {ConstructPatterns, DestructPatterns}, and {PartitionPlain}, corresponding to translations of each of the types classified above are as follows:

```
def InsertElements ::=
      [] v v list vs vs
    | e:es v v' list nv:vs vs' if
        Translate e Te OUT vs vs" &
        Element OUT list Te v nv &
        InsertElements es nv v' list vs" vs'
    ;


def RemoveElements ::=
      [] v v list vs vs
    | e:es nv v' list nv:vs vs' if
        Element IN list Te v nv &
        Translate e Te IN vs vs" &
        RemoveElements es v v' list vs" vs'
    ;


def ConstructPatterns ::=
      [] v v list vs vs
    | P(vars,template):ps v v' list nv:vs vs' if
        Pattern OUT list v vars template nv &
        ConstructPatterns ps nv v' list vs vs'
    ;
```

```
def DestructPatterns ::=
      [] v v list vs vs
    | P(vars,template):ps nv v' list nv:vs vs' if
        Pattern IN list nv vars template v &
        DestructPatterns ps v v' list vs vs'
    ;

def Pattern ::=
     IN SetOf in vars template out if
       [ [@destructPattern/4, in, vars, template, out] ]
    | IN BagOf in vars template out if
       [ [@bagDestructPattern/4, in, vars, template, out] ]
    | OUT SetOf in [(id, var)] template out if // restriction on form of pattern in antecedents
       [ [@constructPattern/5, in, var, id, template, out] ]
    | OUT BagOf in [(id, var)] template out if
       [ [@bagConstructPattern/5, in, var, id, template, out] ]
    ;

def PartitionPlain ::=
     [] sense [] list vs vs
    | [p] sense p list vs vs
    | id:p:ps sense nv list nv:vs vs' if
       AddCollection sense list id rest nv &
       PartitionPlain p:ps sense rest list vs vs'
    ;
```

7.5.4.5 Supporting predicates

The predicates destructPattern/5 and constructPattern/4 implement the IN and OUT senses of pattern processing when a rule is applied during a derivation.

Taking patterns apart

When a pattern occurs in the conclusion, the predicate destructPattern/4 is used to filter out any occurrences of *CIF* expressions that match its template. Collections of subformulas of formulas matching the template expression are constructed.

```
.   destructPattern input formal template remainder
;
```

Where *input* is the initial collection, *formal* is a list of correspondences between collection metavariables and constants in the template. The *template* is a *CIF* expression containing constants which mark points where subformulas are to be collected. *remainder* is used to hold the *CIF* expressions which do not match the template.

The first step of this process, performed by ExtractLogicalVariables/4, analyses the list of correspondences denoted by *formal*. It assembles a list of pairs (*parts*). Each pair in the list has the form '(a, x)' where 'a' is a constant appearing in the template and 'x' is a new logical-variable. This structure allows matching *CIF* subformulas to be collected efficiently by Matches/4. Next, MatchExpressions/4 does the pattern matching. Lastly, BindCollectionMetavariables/2 collapses the collections as necessary.

```
def destructPattern ::
      input formal template remainder if
        ExtractLogicalVariables formal variables parts [] &
        MatchExpressions remainder input parts template &
        BindCollectionMetavariables formal variables


def ExtractLogicalVariables ::=
      [] []
    | (id, v):formal v':variables if
        [ (id, v') ] &
        ExtractLogicalVariables formal variables
    ;
```

MatchExpressions/4 iterates over the collection. When the collection is empty the *parts* list is terminated. Otherwise an element is checked by Matches/4 to see if it matches the *template*. If it does not it is added to the *remainder* collection.

```
def MatchExpressions ::
      [] parts template [] if
        TerminateLists parts
    | e:es parts template remainder if
        Matches template parts e parts' &
        MatchExpressions es parts' template remainder
    | e:es parts template e:remainder if
        not Matches template parts e parts &
        MatchExpressions es parts template remainder
    ;
```

The predicate Matches/4 matches the expression against the template, and if successful binds the designated subexpressions. If the template contains a non-collection metavariable, it will be bound and must remain consistently substituted throughout the match.

```
def Matches ::
      v parts e parts if
        $var v && $eq v e
    | op:formal parts op:actual parts' if
        MatchArgs formal parts actual parts'
    | id parts e parts' if
        $con id &
        replace parts (id, e:v) (id, v) parts'
    | e parts e parts
    where {
```

```
    def MatchArgs ::
          [] parts [] parts
        | a:as parts a':as' parts' if
            Matches a parts a' parts" &
            MatchArgs as parts" as' parts'
        ;
    }

def TerminateLists ::  // complete the difference lists
      []
    | (id, []):rest if
        TerminateLists rest
    ;

def BindCollectionMetavariables ::  // collapse to sets
      [] []
    | (id, v):formal v':variables if
        uniq v' v &
        BindCollectionMetavariables formal variables
    ;
```

Assembling patterns

The predicate constructPattern/5 assembles a pattern from a stream of subcomponents[2].

```
constructPattern input elements const template output
```

Add the *elements* to the initial collection *input* according to the transformation specified by *const* and *template*. Place the resulting collection in *output*.

```
def constructPattern ::
      output [] var template output
    | input e:es var template output if
        substitute template var e e' &
        addAssumption e' input output' &&
        constructPattern output' es var template output
    where {
    def substitute ::
          var var e e if /
        | op:args var e op:args' if
            / & substituteArgs args var e args'
        | f var e f
        where {
        def substituteArgs ::
              [] var e []
            | a:as var e a':as' if
                substitute a var e a' &
                substituteArgs as var e as'
            ;
        }
    }
```

[2]Patterns in antecedents involving several collection metavariables, such as 'Γ∧Δ', are not presently implemented. The reason for this is that they are open to a variety of interpretations and are not presently required.

7.5.4.6 Examples

The following examples illustrate the translation described above.

The first two example rules are from a presentation of classical logic. This is derived from an intuitionistic presentation by adding the rules *START* and *RESTART*. In this presentation, the *START* rule remembers the original sequent. Later in a derivation, the *RESTART* rule may be used to recall the original assumptions and conclusion, whilst keeping other assumptions previously made in the derivation (see §2.8.5).

$$\frac{[\Gamma \vdash \varphi], \Gamma \vdash \varphi}{\Gamma \vdash \varphi}\ START$$

The rule introduces a single element, '$[\Gamma \vdash \varphi]$', in the antecedent sequent. This insertion is done through appendAssumption/3. This predicate is chosen by Element/7 in TranslateGroupOut/9 above. Note that SFP is not present in the property list of this rule.

```
def START ::
      RULE [SEQ,Γ,φ] [[SEQ,a,φ]] if
        appendAssumption [RES,Γ,φ] Γ a
    | PROP [1,START] [[],NODIV]
    | META ···
    ;
```

$$\frac{\Gamma, \Delta \vdash \varphi}{[\Gamma \vdash \varphi], \Delta \vdash \bot}\ RESTART$$

This rule illustrates the use of a nested collection. The category of formulas in the restart system includes an element:

'[' a:SetOf(Formula) '⊢' b:Formula ']' → RES(a,b):Formula

which embeds a sequent in the formula category. The Γ, Δ in the construction of the antecedent is performed by appendAssumptions/3.

```
def RESTART ::
      RULE [SEQ,a,BOT] [[SEQ,b,φ]] if
        permuteAssumptions [RES,Γ,φ] a Δ &
        addAssumption [RES,Γ,φ] Δ c &
        appendAssumptions c Γ b
    | PROP [1,RESTART] [[],NODIV,SFP]
    | META ···
    ;
```

The next two rules contain modal patterns.

$$\frac{\Gamma \vdash \varphi, \Delta}{\Box\Gamma, \Gamma' \vdash \Box\varphi, \Diamond\Delta, \Delta'}\ \textit{BOX-K-D-T}$$

The patterns '$\Box\Gamma$' and '$\Diamond\Delta$' are taken apart by **destructPattern/4**. Consider the first occurrence of this predicate. Its second parameter is a list, '`[(a,Γ)]`', which uses placeholders in the template to represent any collection metavariables that occur in the pattern. Here, there is a single metavariable, Γ, associated with the placeholder selector '`a`'. The template is '`[BOX,a]`'. Elements of the collection occurring in the conclusion are passed into the predicate by the logical variable *a*. The predicate removes elements that match the pattern from *a* and places them, without their containing pattern, in the collection Γ. Elements that fail to match the pattern are bound to the fourth parameter, Γ′. (Since Γ′ does not occur in any antecedents of the rule, this collection is discarded.) The second occurrence of **destructPattern** is much the same as the first but for the fact that **permuteAssumptions** removes the element '$\Box\varphi$' from *b* before passing the collection, now called *c*, to **destructPattern**. The antecedent of the rule is formed by adding the element 'φ' to the collections of subformulas given by Δ to form *d*, the right-hand side of the sequent.

```
def BOX_K_D_T ::
      RULE [SEQ,a,b] [[SEQ,Γ,d]] if
        destructPattern a [(a,Γ)] [BOX,a] Γ' &
        permuteAssumptions [BOX,φ] b c &
        destructPattern c [(a,Δ)] [DIA,a] Δ' &
        appendAssumption φ Δ d
    | PROP [1,BOX,K,D,T] [[Γ,Δ,φ],NODIV,SFP,...]
    | META ···
    ;
```

$$\frac{\Box\Gamma \vdash \varphi, \Diamond\Delta}{\Box\Gamma, \Gamma' \vdash \Box\varphi, \Diamond\Delta, \Delta'}\ \textit{BOX-S4}$$

The rule *BOX-S4* is contains examples of patterns in its antecedent sequent. The predicate **constructPattern/5** is used to form collections containing patterns occurring in antecedents.

The second occurrence of **constructPattern/5** constructs the right-hand collection '$\varphi, \Diamond\Delta$'. The first parameter of this predicate holds the contents of the collection before construction using the pattern. This will be the singleton '`[φ]`'. Subsequent parameters of the predicate are: the merging collection, 'Δ'; followed by its placeholder '`a`'; the template, '`[DIA,a]`', which gives the effect of '$\Diamond\Delta$'; and the collection formed by adding the pattern, *d*.

```
def BOX_S4 ::
      RULE [SEQ,a,b] [[SEQ,d,f]] if
        destructPattern a [(a,Γ)] [BOX,a] Γ' &
        permuteAssumptions [BOX,φ] b c &
        destructPattern c [(a,Δ)] [DIA,a] Δ' &
        constructPattern [] Γ a [BOX,a] d &
        constructPattern [] Δ a [DIA,a] e &
        appendAssumption φ e f
    | PROP [1,BOX,S4] [[Γ,Δ,φ],NODIV,SFP, ...]
    | META ···
    ;
```

The next example illustrates the translation of a rule involving a parameterised modal operator.

$$\frac{\Gamma \vdash \varphi}{[i]\Gamma, \Gamma' \vdash [i]\varphi, \Delta}$$

where '$[i]\varphi$' (or '$[i]\Gamma$') means that the individual 'i' believes or knows some proposition 'φ' (or collection of beliefs 'Γ').

```
def Believes ::
      RULE [SEQ,a,b] [[SEQ,Γ,[φ]]] if
        destructPattern a [(a,Γ)] [BELIEVES,i,a] Γ'] &
        permuteAssumptions [BELIEVES,i,φ] b Δ
    | PROP [1,Believes] [[Γ,i,φ],NODIV,SFP,
          MODAL(E(Formula,[BELIEVES,i,Γ])),
          RHS(E(Formula,[BELIEVES,i,φ]))]
    | META ···
    ;
```

The choice of 'i' is non-deterministic but once chosen, must be consistent throughout the application of the rule.

The following example illustrates the translation of a rule that uses a negative judgement. The rule introduces '$\downarrow$' on the left hand side of a sequent (§3.7).

$$\frac{\Gamma \nvdash \psi, \Delta \quad \Gamma, \varphi \vdash \psi, \Delta}{\Gamma, \varphi \downarrow \psi \vdash \Delta}$$

This is translated to the internal form:

```
def UNLESS_left ::
      RULE [SEQ,a,Δ] [Neg([NSEQ,Γ,b]),[SEQ,c,d]] if
        append Γ [[UNLESS,φ, ψ]] a &
        appendAssumption ψ Δ b &
        appendAssumption φ Γ c &
        appendAssumptions ψ Δ d
    | PROP [2,UNLESS,left] [[ψ,φ],NODIV,LHS(E(Formula, [UNLESS,φ,ψ]))]
    | META ···
    ;
```

The negated judgement '$\Gamma \nvdash \psi, \Delta$' is labelled by enclosing it with '`Neg(···)`'. This is enough to distinguish it from the positive judgements when constructing a derivation.

7.5.5 Side-conditions

Side-conditions are sometimes required to make special syntactic constraints on the applicability of a rule. A side condition is written to the right of the horizontal bar of a rule. The hypersequent rules shown in figure §3.28 provide an example of a side-condition

$$\frac{[\varphi, \Box\varphi, \Gamma \vdash \Delta; \Sigma']}{[\Box\varphi, \Gamma \vdash \Delta; \Sigma]} : \texttt{hyper } (\Box\varphi)\ \Sigma\ \Sigma'\ ;\quad S5\text{-}BOX$$

The parentheses are required around complex terms. A side-condition is translated to a predicate of the appropriate arity in the module side. The translation for the example shown is "`@hyper/3 [BOX,`φ`]` Σ Σ'". Complex terms are replaced by their corresponding *CIF* expressions.

7.6 Heuristics

The heuristic language must be introduced before discussing the way in which derivations of judgements are constructed. The heuristic language guides the selection of rules in the proof procedure by breaking the derivation into distinct regions.

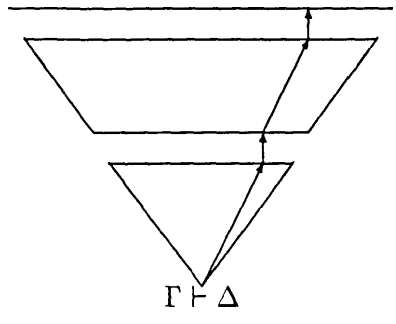


Figure 7.8: Derivation Tree

In the figure above, the proof is broken into horizontal regions. The structure of the path shown through the derivation is specified by the heuristic. The horizontal regions correspond to *segments* of the heuristic.

There are three different types of segment: eager '[···]', lazy '<···>', and once '(···)'. An eager segment grows to be as large as possible, whereas lazy segments grow only when necessary. A segment of type 'once' produces a region containing a single application of a rule. Each segment contains rule specifiers that select some of the rules in the system. The selected rules are used to grow the segment.

Figure 7.9 gives the syntax of heuristics. The operations on *Rule-specifiers* are: '&' (intersection), ',' (union), '-' (difference). The operator '*' stands for all currently selectable rules. Integers are used to select rules with a particular degree of branching, and round brackets can be used to construct complex selections.

7.6.1 Example heuristics

```
<left,right>[BASIC]
<1&(left,right),left,right>[BASIC]
{d=20,(not,and)=3}<left,right>[BASIC]
```

```
Heuristic        ::=  { Global-constraint } { Segment }+

Segment          ::=  < Rule-specifiers >
                    |  ( Rule-specifiers )
                    |  [ Rule-specifiers ]

Rule-specifiers  ::=  Rule-subname
                    |  integer
                    |  *
                    |  ( Rule-specifiers )
                    |  - Rule-specifiers
                    |  Rule-specifiers & Rule-specifiers
                    |  Rule-specifiers , Rule-specifiers
                    |  Rule-specifiers - Rule-specifiers

Global-constraint ::=  { Condition { , Condition }* }

Condition        ::=  d = integer
                    |  b = integer
                    |  Rule-specifiers = integer
```

Figure 7.9: BNF for heuristics

Explanation

In the first example, the eager segment '`[BASIC]`' forces the end of every branch to be an application of a *basic* rule. Other rules used on the branch derive from the lazy segment '`<left,right>`'. The growth of a proof using this heuristic is governed by the test-generate cycle which is produced by the interaction of the lazy and eager segments. The lazy first segment is subordinate to the eager second segment.

The second example is similar to the first except that amongst rules with the names *left* and *right*, those with a single sub-goal are considered first. This reduces the branching in the proof. It is worth remarking that a rule is only considered once in any single use of a segment.

The third example is again similar to the first, except that it uses a global constraint, containing counters to limit the resources used. Proofs are restricted to a depth of 20 inferences (`d=20`), and the number of rules concerned with `and` and `not` is limited to a maximum of 3 in each branch. The number of rules not possessing the subformula property can be restricted by setting the `b` counter. Global constraints are useful for analysing wayward rule sets.

Figure 3.6 on page 79 illustrates some modal heuristics. The heuristics used for the restart example of §2.8.5 on page 55 provide a good illustration of an initial 'once' region for the *start* rule:

```
(start)<left,right,restart>[BASIC]
```

7.6.2 Expansion of a heuristic

Heuristics must be expanded, using ExpandHeuristic/3, before they are suitable for use in constructing derivations. The process of expansion converts the contents of each segment in the heuristic to the ordered list of rules that the segment designates.

```
def ExpandHeuristic ::
      paths logic paths' if
        RuleNamesOf logic possibleRules &
        ExpandHeuristic' paths possibleRules paths' &&
        UnparseExpandedHeuristic paths' toks []
    where {
    // Expand each try list into a list of rules that satisfy the conditions
    def ExpandHeuristic' ::
          [] possRules []
        | GLOBAL(info):others possRules GLOBAL(info'):others' if
            ExpandGlobal info possRules info' &&
            ExpandHeuristic' others possRules others'
        | effort(ruleDesc):others possRules effort(rules):others' if
            ExpandRuleDesc ruleDesc possRules rules &
            CheckExpansion rules &
            ExpandHeuristic' others possRules others'
        ;
```

A check is made to remind the user if the expansion of a segment yields no matching rules.

```
    def CheckExpansion ::
          [] if
            Error ["One of the segments is empty"]
        | rules
        ;
    }
```

Expansion of counters in the global declaration is performed by ExpandGlobal/3

```
    def ExpandGlobal ::
          [] possRules []
        | Counter(desc,n):rest possRules Counter(rules,n):rest' iff
            ExpandRuleDesc desc possRules rules &
            CheckExpansion rules &&
            ExpandGlobal rest possRules rest'
        | other:rest possRules other:rest' if
            ExpandGlobal rest possRules rest'
        ;
```

The abstract syntax of the rule specification is used to construct an ordered list of matching rules as follows:

```
def ExpandRuleDesc ::
      All possible possible
    | Pattern(p) possible matching if
        iter possible @matchesPattern/3 [p] matching rejected
    | RuleName(r) possible matching if
        iter possible @member/3 [r] matching rejected
    | SubGoals(n) possible matching if
        iter possible @member/3 [n] matching rejected
    | NOT(p) possible negated if
        ExpandRuleDesc p possible matching &
        filter matching possible negated
    | LESS(p1,p2) possible negated if
        ExpandRuleDesc p1 possible matching &
        ExpandRuleDesc p2 possible matching' &
        filter matching' matching negated // negated = matching − matching'
    | OR(p1,p2) possible disjunct if
        ExpandRuleDesc p1 possible matching &
        ExpandRuleDesc p2 possible matching' &
        append matching matching' disjunct' &
        uniq disjunct' disjunct
    | AND(p1,p2) possible conjunct if
        ExpandRuleDesc p1 possible matching &
        ExpandRuleDesc p2 matching conjunct
    ;
```

matchesPattern/3 and member/3 are the predicates that are given to iter/5.

```
def matchesPattern ::
      r types p if
        member t types &
        re p t
    ;

def member ::
      r types p if
        member p types
    ;
```

iter/5 partitions the list of rules according to the success or failure of its predicate.

```
def iter ::
      [] pred args [] []
    | rule:rules pred args rule:sat nsat if
        RuleName rule types &
        META pred rule:types:args &
        iter rules pred args sat nsat
    | rule:rules pred args sat rule:nsat if
        iter rules pred args sat nsat
    ;
```

7.6.2.1 Example

The heuristic <left,right>[BASIC], used in the context of the Intuitionistic Propositional system, is first parsed to give a list of segments from its underlying *tokens*.

```
ParseHeuristic [LAZY([[left],[right]]),EAGER([[BASIC]])] tokens []
```

The segments are next expanded according to the above rules giving the following:

```
ExpandHeuristic [LAZY([[left],[right]]),EAGER([[BASIC]])] logic
    [
     LAZY([in/prop/AND_left/3, in/prop/OR_left/3, ...,
       in/prop/AND_right/3, in/prop/OR_right/3, ...]),
     BASIC([in/prop/BASIC/3])
    ]
```

7.7 Constructing Derivations

This section describes the procedure used to construct derivations with heuristics. The procedure uses the heuristic to constrain branches of the derivation tree, dividing them into layers which correspond with the segments of the heuristic. This correspondence means that a region contains only applications of rules allowed by the heuristic segment. Branches of the derivation are also forced by the derivation procedure to be irredundant. A redundant branch is one containing a judgement that can be subsumed by a judgement below it in the derivation. Subsumption means that there is some substitution, which if it is applied to the lower judgement, makes it the same as the higher judgement. The predicate SawGoal/5 ensures that branches are irredundant.

The interface between the prover module and the remainder of the environment is through the predicate HeuristicSatisfiesJudgement/5 which takes five parameters as follows:

```
HeuristicSatisfiesJudgement heuristic judgement premises proof language
```

The initial judgement is treated as the only leaf in an existing proof; the predicate extendTip/7 takes this leaf and attempts to find a derivation that satisfies the heuristic. The derivation constructed has the form:

derivation ::= *Premise(*premise*)*
| rule*(*conclusion,antecedents*)*

where *premise* and *conclusion* are judgements, and *antecedents* are derivations corresponding to the rule's antecedents.

```
def HeuristicSatisfiesJudgement ::
      heuristic judgement premises proof language if
        InitiallySeen seen & // initialise seen to []
        InitialTrace trace heuristic heuristic' &
        extendTip judgement heuristic' [] seen premises proof language
    .;
```

The predicate below initialises the *trace* with counters which record the resources available for use in the branch. The predicate select/4 either extracts the user's definition or sets a default value.

```
def InitialTrace ::
      [rule(depth:baddies:counters)] heuristic heuristic' if
        select GLOBAL(info) GLOBAL([]) heuristic heuristic' &
        select Depth(depth) Depth(40) info info' &
        select Baddies(baddies) Baddies(10) info' counters &/
    where {
```

```
def select ::
      pattern default in out if
        remove pattern in out &/
    | default default list list
    ;
}
```

The predicate extendTip/7 extends a leaf of the derivation using the heuristic as a guide. This terminates if the judgement is subsumed by a premise, but in most cases, no premises are provided initially. In this case, the first segment of the heuristic is used to extend the leaf. If the goal is negated, then the success of the derivation is inverted. No proof is recorded for negative judgements. The inversion is done by *not i.e.* negation by failure.

```
def extendTip ::
      judgement heuristic trace seen p:ps Premise(judgement, []) language if
        member premise p:ps &
        EqualityOf language premise judgement
    | Neg(judgement) hunger(rules):heuristic trace seen premises
          Negated(judgement, []) language if
        not extendUsingBlock hunger rules heuristic trace
                  judgement seen premises proof language
    | judgement:args hunger(rules):heuristic trace seen premises
          proof language if
        extendUsingBlock hunger rules heuristic trace
                  judgement:args seen premises proof language
    ;
```

What happens next depends on the "hunger" of the segment. There are three cases to consider according to the type of hunger. The predicate extendOnce/7 is used to apply a single rule to the leaf:

ONCE An occurrence of a rule in this segment must appear at this point in the proof. Use extendOnce to insert the rule. If successful, recursively extend each of the subgoals. If no extension is possible then *fail.*

EAGER For an eager segment, try to extend the current leaf. If this is successful, then use the same segment to extend subsequent leaves. Otherwise, continue with the next segment.

LAZY Defer the present lazy segment until no extension of the present leaf using the next and subsequent segments is possible.

```
def extendUsingSegment ::
      ONCE rules heuristic trace goal seen premises rule(goal,proofs) language if
        extendOnce rules trace goal rule trace' subGoals (goal,seen) &
        SawGoal subGoals goal seen seen' language &
        // carry on with the rest of the heuristic
```

```
        extendConj subGoals heuristic trace' seen' premises proofs language
    | EAGER rules heuristic trace goal seen premises rule(goal,proofs) language if
        extendOnce rules trace goal rule trace' subGoals (goal,seen) &
        SawGoal subGoals goal seen seen' language &
        // carry on being eager
        extendConj subGoals EAGER(rules):heuristic trace' seen' premises proofs language
    | EAGER rules heuristic trace goal seen premises proof language if
        extendTip goal heuristic trace seen premises proof language
    | LAZY rules hunger(rules'):heuristic trace goal seen premises proof language if
        extendUsingSegment hunger rules' heuristic trace goal seen premises proof language
    | LAZY rules heuristic trace goal seen premises rule(goal, proofs) language if
        extendOnce rules trace goal rule trace' subGoals (goal,seen) &
        SawGoal subGoals goal seen seen' language &
        // carry on being lazy once more
        extendConj subGoals LAZY(rules):heuristic trace' seen' premises proofs language
    ;
```

Extend a conjunction of subgoals by extending them separately using the same heuristic. Adjust the trace to record the number of conjuncts remaining and the depth of the search; this is done for control and diagnostic purposes.

```
def extendConj ::
      // conjunction is satisfied
      [] heuristic trace seen premises [] language if /
    | goal:goals heuristic trace seen premises proof:proof' language if
        extendTip goal heuristic trace seen premises proof language &
        changeTrace trace trace' &
        extendConj goals heuristic trace' seen premises proof' language
    where {
    def changeTrace ::
          rule(depth,index):trace rule(depth,index'):trace if
            succNumber index' index
        ;
    }
```

Perform a one-step extension of the proof. Find all possible extensions of the goal by enumerating them using backtracking. The successful application of a rule is conditional on satisfying the global conditions. Global conditions are represented in each part of the trace by a vector of counters. The first two elements correspond to d and b, while any remaining elements are indexed by the counters they represent. The trace takes the form of a list

$$rule(depth:badness:counters):rule'(\cdots):\cdots$$

where the *rule* is the current rule application and *rule'* is the previous one, *etc.*

```
def extendOnce ::
      rules trace goal rule trace' subGoals info if
        member rule rules &
        RuleDefinition rule goal subGoals &
        adjustTrace trace rule trace'
```

```
where {
def GoodRule ::
      rule if
        RuleProperties rule properties &
        member NODIV properties &
        member SFP properties & /
    ;
def adjustTrace ::
    rule(depth:bad:counters):trace rule
            r(depth':bad':counters'):rule(depth:bad:counters):trace if
        prevNumber depth "d" depth' & // limit length of a branch
        adjustBadness bad rule bad' && // limit number of bad rules on a branch
        adjustCounters counters rule counters'
    ;
def adjustBadness ::
      bad rule bad if
        GoodRule rule
    | bad rule bad' if
        prevNumber bad "b" bad'
    ;
def adjustCounters ::
      [] rule []
    | Counter(rules,n):counters rule Counter(rules,n'):counters' if
        member rule rules &&
        prevNumber n "c" n' &
        adjustCounters counters rule counters'
    | other:counters rule other:counters' if
        adjustCounters counters rule counters'
    ;
}
```

Perform loop checking. A goal has been seen before if it occurs on the present path. Goals are compared using the equality predicate defined by the language of the system.

```
def SawGoal ::
    [] goal seen goal:seen language
  | subgoal:subgoals goal seen seen' language if
      not HaveSeen subgoal:subgoals goal:seen language
  where {
  def HaveSeen ::
      subgoal:rest seen language if
        SeenGoal subgoal seen language
    | subgoal:rest seen language if
        HaveSeen rest seen language
    ;
  def SeenGoal ::
      goal previous:rest language if
        EqualityOf language goal previous
    | goal previous:rest language if
        SeenGoal goal rest language
    ;
  }
```

Remarks

The procedure as a whole is effective, as strategies are easily interpreted. Further, using the internal form of rules, it is possible to decide quickly whether a rule is applicable. In this way, the interpretation overheads associated with the use of rules are kept down.

A number of optimisations were tried in an attempt to improve the effectiveness of the search procedure. In particular, failed judgements were memorised and used to prevent replicated searching, but the overheads of storing and checking these judgements outweighed the benefits.

Chapter 8

Conclusions & Future Work

The framework described in this thesis differs from other systems such as Edinburgh's Logical Framework, which is based on the propositions-as-types paradigm and the typed lambda calculus. Our motivation was less theoretical, and, as our early ideas were formed in the context of the GENESIS project[1], slanted towards the development of a usable interface.

The linguistic framework has nevertheless proved to be capable of supporting a wide variety of logical systems in the all-introduction style. Although such presentations can reasonably be expected to support logics with truth-functional connectives, they are surprisingly versatile at supporting non-truth functional systems such as Intuitionistic logic, Modal logics, and so on. However, it should be noted that not all systems can be presented in this way (*e.g.* when arbitrary axioms are present), and some of the presentations are more natural than others.

There are still many avenues left open for exploration:

- Quantification is still relatively poorly supported by the environment, further work is required to provide interactive proof construction (rather than just a strategy driven one). An idea here is to explore the close relationship between sequent-style presentations and tableaux ones. Tableaux presentations are useful as they tend to minimise the clutter related with large sequents. But, it is not clear how to relate all forms of rules that are the possible in the current framework into a suitable generalisation of a tableau setting.

  Some of the basic graphical user interface structures needed to support the use of tableaux have already been devised, and a prototype tableau widget has been developed. But more work is required.

[1]The author was employed as a research assistant on the GENESIS project, ESPRIT 1222(1041), from April 1986 to July 1989.

- The formulation of new systems continues to provide insights into the development of the environment. In particular, some are discussed in Chapter 3, namely modal patterns and negative judgements. These have proved very useful. A pragmatic approach is adopted to deal with the introduction of new facilities into the description of language and rules of a system. The objective being to minimise the use of side-conditions. Several research students in the department have used to environment to clarify their ideas about logics of Commitment and Dialogue, *e.g.* [FRS89].

Bibliography

[Abr87] S. Abramsky. Domain theory in logical form. In *Symposium on Logic in Computer Science*, pages 47–53. IEEE Computer Society, 1987.

[Avr87] A. Avron. Simple consequence relations. Technical Report ECS-LFCS-87-30, LFCS, Department of Computer Science, University of Edinburgh, 1987.

[Avr88] A. Avron. Foundations and proof theory of 3-valued logics. Technical report, LFCS, Department of Computer Science, University of Edinburgh, April 1988.

[C+86] R. L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, 1986.

[Car87] W. A. Carnielli. Systematization of finite many-valued logics through the method of tableaux. *Journal of Symbolic Logic*, 52(2):473–493, 1987.

[CH85] T. Coquand and G. Huet. Constructions: A higher order proof system for mechanizing mathematics. In *Proceedings of EUROCAL '85*, Linz, Austria, 1985.

[Che86] J. H. Cheng. *A Logic of Partial Functions.* PhD thesis, Department of Computing, University of Manchester, 1986.

[CKPR73] A. Colmerauer, H. Kanoui, R. Pasero, and P. Roussel. Un systeme de comunication homme-machine en francais. Technical report, Groupe Intelligence Artificielle, Universite d'Aix Marseille, Luminy, 1973.

[Com85] Apple Computer. *Inside Macintosh. Volumes I, II and III.* Addison Wesley, 1985.

[Cur77] H. B. Curry. *Foundations of Mathematical Logic.* Dover Edition, 1977.

[dB80] N. de Bruijn. A survey of the project AUTOMATH. In Seldin and Hindley [SH80], pages 579–606.

[DM82] L. Damas and R. Milner. Principle type-schemes for functional programs. In *POPL*, pages 207–212, 1982.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, March 1960.

[DP89] M. Dawson and J.-S. Pendry. ω-prolog reference manual. Technical report, Imperial College, University of London, 1989.

[dQ88] R. J. G. B. de Querioz. A proof-theoretic account of programming and the rôle of reduction rules. *Dialectica*, 42(4), 1988.

[Dum77] M. Dummett. *Elements of Intuitionism.* Clarendon Press, Oxford, 1977.

[Fit83] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics.* D. Reidel Publishing Company, 1983.

[Fre67] G. Frege. Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought. In van Heijenoort [vH67], pages 1–82.

[FRS89] H. Fuks, M. Ryan, and M. Sadler. Outline of a commitment logic for legal reasoning. In *Proc 3rd International Conference on Logics, Informatics and Law; Florence*, November 1989.

[Gab89] D. M. Gabbay. LDS - labelled deductive systems. Technical report, Department of Computing, Imperial College, November 1989.

[Gab91] D. M. Gabbay. Elements of algorithmic proof. In *Handbook of Logic in Computer Science*, volume 1. Oxford University Press, 1991.

[Gal87] J. H. Gallier. *Logic for Computer Science: foundations of automatic theorem proving.* John Wiley & Sons, 1987.

[Gen69] G. Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, chapter 3, pages 68–129. North-Holland Publishing Company, 1969.

[GMW79] M. J. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF: a mechanised logic of computation.* Springer-Verlag, 1979.

[GN87] M. Genesereth and N. Nilsson. *Logical Foundations of Artifical Intelligence.* Morgan Kaufmann Publishers, Inc, 1987.

[Gol82] R. Goldblatt. *Axiomatising the Logic of Computer Programming*, volume 130 of *Lecture Notes in Computer Science.* Springer-Verlag, 1982.

[Gol87] R. Goldblatt. *Logics of Time and Computation.* CLSI Lecture Notes Number 7, 1987.

[GSN89] J. Gettys, R. W. Scheifler, and R. Newman. Xlib - C language X interface. Technical report, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1989.

[Haa78] S. Haack. *Philosophy of Logics.* Cambridge University Press, 1978.

[Har85] W. Harwood. The Genesis project: Technical annex. Technical report, Imperial Software Technology Ltd., 1985.

[HC84] G. Hughes and M. Creswell. *A Companion to Modal Logic.* Methuen, 1984.

[Hil67] D. Hilbert. Foundations of mathematics. In van Heijenoort [vH67], pages 464–479.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the A.C.M.*, 21(8):666–77, 1978.

[Hoa81] C. A. R. Hoare. A calculus of total correctness for communicating processes. *Science of Computer Programming*, 1:49–72, 1981.

[Hoa89] C. A. R. Hoare. An axiomatic basis for computer programming. In C. B. Jones, editor, *Essays in Computing Science.* Prentice Hall International, 1989.

[How80] W. Howard. The formulae-as-types notion of construction. In Seldin and Hindley [SH80], pages 479–490.

[Jon86] C. B. Jones. *Systematic software development using VDM.* Prentice-Hall, 1986.

[Jut77] L. Jutting. *Checking Landau's Grundlagen in the AUTOMATH system.* PhD thesis, Technische Hogeschool, Eindhoven, 1977.

[Kle52] S. C. Kleene. *Introduction to Metamathematics.* North-Holland Publishing Company, 1952.

[KLM90] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artifical Intelligence*, 44:167–207, 1990.

[Kow74] R. Kowalski. Predicate logic as a programming language. In *IFIP*, pages 569–574, 1974.

[Kri59] S. Kripke. A completeness theorem in modal logic. *Journal of Symbolic Logic*, 24:1–14, 1959.

[Mar72] P. Martin-Löf. An intuitionistic theory of types: predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73*. North-Holland, 1972.

[MAS89] J. McCormack, P. Asente, and R. R. Swick. X toolkit intrinsics. Technical report, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1989.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

[Pau88] L. C. Paulson. *Logic and Computation - Interactive Proof with Cambridge LCF*. Cambridge University Press, 1988.

[Pau89] L. C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5:363–397, 1989.

[Pet82] K. Petersson. A programming system for type theory. Technical report, Programming Methodology Group, Department of Computing Sciences, University of Göteborg, Chalmers University of Technology, Sweden, 1982.

[Pet89] C. D. Peterson. Athena widget set - C language reference. Technical report, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1989.

[PHV60] D. Prawitz, Hakan, and Vogera. A mechanical proof procedure and its realisation on a computer. *Journal of the ACM*, 7:102–28, 1960.

[Pot83] G Pottinger. Uniform, cut-free presentations of T, S4 and S5 (abstract). *Journal of Symbolic Logic*, 48, 1983.

[Pra65] D. Prawitz. *Natural Deduction: A Proof-Theoretical Study*. Almqvist & Wiksell, 1965.

[Pra80] V. Pratt. Applications of modal logic to programming. *Studia Logica*, 39:257–274, 1980.

[Rob65] J. A. Robinson. A machine oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[Sad84] M. R. Sadler. Mapping out specification. Technical report, Imperial College, London, 1984.

[Sco82] D. Scott. Domains for denotational semantics. In *ICALP, Lecture Notes in Computer Science 140*, pages 577–613. Springer-Verlag, 1982.

[SH80] J. Seldin and J. Hindley, editors. *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, 1980.

[Tur84] R. Turner. *Logics for Artifical Intelligence.* Ellis Horwood Ltd, 1984.

[vH67] J. van Heijenoort, editor. *From Frege to Gödel: A Source Book in Mathematical Logic. 1879–1931.* Harvard University Press, 1967.

[W+84] L. Wos et al. *Automated reasoning: introduction and applications.* Prentice-Hall, 1984.

[Wra87] M. Wray. The genesis interface. Technical report, Genesis Project, 1987.