Verification in ASL
and
related specification languages

Jorge Farrés-Casals

Ph.D.

University of Edinburgh

1992

Abstract

In recent years a new framework for specification has been defined around *ASL* [SW 83, Wir 86, ST 88a, ST 88b]. Stress has been put on defining a specification language consisting of a few powerful specification building operations (SBO's for short) with simple semantics and an elegant implementation notion. Some important features of this work are the generalization to an arbitrary institution [GB 84] of a lot of previous work done on algebraic specification and the study of behavioural abstraction in the context of a model-oriented specification language.

This basic research on formal specifications is generally regarded as the basis for a new generation of specification languages. These specification languages will instantiate *ASL* with their institution of interest, and will define their own specification constructs and implementation notion in terms of *ASL*'s primitive SBO's and implementation notion.

However, any useful formal framework for program development needs an inference system for the implementation relation, *i.e.* proofs that one specification implements another must be produced by a fixed family of rules without model-theoretical considerations. This poses a new and difficult problem to *ASL* due to its model-oriented nature and the great generality of both the implementation relation and the SBO's.

In this thesis we study this problem starting from a simple specification language with only three SBO's, and progressively adding other common SBO's. In the course of this analysis we encounter four main problems for the verification of implementations: hiding of auxiliary functions, behavioural abstraction, reachability constraints and parameterization. These problems can be considered classical of algebraic specifications and the study of their verification aspects at an institution-independent level provides valuable results for many other specification languages.

New results for the verification of implementations w.r.t. specifications with hidden parts and abstracted specifications at an institution-independent level are the main contribution of the thesis. Verification of reachability constraints is shown to be below the institutional level. In this case, a common institution for constraints is formally presented showing some ignored verification aspects. Finally, an original presentation of parameterization and structured implementations concludes the thesis.

In conclusion, this thesis presents a collection of sublanguages, inference systems and side conditions which add a new dimension to the fascinating job started by *ASL* in [SW 83].

Statement

The work described in this thesis is my own and the thesis was composed by myself. Parts of chapters 3 and 4 were previously published in [Far 89] and [Far 90], respectively.

Acknowledgements

I would like to thank my supervisor Don Sannella for introducing me to the topic of formal specifications and helping me closely along my research, as well as for being so encouraging and patient with me.

Many other people helped me during my stay in Edinburgh. Professor Rod Burstall acted as my second supervisor and encouraged me in the difficult beginning. Many others, particularly among the PhD students, helped to improve my understanding in many areas of theoretical computer science and became my friends, among them I like to mention Eugenio Moggi, Bernhard Steffen, Luo, Paweł, Claudio, Andreas, Bill, Ed, Barry, Michael, etc.

I would also like to mention, despite their short stay in Edinburgh, Andrzej Tarlecki and Terry Stroup for their good insight in many problems, and Mauro and Gloria for their wonderful enthusiasm.

Home in Barcelona, I must thank my old professors for teaching me well and for their personal involvement in helping me to come to Edinburgh, in particular I have to mention Professor Fernando Orejas who kept his interest in my progress for four years.

Finally, it is a pleasure to mention the very good group of friends which made of Edinburgh a warm city.

My PhD study was initially funded by *La Caixa* and during the last two years by the *Ministerio de Educación y Ciencia.*

Table of Contents

Chapter 1

Introduction

1.1 Specifying software systems

Generally a specification is a description of a system establishing some requirements that the engineer must fulfil in the implementation of the system. Alternatively, a specification can be a document describing the properties of a system to the users.

Similarly, in computer science, specifications appear as descriptions of tasks that programs should achieve and as descriptions of programs for the users. Ideally, these descriptions must capture all the requirements/properties which need to be fulfilled/known and only those. In practice, these two kinds of specifications are confused in only one, and we talk about specifications of program behaviours which happen to be all that a user needs to know and all that the programmer needs to fulfil.

A common approach to the specification of software systems considers a system to be a black box and its specification to be a description of the relation between the input and the output of the system. These specifications are known as *algebraic specifications*.

Algebraic specifications

Algebraic specifications are called *algebraic* because they only refer to the functionality of the system and can denote, at most, an algebra defining the input/output relation of a system.

When the output is uniquely defined for each input, an algebraic specification denotes an algebra. Otherwise, an algebraic specification denotes a class of algebras, *i.e.* all the algebras which satisfy the input/output requirements in the specification.

This approach simplifies specifications greatly; perhaps too much, since it ignores possible requirements on time, limited resources, non-determinism, synchronization, etc. Nevertheless, the problem of correctness, *i.e.* the production of results according to the requirements, is conveniently isolated in the algebraic framework.

There exist a large variety of algebraic specification languages. They differ in the logics used to express the input/output requirements, in the operations used to structure the specification and in the semantics.

Very popular among specification languages are those using equations to express requirements, parameterization to structure specifications and *initial semantics* to interpret them (see [GTW 76, EM 85]). We shall say that these languages follow the **initial approach** to specification.

Alongside the initial approach there are many other alternative approaches. Some of them, like the initial approach, use equations and define each specification to denote an isomorphic class of algebras, for example the final approach [Wan 79, Kam 83] and the initial-behavioural approach [GGM 76, NO 88].

Other specification languages have a **loose** semantics allowing a specification to denote a class of non-isomorphic algebras. Among these CLEAR [BG 77, BG 80] and LARCH [GHW 85] use semantics based on theories, whereas *ASL*-related specification languages allow specifications to denote arbitrary classes of algebras [SW 83, Wir 86, ST 88a] or structures [WB 89].

Finally, common *non-algebraic* specification languages such as Z [Spi 85] and VDM [Jon 80] use a model to specify a software system. These languages are not considered algebraic because they present a mathematical model for the system instead of some requirements on the relationship between the input and the output of the system. Nevertheless, semantics as those given to Z in [Spi 85] consider that an abstract model specification denotes only the algebra implicitly described through the model. In this sense, these specification languages are also algebraic. The basic difference remaining between abstract model specifications such as Z and common algebraic specifications is the way in which requirements are expressed, that is using some predefined mathematical models or using some axioms and a satisfaction relation between axioms and models respectively.

The reasons for this proliferation of specification languages can be found in the difficulties of combining a specification language with a flexible notion of implementation and an effective verification method.

Formal production of software

All frameworks for the formal development of programs consist of three basic elements: a specification language, a notion of implementation and an inference system to prove implementations correct.

These three elements together allow us to write a high level specification of the problem and refine it successively into more detailed specifications until a

specification is obtained which is so low level that it can be executed, in other words a program. And finally, provided the implementation relation is transitive, the proof system allows us to prove the correctness of the program w.r.t. its original specification by proving that each refinement step is a correct implementation.

The requirements for a good specification language are basically two: sufficient expressive power and simplicity of use. The first requirement should ensure that a specification language allows us to describe the behaviour of the software systems we want to specify. The second requirement concerns various aspects of the language such as modularity, readability, re-usability, etc.

A good notion of implementation is expected to relate a high level specification to a more concrete, detailed or complete specification. Formalizing this relation, including all that our intuition suggests and no more, is a very vague task. Normally, the implementation relation is expected to be transitive and to embody informal concepts such as:

- A implements B if A completes the input/output description given in B.
- A implements B if A can *simulate* B.
- A implements B if A is more *deterministic* than B.

Finally, an inference system should be available for proving that such implementation relations hold, *i.e.* proving that A implements B for any of the cases mentioned above. These proofs are called **correctness proofs** and we can say that A is proven correct w.r.t. B if A is proven to implement B. This terminology extends the idea of a program being correct w.r.t. a specification.

The nature of these proofs depends on the definition of implementation chosen, but it always involves proving that all the input/output requirements in B are satisfied by the models of A or something 'constructed' from them. A good

inference system should be sound and complete with respect to the relation of implementation, so that all valid implementations and only those can be derived in the system.

If we look at the literature we find many proposed notions of implementation but none of them is in very widespread use. The reason comes from the difficulty of combining useful specification languages and implementation concepts with practical proof systems. The different frameworks can be roughly classified according to the priority given to each of these issues.

The most common specification languages are the ones stressing the proof capabilities which are considered the ***bottleneck*** for the framework. For this purpose, the expressive power of specifications and the flexibility of the implementation notion are constrained in different ways so that sensible verification methods are attained.

OBJ [FGJM 85], among others, is developed on top of a re-writing engine. This results in a good proof system for the language but it limits from the beginning all aspects of the specification language and the implementation notion.

Specification languages with semantics expressed in terms of theories (LARCH [GHW 85]) or, even, presentations (finite sets of sentences) provide complete proof systems by definition, provided the underlying logic has a sound and complete inference system. Unlike OBJ these languages do not expect automatic proofs but they still enforce serious constraints on the specification language and the implementation notion.

Other frameworks for formal program development stress the need for a comfortable specification language to ease specification design. Some languages propose the use of a high level of modularity and stronger logics, *e.g.* Clear [BG 80], others

encourage specifications via the use of abstract models allowing the definition of type representations, *e.g.* Z [Spi 85].

Finally, most recent approaches stress the need for a simple, composable and general implementation notion. Among these languages we find most of the specification languages with loose semantics, such as *ASL* [SW 83].

1.2 *ASL* and related languages

ASL was originally presented by D.Sannella and M.Wirsing in [SW 83] and developed later in [Wir 86, ST 88a].

ASL is a specification language designed around a few specification building operations (SBO's) with simple semantics and powerful enough to describe most of the operations in other specification languages and most of the relations of *simulation* used in implementation notions.

At the same time, *ASL* has institution-independent semantics [ST 88a] which means that *ASL* is independent of the logic used to express the requirements in a specification and, also, independent of the nature of the models which can be algebras or other mathematical structures. The notion of implementation used in *ASL* [ST 88b] is institution-independent, transitive (vertically composable) and distributes w.r.t. the structure of specifications (horizontally composable).

The purpose of *ASL* is to investigate the foundations of algebraic specification languages. In this sense, *ASL* shows that a lot of work can be done at the institution-independent level, in particular the semantics of the SBO's, the definition of implementation, results on vertical and horizontal composability, some inference rules for sentences, equivalences among specification terms, etc.

ASL is not so much an established specification language as a tool for research. For this reason, *ASL* often appears with slightly different semantics or SBO's, and

sometimes, *ASL* is completely hidden behind friendlier specification languages such as PLUSS [Gau 84]. In this thesis, we shall also play with the basic SBO's, building a number of different languages and sublanguages. Because of that, we often refer to them collectively as *ASL and related languages.*

Among other specification frameworks, the framework based on *ASL* is distinguished by the stress on generality and flexibility, disregarding systematically the existence (or not) of good verification methods.

Focussing research on specification semantics and implementation has led to a very general framework where verification faces at least as many difficulties as in all other specification frameworks combined.

This thesis seeks a formal system for proving implementation steps to be correct in *ASL* and related languages. In the process, we use *ASL* as a tool to investigate the verification of implementations in general. Analogously to results obtained for the semantics of structuring operations and the implementation notion, we shall show how many verification aspects are independent of the logic used and, also, exhibit trade-offs between sensible verification methods and the expressive power of specifications and the flexibility of implementations.

1.3 Verification: what are the main problems?

In this section the main problems faced in the verification of implementations in *ASL* and related languages are surveyed.

In order to be formal, verification must rely on an inference system, that is, a fixed collection of axioms and inference rules which can be combined in formal

proofs. In this sense, verification techniques based on semantic constructions between models such as [Hoa 72] and [Sch 87] are discarded.

The problem with inference systems is that they work at the level of theories and there are some very common concepts in algebraic specification which can only be properly described at the level of models. There are four main sources of problems:

1. Hiding auxiliary functions is a simple abstraction operation with a very simple semantics at the level of models. However, at the level of theories, theory morphisms are a very poor approximation to hiding.

2. Similarly, behavioural abstraction can only be described by a theory morphism in very concrete cases where a set of observations characterize a behaviour (see chapter 5) whereas defining behavioural abstraction at the level of models only takes a few lines.

3. Induction principles associated to reachability constraints are a second order concept and can be described at a model theoretic level by means of infinitary rules such as

$$\frac{P(0), P(1), ..., P(k), ...}{\forall n.\ P(n)}$$

for natural numbers. On the other hand, there is no first order consistent and complete axiomatizable theory for arithmetic, by Gödel's first incompleteness theorem (*cf.* [Bar 77, BJ 80]), so theories are inadequate for specifying the natural numbers completely.

4. Parameterization produces neat structured specifications if the meaning of the parameters is respected by all the parameterized specifications used, *i.e.*

if they are invariant[1]. At the theory level, conservative extensions are a very poor approximation to invariance.

Due to at least these four reasons, specification languages tend to discard some of these common constructions in order to ease verification.

1.4 Related work

Although this thesis studies verification of implementation steps in *ASL*, the actual work refers to verification of implementation steps in arbitrary specification languages using the mentioned four constructions.

For this reason, this does not only relate to specific work on verification in *ASL* such as [Bre 89], but also to a lot of previous work on proving correctness of implementations w.r.t. structured algebraic specifications.

In the following we review work and results in different frameworks for each of the four main sources of verification problems.

Hiding in the literature

Hiding some of the components specified in an algebraic specification is an old common practice in equational specifications [GTW 76] which is fully justified in [TWW 78, BBTW 81]. The use of hidden parts enhances the expressive power of equational logic to include all functions of interest (semi-computable functions).

---

[1]The more popular term *persistent* is avoided in this context since it is formally defined in chapter 2 for another purpose. However, both *invariance* and *persistency* (as defined in chapter 2) extend the standard notion of *persistency* from functors to SBO's.

For the same reason that using hidden parts enhances expressive power, it prevents an adequate semantics based on presentations or theories, and it makes verification rather difficult.

In the literature, verification is frequently restricted to proving theorems from specifications with hidden parts. This is not a major problem and an institution-independent and complete rule is given in [ST 88a].

On the other hand, proving implementations correct w.r.t. specifications with hidden parts is difficult. Proofs are frequently *ad hoc*, exploiting the implementation of the hidden components (if present) or using a general *weakening* rule for adding some extra components to the implementation.

Similarly, in abstract model specifications many details of the representation are hidden, *i.e.* they do not need to be implemented. In this context, the problem is known as *specification bias* and it gives rise to known problems in the verification of data reifications (see [Jon 86]).

Some specification languages such as CLEAR [BG 80] and LARCH [GHW 85] prefer to substitute true hiding by a theory-based SBO. This is not a solution to the problem but a decision to give up true hiding in exchange for another SBO with a better proof treatment but with less expressive power and, probably, less clear semantics.

Behavioural abstraction in the literature

Algebraic specifications aim to describe the input/output behaviour of a system. Good algebraic frameworks for software development should allow any system respecting the input/output behaviour of a specification to be among its valid implementations. Unfortunately, this is frequently not the case.

Since early work in [GGM 76] the initial approach to specification has been questioned for its unsatisfactory treatment of behaviour. The final approach and, more successfully, the initial-behavioural approach give semantics to equational specifications in such a way that all desired implementations are valid.

This improvement of the semantics results in a loss on the proof-theoretical side. In the initial approach all ground terms are different unless they can be proven equal using equational logic [GM 81, MS 85], *i.e.* equational logic is complete for ground equations. This property is lost in the final and initial-behavioural approaches.

Theorem proving is still feasible in the final and initial-behavioural approaches using proofs by consistency [KM 87, Lin 87] but they are rather more cumbersome. Note, also, that for proving implementations correct, the initial approach is much simpler since all requirements for ground terms are *encapsulated* in a finite presentation.

In many approaches abstraction is related to an explicit or implicit collection of *observable* sentences or terms, *e.g. ASL* [Wir 86, ST 87]. In these cases correctness proofs might proceed by proving that all observable consequences of a specification are satisfied in its implementation. In practice, some of these proofs are not difficult using induction on observable contexts [Hen 88, Hen 89].

Outside the scope of equational specifications, specification languages with loose semantics frequently interpret the equality symbol used in specifications as an arbitrary equivalence relation or congruence [WB 89]. This solution works for some well-known examples where the implementation of the equality symbol involves, basically, an abstraction function from the values of the implementation to those in the specification. However, no attempt has been made to relate this technique to a formal notion of behaviour.

Finally, in abstract model specifications a strong concern with behavioural abstraction has led to abandoning implementation relations based on abstraction functions, as has been common practice since [Hoa 72], in favor of relations between abstract and concrete values [Nip 86, Jon 86, Sch 87]. This results in more complicated proofs of correctness.

Reachability constraints in the literature

A reachability constraint prevents models of an abstract data type from having more values than those finitely generated by its operations; *i.e.* a reachability constraint imposes an induction principle. This technique is frequently used in algebraic specifications in order to specify standard models such as natural numbers, finite lists, etc.

Reachability constraints can be explicitly applied in some specification languages but, often, they are automatically imposed by the semantics of the language. In any case, reachability constraints increase the expressive power of first order specification languages but make a complete theorem prover out of the question [MS 85].

The lack of a complete proof system for first order theories with an induction principle does not seem very serious since, very frequently, a common induction rule such as structural induction [Bur 69] allows all desired theorems to be proven.

Nevertheless, for proving implementations correct, reachability constraints in one specification must be *satisfied* by reachability constraints in another specification. This aspect of proofs of correctness has long been neglected [Far 89].

Some theory-based specification languages prefer to substitute true induction principles by a structural induction schema (see the LARCH semantics in [GHW 85]); this results in a simpler proof system but reduces the expressive

power, *e.g.* it prevents specifications from distinguishing between standard and non-standard natural numbers.

Persistency in the literature

Parameterized specifications are persistent if they do not affect the meaning of their parameters.

Since early work in the initial approach [TWW 78, Ore 83, EM 85] persistency has been shown to be a sufficient and *almost* necessary property for interpreting parameter passing as *putting together* the equations from the parameterized specification and the parameter. In other words, persistency allows the prover to ignore the functional semantics of parameterized specifications and work *safely* with the combination of equational presentations.

Conversely, if parameter passing is defined to be a putting together of sets of equations, persistency guarantees a functional interpretation of parameterized specifications which results in clearer semantics allowing structured implementation and verification.

Results in [Ore 83] show persistency to be a necessary condition for *correct* parameter passing in the initial approach. H.Ganzinger presents in [Gan 83] a proof-theoretic sufficient condition for persistency in the same context.

For specification languages with loose semantics the standard notion of persistency must be generalized from relations between models to relations between classes of models. In this thesis, we use two different generalizations, one called *persistency* and another called *invariance*, both defined in chapter 2.

In the context of specification languages with loose semantics, persistency (resp. invariance) is rarely mentioned [Ber 87]. However, the basic concerns about structured implementation and verification are explicitly stated [GB 80, ST 88b].

In practice, performing structured correctness proofs depends a great deal on preserving the meaning of whole subspecifications during implementation steps [Far 89]. This idea is formalized in this thesis by the relationship between *invariance* and a calculus for the management of subspecifications in proofs of correctness.

1.5 Overview of the thesis

ASL is basically the combination of five SBO's plus mechanisms for abstraction and parameterization. Starting from a subset of three of these SBO's and adding, later, the rest one by one or in groups we define and study many different sublanguages of *ASL*.

This thesis progresses from simpler to more difficult sublanguages studying the verification of implementation steps in each of them. This journey includes the study of theorem proving in *ASL* and of the four main sources of verification problems mentioned above.

Chapter 2 introduces the necessary background for the thesis. In particular, it includes the definition of the specification language, the implementation relation, and the notion of logical system (institution) which parameterizes the specification language.

Chapter 3 gives an account of the results obtained by substituting actual specifications by theories *approximating* them. In this way, verification of correctness of implementation steps is reduced to theorem proving. This procedure is shown to be insufficient unless specifications are severely constrained. This chapter introduces the notion of *M-completeness* for characterizing inference rules, shows

that the rule for hiding is not M-complete, and gives various inference rules for common algebraic constructors.

Chapter 4 deals exclusively with the problem of hiding, first in isolation and then in the context of the other specification building operations. This chapter defines an institution-independent strategy for proving implementations correct w.r.t. specifications with hidden parts, shows that this strategy is sound and complete provided some conditions are met, and discusses the nature of these side conditions as well as how they can be established.

Chapter 5 deals with the problem of abstraction and behavioural abstraction. First some general notions related to the definition of abstraction are discussed, with the result that abstraction is identified with a certain kind of hiding and verification techniques for it are therefore inherited from chapter 4. Behavioural abstraction and related proof techniques used in the literature are analyzed in the context of our general results.

Chapter 6 deals with the problem of reachability constraints. In this case, there is little to say for an arbitrary institution, so detailed research is dedicated to the algebraic case. An inference system for the inference of reachability constraints in the algebraic framework is presented and shown to be sound and complete for single constraints and sets of independent constraints. Some known concepts such as sufficient completeness are revisited in relation to the interaction of reachability constraints and equations.

Finally, chapter 7 faces the matter of structure in specifications. In particular, it studies in which cases the structure of specifications can be used to structure proofs of correctness. Notions such as invariance turn out to be essential and are introduced via an inference system for shared subspecifications.

In summary, we expect to provide a wide review of the most important problems in the formal verification of implementations, working in a framework general

enough to include most possible cases. This panoramic view of the situation should help to distinguish the *good* kind of specifications and implementations, *i.e.* those which possess adequate proof methods.

Chapter 2

Preliminaries

2.1 Institutions

The concept of institution [GB 84] formalizes the idea of *logical system.* An institution is a category of models and a family of sentences, both indexed by a category of signatures and with a relation of satisfaction between models and sentences, such that satisfaction is preserved by translations along signature morphisms. These minimal requirements allow very many logical systems to be viewed as institutions, in particular, most of those used in formal specifications such as equational logic, Horn clause logic, higher order logic, infinitary logic, etc.

Definition 2.1.1 *An* **institution I** *consists of*

1. *a category SIG, whose objects are called signatures,*

2. *a functor Sen : SIG $\rightarrow$ Set, giving for each signature Σ a set whose elements are called sentences over that signature, Σ-sentences,*

3. *a functor Mod : SIG $\rightarrow Cat^{op}$ giving for each signature Σ a category whose objects are called models over that signature, Σ-models, and whose arrows are called Σ-morphisms,*

4. *and a relation* $\models_\Sigma \subseteq |Mod(\Sigma)| \times Sen(\Sigma)$ *for each* $\Sigma \in |SIG|$, *called* Σ-*satisfaction, such that for each morphism* $\sigma : \Sigma 1 \rightarrow \Sigma 2$ *in SIG, the satisfaction condition*

$$M \models_{\Sigma 2} Sen(\sigma)(\varphi) \quad \text{iff} \quad Mod(\sigma)(M) \models_{\Sigma 1} \varphi$$

holds for all $M \in |Mod(\Sigma 2)|$ *and* $\varphi \in Sen(\Sigma 1)$.

It is common practice to write $\sigma(\varphi)$ instead of $Sen(\sigma)(\varphi)$ and $M|_\sigma$ instead of $Mod(\sigma)(M)$, so that the satisfaction condition can be expressed as

$$M \models_{\Sigma 2} \sigma(\varphi) \quad \text{iff} \quad M|_\sigma \models_{\Sigma 1} \varphi$$

It is also very common to extend Σ-satisfaction from sentences to sets of sentences, so that for any $\Phi \subseteq Sen(\Sigma)$ and model $M \in |Mod(\Sigma)|$ satisfaction is defined as:

$$M \models_\Sigma \Phi \stackrel{\text{def}}{\iff} \forall \varphi \in \Phi.\ M \models_\Sigma \varphi$$

Similarly, Σ-satisfaction can be extended to a relation between sets of Σ-sentences, so that for any $\Phi 1, \Phi 2 \subseteq Sen(\Sigma)$ satisfaction is defined as:

$$\Phi 1 \models_\Sigma \Phi 2 \stackrel{\text{def}}{\iff} \forall M \in |Mod(\Sigma)|.\ M \models_\Sigma \Phi 1 \Rightarrow M \models_\Sigma \Phi 2$$

Particular cases include the satisfaction of a sentence by a set of sentences $\Phi \models_\Sigma \varphi$ and satisfaction between single sentences, $\varphi \models_\Sigma \varphi'$, for $\varphi, \varphi' \in Sen(\Sigma)$ and $\Phi \subseteq Sen(\Sigma)$.

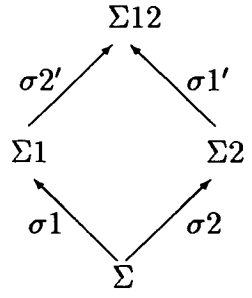
Occasionally, we shall use the notion $\not\models_\Sigma$ between Σ-models and sets of Σ-sentence to mean that a Σ-model does not satisfy any sentence in a set of Σ-sentences.

These concepts were first introduced in [BG 80] and studied further in [GB 84, GB 86, GB 90, Tar 86a, Tar 86b]. The concept of institution allows the structural mechanisms of a specification language to be defined for an arbitrary institution [ST 88a], so that a great deal of work can be done, once and for all, at an institution independent level and reused for any particular institution.

Sometimes the notion of institution is too general and some extra conditions are frequently added. Unless the contrary is stated, we shall assume institutions to have a finitely co-complete category of signatures and a finitely co-continuous model functor, like *institutions with composable signatures* in [ST 92] and *reasonable and exact institutions* in [DGS 91].

Intuitively, these properties allow the symbols defined in different signatures to be *put together* in a single signature without confusion but respecting "shared" symbols, in particular a pushout signature is a finite co-limit in *SIG*. Moreover, if the model functor *Mod* preserves finite co-limits (finitely co-continuous), every finite co-limit diagram in *SIG* produces a limit diagram in *Cat*. In particular these properties imply the amalgamation lemma (cf. [EM 85, ST 88b]).

Lemma 2.1.2 (Amalgamation lemma) *Let* $\mathbf{I} = \langle SIG, Mod, Sen, \models \rangle$ *be an institution such that SIG is finitely co-complete and Mod is finitely co-continuous. Consider a pushout in SIG as in the diagram:*



Then, for any two models $A \in |Mod(\Sigma 1)|$ *and* $B \in |Mod(\Sigma 2)|$ *such that* $A|_{\sigma 1} = B|_{\sigma 2}$, *there exists a unique* $\Sigma 12$*-model* $A \oplus B$ *such that* $(A \oplus B)|_{\sigma 2'} = A$ *and* $(A \oplus B)|_{\sigma 1'} = B$. $A \oplus B$ *is called the* **amalgamated union** *of* A *and* B.

A fundamental piece missing in the definition of institution is an inference system allowing a sentence to be concluded from a set of sentences in a given logical system. That is why sometimes a institution $\mathbf{I}$ is equipped with a sound inference system $\vdash^{\mathbf{I}}$, as in [HST 89a].

Usually an inference system $\vdash^{\mathbf{I}}$ is presented in the form of some axioms and some inference rules which allow us to conclude judgements of the form

$$\Phi \vdash^{\mathbf{I}}_{\Sigma} \varphi$$

where φ is a Σ-sentence and Φ is a finite set of Σ-sentences for a signature Σ in $\mathbf{I}$.

Therefore, an inference system $\vdash^{\mathbf{I}}$ denotes a r.e. relation between finite sets of sentences and sentences over each signature. We shall call this relation the **consequence relation of** $\vdash^{\mathbf{I}}$ written $\mathcal{R}_{\Sigma}(\vdash_{\mathbf{I}}) \subseteq \mathcal{P}(Sen(\Sigma)) \times Sen(\Sigma)$ for every signature Σ in $\mathbf{I}$,

$$\Phi \vdash^{\mathbf{I}}_{\Sigma} \varphi \iff^{\text{def}} (\Phi, \varphi) \in \mathcal{R}_{\Sigma}(\vdash^{\mathbf{I}})$$

An inference system $\vdash^{\mathbf{I}}$ is **sound** w.r.t. $\mathbf{I}$ if

$$\Phi \vdash^{\mathbf{I}}_{\Sigma} \varphi \implies \Phi \models_{\Sigma} \varphi$$

And it is **complete** if the converse holds,

$$\Phi \vdash^{\mathbf{I}}_{\Sigma} \varphi \impliedby \Phi \models_{\Sigma} \varphi$$

The most common institutions used in formal specifications, such as equational logic, first order logic, Horn clause logic, etc., are equipped with sound and complete inference systems. This thesis will assume that every institution $\mathbf{I}$ is equipped with a sound inference system $\vdash^{\mathbf{I}}$; completeness of $\vdash^{\mathbf{I}}$ will not be required.

Finally we need a concept to relate different institutions. Although some details have been a topic of discussion, we shall adopt institution morphisms as defined in [GB 84] with a slight technical difference as proposed in [ST 92].

Definition 2.1.3 *Let*

$$\mathbf{I1} = \{SIG1, Mod1, Sen1, \models^1\} \quad \textit{and} \quad \mathbf{I2} = \{SIG2, Mod2, Sen2, \models^2\}$$

be two institutions. Then, an **institution morphism** $\alpha : \mathbf{I1} \to \mathbf{I2}$ *consists of*

1. *a functor* $\alpha_{SIG} : SIG1 \to SIG2$,

2. *a natural transformation* $\alpha_{Mod} : Mod1^{op} \Rightarrow \alpha_{SIG}^{op} ; Mod2^{op}$, [1] *that is for each* $\Sigma \in |SIG1|$ *(i.e.* $\Sigma \in |SIG1^{op}|$*) there exists a functor*

$$\alpha_{Mod}\Sigma : Mod1(\Sigma) \to Mod2(\alpha_{SIG}(\Sigma))$$

which is consistent with changes of signature; i.e. for all signature morphism $\sigma : \Sigma \to \Sigma'$ *in* $SIG1$ *(i.e.* $\sigma^{op} : \Sigma' \to \Sigma$ *in* $SIG1^{op}$*),*

$$Mod1^{op}(\sigma^{op}); \alpha_{Mod}\Sigma = \alpha_{Mod}\Sigma'; (\alpha_{SIG}^{op}; Mod2^{op})(\sigma^{op})$$

3. *a natural transformation* $\alpha_{Sen} : \alpha_{SIG}; Sen2 \Rightarrow Sen1$, *that is for each* $\Sigma \in |SIG1|$ *there exists a function,*

$$\alpha_{Sen}\Sigma : Sen2(\alpha_{SIG}(\Sigma)) \to Sen1(\Sigma)$$

which is consistent with changes of signature; i.e. for all signature morphism $\sigma : \Sigma \to \Sigma'$ *in* $SIG1$

$$\alpha_{Sen}\Sigma; Sen1(\sigma) = (\alpha_{SIG}; Sen2)(\sigma); \alpha_{Sen}\Sigma'$$

such that the following satisfaction condition holds

$$M \models^1_\Sigma \alpha_{Sen}\Sigma(\varphi) \quad \text{iff} \quad \alpha_{Mod}\Sigma(M) \models^2_{\alpha_{SIG}(\Sigma)} \varphi$$

for all $\Sigma \in |SIG1|$, $M \in Mod1(\Sigma)$ *and* $\varphi \in Sen2(\alpha_{SIG}(\Sigma))$.

Intuitively, an institution morphism describes the situation in which satisfaction is independent of the change of institution.

---

[1] Recall that $Mod1^{op} : SIG1^{op} \to Cat$, $Mod2^{op} : SIG2^{op} \to Cat$ and $\alpha_{SIG}^{op} : SIG1^{op} \to SIG2^{op}$ are as functors $Mod1 : SIG1 \to Cat^{op}$, $Mod2 : SIG2 \to Cat^{op}$ and $\alpha_{SIG} : SIG1 \to SIG2$ but considered between the opposite categories.

2.2 *ASL* and constructors

In the following we give the syntax and semantics of a specification language like the one described in [SW 83, ST 88a, Wir 86] parameterized by an institution **I**. Note that here we use the name *ASL* for the union of six specification building operations — SBO's for short — to which parameterization mechanisms, an abstraction operation and some other SBO's called constructors may be added *a posteriori*.

Syntax

ASL(**I**) is a specification language with raw terms

$$SP == A_\Phi SP \mid T_\sigma SP \mid D_\sigma SP \mid M_{(\sigma,\iota)} SP \mid SP\ U\ SP \mid \epsilon_\Sigma$$

where Φ is a finite set of sentences, Σ a signature and σ and ι signature morphisms in the institution **I**.

Now, we can recursively define a signature for each of these terms.

Definition 2.2.1 *Given an institution* **I**, *the* **signature** *of an ASL*(**I**) *term SP, written Sig*[*SP*], *is defined as follows:*

$$Sig[A_\Phi SP] = Sig[SP] \qquad Sig[T_\sigma SP] = \uparrow\sigma$$
$$Sig[D_\sigma SP] = \downarrow\sigma \qquad Sig[M_{(\sigma,\iota)} SP] = Sig[SP]$$
$$Sig[SP1\ U\ SP2] = Sig[SP1] \qquad Sig[\epsilon_\Sigma] = \Sigma$$

where the notation $\downarrow\sigma$ *and* $\uparrow\sigma$ *refers respectively to the source and target objects of an arrow* σ.

Among all raw terms we can distinguish the well-formed terms, which are those satisfying the following context conditions:

Definition 2.2.2 *Given a specification SP of signature Σ,*

- *$A_\Phi SP$ is well-formed if $\Phi \subseteq Sen(\Sigma)$ and SP is well-formed.*
- *$T_\sigma SP$ is well-formed if $\downarrow \sigma = \Sigma$ and SP is well-formed.*
- *$D_\sigma SP$ is well-formed if $\uparrow \sigma = \Sigma$ and SP is well-formed.*
- *$M_{\langle \sigma, \iota \rangle} SP$ is well-formed if $\uparrow \sigma = \downarrow \iota$ and $\uparrow(\sigma; \iota) = \Sigma$ and SP is well-formed.*
- *$SP1$ U $SP2$ is well-formed if $SP1$ and $SP2$ have the same signature and $SP1$ and $SP2$ are both well-formed.*
- *ϵ_Σ is well-formed.*

Semantics

Every well-formed specification SP in $ASL(\mathbf{I})$ denotes a signature $Sig[SP]$ in SIG and a class of models $Mod[SP]$ among the models $|Mod(Sig[SP])|$. The signature of a well-formed specification $Sig[SP]$ is as defined for all specification terms and $Mod[SP]$ is defined recursively as follows:

Definition 2.2.3 *Given an institution $\mathbf{I} = \langle SIG, Mod, Sen, \models \rangle$, the class of models denoted by a well-formed specification in $ASL(\mathbf{I})$, written $Mod[SP]$. is defined as follows:*

$$Mod[A_\Phi SP] = \{M \in Mod[SP] \mid M \models \Phi\}$$
$$Mod[T_\sigma SP] = \{M \in |Mod(\uparrow \sigma)| \mid M|_\sigma \in Mod[SP]\}$$

$$Mod[D_\sigma SP] = \{M|_\sigma \mid M \in Mod[SP]\}$$
$$Mod[M_{(\sigma,\iota)}SP] = \{M \in Mod[SP] \mid M|_\iota \text{ is } \sigma\text{-minimal}\}$$
$$Mod[SP1 \ U \ SP2] = Mod[SP1] \cap Mod[SP2]$$
$$Mod[\epsilon_\Sigma] = |Mod(\Sigma)|$$

where a model $M \in |Mod(\uparrow\sigma)|$ *is* σ-**minimal** *if for every model* $N \in |Mod(\uparrow\sigma)|$, *any monomorphism* $m : N \rightarrow M$ *such that* $m|_\sigma : N|_\sigma \rightarrow M|_\sigma$ *is an isomorphism, is in fact an isomorphism in* $Mod(\uparrow\sigma)$.

Intuitively, A_Φ is the basic specification mechanism requiring some axioms to be satisfied. These axioms can be a list of equations, some first order axioms or something else depending on the institution used.

T_σ and D_σ provide the mechanisms to enlarge, reduce and modify the signature of a specification and the models accordingly, *e.g.* in the course of an enrichment or hiding. According to their original names in [ST 88a], T_σ and D_σ for $\sigma : \Sigma \rightarrow \Sigma'$ should be read as:

- *translate a* Σ*-specification SP to another signature* Σ' *along* σ, and
- *derive a* Σ*-specification from a specification SP over a richer signature* Σ' *using* σ

respectively.

U allows specifications over the same signature to be combined, giving the union of their requirements. Frequently, the specifications to be combined refer to different components of a system and, therefore, they are defined over different signatures. In this case, a T_σ operator should be used to expand the signature of each specification to a common one so that U can be applied.

Although defined as a binary SBO, U is often used as an n-ary operation meaning a successive application of binary U operations. Associativity of U makes the order of application irrelevant.

$M_{\langle\sigma,\iota\rangle}$ allows some *basic models* of a specification to be selected. The precise meaning of *basic model* varies according to the meaning of the arrows in the category of models. Later, we shall see that taking models to be algebras and monomorphisms in $Mod(\uparrow\sigma)$ to be injective homomorphisms, $M_{\langle\sigma,\iota\rangle}$ constrains some sorts to be finitely generated. For a given target signature $\uparrow\iota$, the choice of σ and ι decides which sorts are finitely generated and which operations are the generators.

Occasionally in some proofs we may use another SBO, namely $\{\_\}$. This operation takes a model and delivers a specification as follows: Given a signature Σ and a Σ-model A,

- $\{A\}$ is well-formed

- $Sig[\{A\}] = \Sigma$ and $Mod[\{A\}] = \{A\}$

Equality between *ASL* specifications will be often used to indicate that two specifications have the same signature and class of models

$$SP1 = SP2 \quad \textit{iff} \quad Sig[SP1] = Sig[SP2] \quad \textit{and} \quad Mod[SP1] = Mod[SP2]$$

We shall use the name **selectors** for those SBO's L which always yield a specification with a class of models contained in the class of models of the parameter,

i.e. $Mod[L\ SP] \subseteq Mod[SP]$ for any specification SP. In ASL, A_Φ, $M_{\langle\sigma,\iota\rangle}$ and U[2] are selectors.

We shall use the name **constructors** for those SBO's k whose semantics are defined pointwise by a function f_k over the class of models of the parameter, *i.e.* $Mod[k\ SP] = \{f_k(M) \mid M \in Mod[SP]\}$ for any specification SP. In ASL, D_σ is a constructor where f_{D_σ} is $\_|_\sigma$.

There some general facts of interest holding for the different SBO's.

Proposition 2.2.4 *Specification building operations U, A_Φ, $M_{\langle\sigma,\iota\rangle}$, T_σ and all constructors are* **monotonic** *w.r.t. inclusion of model classes, i.e. if $Mod[SP1] \subseteq Mod[SP2]$ then $Mod[\xi SP1] \subseteq Mod[\xi SP2]$ for all SBO's ξ and specifications $SP1$ and $SP2$.*

Proof Immediate from their definition. □

Because of this fact, the following inequality

$$Mod[\xi SP1]\ \cup\ Mod[\xi SP2]\ \subseteq Mod[\xi SP12]$$

where $Mod[SP12] = Mod[SP1] \cup Mod[SP2]$ holds in general. Moreover, the opposite inclusion holds for $SP1$, $SP2$ built using the SBO's defined above.

Proposition 2.2.5 *Specification building operations U, A_Φ, $M_{\langle\sigma,\iota\rangle}$, T_σ and all constructors distribute over the union of model classes, i.e. for all specifications SP', $SP1, SP2$ and $SP12$ such that $Mod[SP12] = Mod[SP1] \cup Mod[SP2]$,*

$$Mod[A_\Phi SP12] = Mod[A_\Phi SP1]\ \cup\ Mod[A_\Phi SP2]$$

---

[2]More precisely, not U but the unary SBO's "$\_\ U\ SP2$" and "$SP1\ U\ \_$" for arbitrary specifications $SP1$ and $SP2$, are selectors.

$Mod[M_{\langle\sigma,\iota\rangle}SP12] = Mod[M_{\langle\sigma,\iota\rangle}SP1] \cup Mod[M_{\langle\sigma,\iota\rangle}SP2]$

$Mod[SP' \ U \ SP12] = Mod[SP' \ U \ SP1] \cup Mod[SP' \ U \ SP2]$

$Mod[SP12 \ U \ SP'] = Mod[SP1 \ U \ SP'] \cup Mod[SP2 \ U \ SP']$

$Mod[T_\sigma SP12] = Mod[T_\sigma SP1] \cup Mod[T_\sigma SP2]$

$Mod[k \ SP12] = Mod[k \ SP1] \cup Mod[k \ SP2]$ *for all constructors* k

Proof By definition, the fact that a model $M \in Mod[SP12]$ is included or not in $Mod[A_\Phi SP12]$, $Mod[M_{\langle\sigma,\iota\rangle}SP12]$, $Mod[SP' \ U \ SP12]$ and $Mod[SP12 \ U_\bullet SP']$ depends on M and on Φ, $M_{\langle\sigma,\iota\rangle}$ and SP' respectively, but *not* on which other models appear in $SP12$. This "pointwise" selection of the models results in distributivity over the union of model classes.

In the case of T_σ and an arbitrary constructor k the resulting class of models $Mod[T_\sigma SP12]$ (and $Mod[k \ SP12]$ respectively) is defined pointwise for each model in $Mod[SP12]$. Hence, distributivity also holds.

□

Distributivity of a SBO over the union of model classes implies that that SBO is defined pointwise, *i.e.* $Mod[\xi SP] = \bigcup_{A \in Mod[SP]} Mod[\xi\{A\}]$. This property is not to be confused with distributivity of a SBO over U.

Proposition 2.2.6 *Specification building operations* U, A_Φ, $M_{\langle\sigma,\iota\rangle}$, T_σ *and all injective constructors distribute over* U, *i.e.* $Mod[\xi(SP1 \ U \ SP2)] = Mod[\xi SP1 \ U \ \xi SP2]$.

Proof Since all these SBO's distribute over the union of model classes, we have that

$$Mod[\xi(SP1 \ U \ SP2)] = \bigcup_{A \in Mod[SP1 \ U \ SP2]} Mod[\xi\{A\}]$$

and, by definition of U,

$$Mod[\xi(SP1 \ U \ SP2)] = \bigcup_{A \in Mod[SP1] \cap Mod[SP2]} Mod[\xi\{A\}] \subseteq Mod[\xi SP1 \ U \xi SP2]$$

On the other direction, we must ensure that each SBO ξ satisfies that

$$\forall A \in Mod[\xi\{M1\}], B \in Mod[\xi\{M2\}].\ A \neq B$$

for any two different models $M1$ and $M2$.

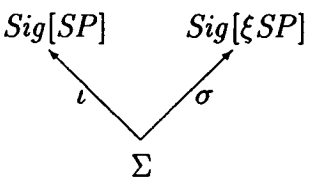
Injective constructors are just those constructors satisfying this property.

The same property holds trivially for selectors since they do not change any model, and also for T_σ since two models cannot be equal and have two different reducts.

□

There are two other properties of interest of an arbitrary SBO's w.r.t. a specification.

Definition 2.2.7 *Given a specification building operation ξ, a specification SP and a pair of morphisms $\langle \iota, \sigma \rangle$ such that $\uparrow \iota = Sig[SP]$, $\uparrow \sigma = Sig[\xi SP]$ and $\Sigma = \downarrow \iota = \downarrow \sigma$, then*



- *ξ is* **persistent** *w.r.t. SP and $\langle \iota, \sigma \rangle$ iff for all models $A \in Mod[SP]$, there exists a model $B \in Mod[\xi SP]$ with the same reduct, $A|_\iota = B|_\sigma$.*

- *ξ is* **invariant** *w.r.t. SP and $\langle \iota, \sigma \rangle$, iff for all models $A \in Mod[SP]$, every model $B \in Mod[\xi\{A\}]$ has the same reduct, $A|_\iota = B|_\sigma$.*

Note that a SBO ξ can be persistent w.r.t. a specification SP and a pair of morphisms $\langle \iota, \sigma \rangle$ without being invariant and vice versa. In the case of persistency, invariance may fail because for some $A \in Mod[SP]$ the reduct $A|_\iota$ belongs to $Mod[D_\sigma \xi SP]$ but not to $Mod[D_\sigma \xi\{A\}]$. In the opposite direction, if invariance holds, for some $A \in Mod[SP]$ it may happen that $Mod[\xi\{A\}] = \emptyset$ and that $A|_\iota \notin Mod[D_\sigma \xi SP]$.

$k{+}ASL$

In a stepwise process of refinement, it is often convenient to mix parts of a specification with *pieces of program* corresponding to parts of the specification which have already been implemented.

In order to formalize this, we can consider these *pieces of program* to be also SBO's. By doing so, we enrich ASL to give a richer specification language $k{+}ASL$ by adding a generic specification building operation $\langle P, \Lambda \rangle$, where P stands for a piece of program and Λ for some properties P satisfies.

Definition 2.2.8 *A programming language $\mathcal{L}$ is* **compatible** *with an institution $\mathbf{I} = \langle SIG, Mod, Sen, \models \rangle$ if each program P in $\mathcal{L}$ denotes a triple*

$$[\![P]\!] = (\Sigma 1, \Sigma 2, f_P)$$

where $\Sigma 1, \Sigma 2 \in |SIG|$ and $f_P : |Mod(\Sigma 1)| \rightarrow |Mod(\Sigma 2)|$.

Then, the new SBO can be formally defined as follows.

Definition 2.2.9 *Given an institution $\mathbf{I} = \langle SIG, Mod, Sen, \models \rangle$ and a compatible programming language $\mathcal{L}$, and let P be a program in $\mathcal{L}$ with semantics*

$$[\![P]\!] = (\Sigma 1, \Sigma 2, f_P)$$

then, $\langle P, \Lambda \rangle SP$ is well-formed if SP is well-formed, $Sig[SP] = \Sigma 1$ and $\Lambda \subseteq \mathcal{P}(Sen(\Sigma 1)) \times Sen(\Sigma 2)$ such that for all $(\Phi, \varphi) \in \Lambda$ and $M \in |Mod(\Sigma 1)|$,

$$M \models_{\Sigma 1} \Phi \quad \Rightarrow \quad f_P(M) \models_{\Sigma 2} \varphi$$

Moreover, the signature and models for such a well-formed $\langle P, \Lambda \rangle SP$ are defined as:

- $Sig[\langle P, \Lambda \rangle SP] = \Sigma 2$

- $Mod[\langle P, \Lambda \rangle SP] = \{f_P(M) \mid M \in Mod[SP]\}$

The semantics of $\langle P, \Lambda \rangle$ are completely defined by the program P. The purpose of Λ is to state all the relevant properties of P in order to allow verification of specification entailments involving such a *program* (see chapters 3 and 7).

Note that these *pieces of program* are SBO's in $k{+}ASL$ and, more precisely, they are constructors, the so-called **generic constructors**.

Now, $k{+}ASL$ is not only parameterized by an institution but also by the programming language used in the generic constructors, thus $k{+}ASL(\mathbf{I}, \mathcal{L})$.

$\lambda{+}ASL$

Normally ASL comes equipped with a parameterization mechanism based on λ-abstraction (see for example [SW 83, Wir 86, ST 88a]). Here, we consider such a mechanism as an extra feature which can be added to ASL, *i.e.* to the original six SBO's, to make up the richer language $\lambda{+}ASL$.

Raw terms in $\lambda{+}ASL$ are those of ASL plus those generated by the inclusion of specification variables and lambda applications:

$$SP == \ldots \mid X_\Sigma \mid (\lambda\, X : \Sigma.\ SP)SP$$

the signature of which is defined by:

$$Sig[X_\Sigma] = \Sigma \qquad\qquad Sig[(\lambda\, X : \Sigma.\ SP1)SP2] = Sig[SP1]$$

Well-formed terms in $\lambda{+}ASL$ are those closed terms whose actual parameters are of the required signature and which respect all the rules laid down in the definition of well-formed term in ASL. This can be summarized saying that well-formed terms in $\lambda{+}ASL$ are recursively defined by the 6 rules given in the definition of a well-formed ASL term (Definition 2.2.2) plus the following seventh rule:

- $(\lambda\, X : \Sigma.SP1)SP2$ is **well-formed** if $Sig[SP2] = \Sigma$ and $SP2$ and $SP1[\epsilon_\Sigma/X_\Sigma]$ are both well-formed.

where $SP1[\epsilon_\Sigma/X_\Sigma]$ is the specification term $SP1$ where all free occurrences of X_Σ have been substituted by ϵ_Σ.

Finally, the models of a well-formed term are defined by adding:

$$Mod[(\lambda\, X : \Sigma.\ SP1)SP2] = Mod[SP1[SP2/X_\Sigma]]$$

to those rules laid down for ASL. By definition of well-formed term in $\lambda + ASL$, solving all parameter substitutions reduces any well-formed $\lambda + ASL$ term to a well-formed ASL term for which Mod has already been defined. As for simply typed lambda calculus, the Church-Rosser property guarantees Mod to be well-defined over well-formed terms in $\lambda + ASL$ and, in particular, $Sig[(\lambda\, X : \Sigma.\ SP1)SP2] = Sig[SP1[SP2/X_\Sigma]]$ for well-formed terms.

Since well-formed terms are always closed, the signature of each variable can always be inferred from the context. For this reason, subscripts indicating the signature of each variable will be dropped.

Lambda expressions of the form $(\lambda\, X : \Sigma.\ SP)$ are operations which can be used to build up specifications, *i.e.* they are SBO's but not specifications. For this reason verification of structured specifications, treated in the last chapter of the thesis, is mostly about arbitrary specification terms satisfying some semantical conditions between terms and subterms. Parameterization can be used to *group* explicitly (at the syntactical level) specification terms with semantical significance.

SBO's of the form $(\lambda\, X : \Sigma.\ SP)$ are monotonic w.r.t. the inclusion of model classes like ordinary SBO's in ASL, however. it may be the case that $(\lambda\, X : \Sigma.\ SP)$ does not distribute over the union of model classes despite the fact that all SBO's

used in *SP* distribute. This is due to the possible multiple occurrences of a formal parameter X in *SP* [SST 90].

$\alpha + ASL$

Occasionally we may use the name $\alpha + ASL$ to refer to the extension of *ASL* with an abstraction SBO as in [SW 83, ST 87]. However, no details will be given of that new operation until chapter 5 where it will be shown to be redundant.

2.3 *ASL*(EQ) — *ASL* in the algebraic institution

Although most of the work in this thesis is valid for an arbitrary institution, in section 6.3 referring to reachability constraints, sections 3.4, 6.4.1 and 7.3.1 referring to *FQRD*-constructors (see definition below), and section 5.3.1 referring to behavioural abstraction, the algebraic institution is used.

This is probably the most widely used institution in work on formal specification and merits some special attention. In this section we review some basic definitions from algebra and revisit *ASL*(I) in the case where I is the algebraic institution EQ, *i.e.* *ASL*(EQ).

The algebraic institution : EQ

In the following the institution EQ is defined by providing a category of signatures *SIG*, a model functor $Alg : SIG \rightarrow Cat^{op}$, a sentence functor $Eq : SIG \rightarrow Set$ and satisfaction relations $\{\models_\Sigma\}_{\Sigma \in |SIG|}$.

A **signature** is a pair $\langle S, \Omega \rangle$ where S is a set of names (sort names) and Ω is a family of sets of names (operation names) $\{\Omega_{\omega,s}\}_{\omega \in S^*, s \in S}$. An element f of $\Omega_{\omega,s}$ is denoted by $f : \omega \rightarrow s$.

A **signature morphism** $\sigma : \langle S, \Omega \rangle \rightarrow \langle S', \Omega' \rangle$ is a pair $\langle \sigma_S, \sigma_\Omega \rangle$ where $\sigma_S : S \rightarrow S'$ and σ_Ω is a family of sets of functions $\{\sigma_{\omega,s} : \Omega_{\omega,s} \rightarrow \Omega'_{\sigma^*_S(\omega),\sigma_S(s)}\}_{\omega \in S^*, s \in S}$ where $\sigma^*_S(\omega)$ means application of σ_S to every component of ω, *i.e.* $\sigma^*_S(s_1...s_n) = \sigma_S(s_1)...\sigma_S(s_n)$.

Two signature morphisms $\sigma 1 : \langle S, \Omega \rangle \rightarrow \langle S', \Omega' \rangle$ and $\sigma 2 : \langle S', \Omega' \rangle \rightarrow \langle S'', \Omega'' \rangle$ compose by composing the functions they consist of; *i.e.*

$$\sigma 1; \sigma 2 = \langle \sigma 1_S; \sigma 2_{S'}, \{\sigma 1_{\omega,s}; \sigma 2_{\sigma 1^*_S(\omega),\sigma 1_S(s)}\}_{\omega \in S^*, s \in S} \rangle$$

Identity morphisms are those with a pair of identity functions between sorts and operation names.

Signatures together with signature morphisms form the category *SIG* of signatures of **EQ**.

For the sake of readability, we shall omit the subscripts and superscripts of σ.

Given a signature $\Sigma = \langle S, \Omega \rangle$ a (total) **Σ-algebra** consists of an S-indexed family of sets (carriers) $|A| =_{def} \{|A|_s\}_{s \in S}$ and function $f_A : |A|_{s_1} \times ... \times |A|_{s_n} \rightarrow |A|_s$ for every $f : s_1, ..., s_n \rightarrow s$ in Ω.

Let A and B be two Σ-algebras. A **Σ-homomorphism** from A to B, $h : A \rightarrow B$, is an S-indexed family of functions $\{h_s\}_{s \in S}$ between the corresponding carriers, *i.e.* $h_s : |A|_s \rightarrow |B|_s$, which is consistent with the operations in Ω, *i.e.* for all $f : s_1, ..., s_n \rightarrow s$ in Ω and $a_1 \in |A|_{s_1}, ..., a_n \in |A|_{s_n}$, $h_s(f_A(a_1, ..., a_n)) = f_B(h_{s_1}(a_1), ..., h_{s_n}(a_n))$.

Two Σ-homomorphisms $h1 : A \rightarrow B$ and $h2 : B \rightarrow C$ compose by composing the functions they consist of, *i.e.* $h1; h2 = \{h1_s; h2_s\}_{s \in S}$. Identity Σ-

homomorphisms $h : A \to A$ consist of an identity function h_s for each carrier $|A|_s$.

The algebras over a signature Σ together with their Σ-homomorphisms form a category $Alg(\Sigma)$.

Given a signature morphism $\sigma : \Sigma \to \Sigma'$, the **$\sigma$-reduct** of a Σ'-algebra A' is defined as the Σ-algebra $A'|_\sigma$ such that for all sorts s in Σ, $|(A'|_\sigma)|_s =_{def} |A'|_{\sigma(s)}$ and for all $f : s_1, ..., s_n \to s$ in Σ, $f_{A'|_\sigma} = \sigma(f)_{A'}$.

We can also define the σ-reduct of a Σ'-homomorphism $h' : A' \to B'$, where A' and B' are Σ'-algebras, as the Σ-homomorphism $h'|_\sigma : A'|_\sigma \to B'|_\sigma$ with $(h'|_\sigma)_s =_{def} h'_{\sigma(s)}$ for all sorts s in Σ.

With this, $\_|_\sigma$ turns out to be a functor from $Alg(\Sigma')$ to $Alg(\Sigma)$, and therefore we can define $Alg : SIG \to Cat^{op}$ as the functor mapping each signature Σ to a category of models $Alg(\Sigma)$ and each signature morphism $\sigma : \Sigma \to \Sigma'$ to $\_|_\sigma : Alg(\Sigma') \to Alg(\Sigma)$, that is, Alg is the model functor in **EQ**.

Given a signature $\Sigma = \langle S, \Omega \rangle$, a **$\Sigma$-term** t of sort s over an S-indexed set of variables X disjoint from Ω, is either a variable $x \in X_s$ or the composition of a function name $f : s_1, ..., s_n \to s \in \Omega$[3] with Σ-terms $t_1, ..., t_n$ over X of sort $s_1, ..., s_n$ respectively.

The set of terms over each sort plus the rules for composing smaller terms to form bigger terms constitute a Σ-algebra, the so-called **term algebra**, $T_\Sigma(X)$. The carrier of a sort s in $T_\Sigma(X)$ consists of all Σ-terms of sort s over X.

A **valuation** of X over a Σ-algebra A is a Σ-homomorphism $\nu : T_\Sigma(X) \to A$.

---

[3]Note that f is a constant for $n = 0$.

Fact 2.3.1 *A valuation ν of X over A is uniquely defined by the values assigned to each variable in X and, if X is empty, ν over A is unique.*

The latter is the case because a Σ-term without variables can be uniquely evaluated in a Σ-algebra A by substituting each constant and function name by its value in A, thus there is a unique Σ-homomorphism $\nu : T_\Sigma(\emptyset) \rightarrow A$; *i.e.* $T_\Sigma(\emptyset)$, usually written T_Σ, is an initial object in the category $Alg(\Sigma)$. Analogously for terms with variables, the value of a term over X in an algebra A is uniquely determined by the values assigned to the variables.

Final objects in $Alg(\Sigma)$ are algebras with each carrier being a singleton. In this case, all functions and constants have a unique possible interpretation and therefore, there is a unique Σ-homomorphism from an arbitrary Σ-algebra A to a final Σ-algebra $\perp_\Sigma$, $h : A \rightarrow \perp_\Sigma$.

Given a signature $\Sigma = \langle S, \Omega \rangle$, a **$\Sigma$-equation** consists of an S-indexed set of variables X and a pair of Σ-terms over X of the same sort $s \in S$, $t1, t2 \in |T_\Sigma(X)|_s$, and it is normally denoted by $\forall X.\ t1 = t2$. A Σ-algebra A **satisfies** a Σ-equation, $A \models \forall X.\ t1 = t2$, if $t1$ and $t2$ evaluate to the same value, $\nu(t1) = \nu(t2)$ (written also $A, \nu \models t1 = t2$), for all valuations $\nu : T_\Sigma(X) \rightarrow A$.

Given a signature morphism $\sigma : \Sigma \rightarrow \Sigma'$, the translation of a Σ-term t over X according to σ is a Σ'-term over $\sigma(X)$, $\sigma(t)$, where $\sigma(X)$ is a S'-indexed set of variables such that, for all $s \in S$

$$x_s{}^{4} \in \sigma(X)_{\sigma(s)} \text{ iff } x \in X_s$$

and $\sigma(X)_{s'}$ is empty for those sorts $s' \in S'$ which are not the image of any sort in S. Then, $\sigma(t)$ is the Σ'-term over $\sigma(X)$ resulting from the substitution

---

[4]Indices prevent variables of different sorts with the same name in X from being confused in $\sigma(X)$.

s formulation is incorrect. See [GB 90] for the correct definition venting two variables of different sorts to have the same name.

of all function names f in t by $\sigma(f)$ and variables $x \in X_s$ by $x_s \in \sigma(X)_{\sigma(s)}$. Finally, the **translation** of the Σ-equation $\forall X.\ t1 = t2$ along σ is the Σ'-equation $\forall X'.\ \sigma(t1) = \sigma(t2)$ where X' is $\sigma(X)$.

The functor *Eq* mapping each signature Σ to the set of Σ-equations and each signature morphism σ to the translation function between equations, is the sentence functor $Eq : SIG \rightarrow Set$ of **EQ**. And each satisfaction relation $\models_\Sigma$ is as defined above between Σ-algebras and Σ-equations.

Fact 2.3.2 (GB 84) *The satisfaction condition holds in* **EQ**.

It is the case that *SIG* has finite co-limits and *Alg* preserves finite co-limits, therefore the amalgamation lemma holds in **EQ**.

Moreover, the institution **EQ** is equipped with a sound and complete inference system $\vdash^{\mathbf{EQ}}$. This is called multi-sorted equational logic and it has been extensively used in algebraic specification (*c.f.* [GM 81, EM 85]).

Some more algebraic background

Given a set of Σ'-equations *eq*, we can define $\mathbf{Mod}[A_{eq} \epsilon_{\Sigma'}]$ to be the full subcategory of $Alg(\Sigma')$ composed of those algebras satisfying the equations in *eq*. It is a well-known result of universal algebra (see [GTW 76]) that:

Fact 2.3.3 *For every morphism $\sigma : \Sigma \rightarrow \Sigma'$ and set of Σ'-equations eq, the left adjoint $Free_\sigma^{eq} : Alg(\Sigma) \rightarrow \mathbf{Mod}[A_{eq} \epsilon_{\Sigma'}]$ of the reduct functor $\_|_\sigma : \mathbf{Mod}[A_{eq} \epsilon_{\Sigma'}] \rightarrow Alg(\Sigma)$ exists.*

In particular, considering Σ to be an empty signature, $Free_\sigma^{eq}$ can only be applied to the empty algebra and it produces an **initial Σ'-algebra** (unique up to isomorphism) in the category $\mathbf{Mod}[A_{eq} \epsilon_{\Sigma'}]$ of Σ'-algebras which satisfy *eq*.

A **subalgebra** B of a Σ-algebra A is another Σ-algebra such that for every sort s in Σ, $|B|_s \subseteq |A|_s$ and for all $f : s_1, ..., s_n \rightarrow s$ in Σ and $b_1 \in |B|_{s_1}, ..., b_n \in |B|_{s_n}$, $f_A(b_1, ..., b_n) = f_B(b_1, ..., b_n)$.

We say that a subalgebra of a Σ-algebra A is **reachable on sort** s if it differs from A only on $|A|_s$, and it contains no proper subalgebra which differs only on $|A|_s$. Similarly reachability can be defined with respect to a set of sorts.

Fact 2.3.4 *For all $\langle S, \Omega \rangle$-algebras A and set of sorts $G \subseteq S$ there is a unique reachable subalgebra $Reach_G(A)$ of A on sorts G.*

Taking all carriers of sorts not in G as identical to the corresponding carriers in A, we can construct the subalgebra $Reach_G(A)$ taking as carriers for sorts in G only those values in A which are finitely generated from constants and values of sorts not in G. By construction, all values in $Reach_G(A)$ should be included in any subalgebra of A which respects carriers of sorts not in G; hence uniqueness.

Intuitively, $Reach_G$ removes from each carrier of each sort in G those values which can not be computed (reached) using the operations in Ω and the values of the sorts not in G. Often, these non-reachable values are called "junk".

The definition of reachable algebra on a set of sorts is clearly related to the general notion of *minimal* model. Consider a category of algebras $Alg(\langle S, \Omega \rangle)$ and a set of distinguished sorts $G \subseteq S$, then the models of a well-formed specification $M_{\langle \sigma, id_{\langle S,\Omega \rangle} \rangle} SP$ where $\sigma : \langle S \setminus G, \Omega' \rangle \hookrightarrow \langle S, \Omega \rangle$ with an arbitrary set of function names $\Omega' \subseteq \Omega$ are:

$$\{A \in Mod[SP] \mid A \text{ is reachable on sorts } G\}$$

If a second morphism $\iota : \langle S, \Omega \rangle \rightarrow \Sigma$ is considered then the models of the specification $M_{\langle \sigma, \iota \rangle} SP$ are:

$$\{A \in Mod[SP] \mid A|_\iota \text{ is reachable on sorts } G\}$$

Alternatively to the definition above, we can also say that a $\langle S, \Omega \rangle$-algebra A is reachable on sorts $G \subseteq S$ if there is a $(S \setminus G)$-sorted set of variables X (disjoint from Ω) such that for every value $v \in |A|_r$ of a sort $r \in G$ there exists a term $t_v \in |T_{\langle S,\Omega \rangle}(X)|$ and a valuation $\nu : T_{\langle S,\Omega \rangle}(X) \to A$ such that $\nu(t_v) = v$. Because of this definition reachable algebras are also called finitely generated algebras.

Let $V/\equiv$ denote the quotient of the set V modulo the equivalence relation $\equiv \subseteq V \times V$.

A **congruence $\sim$ on a $\langle S, \Omega \rangle$-algebra** A is a S-sorted equivalence relation $\sim \subseteq |A| \times |A|$ which is consistent with the operations of A, *i.e.* for all $f : s_1, ..., s_n \to s$ in Ω, if $a_1, b_1 \in |A|_{s_1}, ..., a_n, b_n \in |A|_{s_n}$ and $a_1 \sim b_1, ..., a_n \sim b_n$ then $f_A(a_1, ..., a_n) \sim f_A(b_1, ..., b_n)$. Some well-known facts of universal algebra follow:

Fact 2.3.5 *Let $\sim$ be a congruence on a Σ-algebra A, then $A/\sim$ is a well-defined Σ-algebra, where $|A/\sim|_s = |A|_s/\sim$ and for all $f : s_1, ..., s_n \to s$ in Σ and $a_1 \in |A|_{s_1}, ..., a_n \in |A|_{s_n}$, $f_{A/\sim}([a_1], ..., [a_n]) = [f_A(a_1, ..., a_n)]$.*

Fact 2.3.6 *For any set eq of Σ-equations and Σ-algebra A there is a least congruence $\sim_A^{eq}$ such that $A/\sim_A^{eq} \models eq$.*

Constructors *FQRD*

In algebraic specifications is common to express implementation relations via the combination of some constructors [Ehr 81, EKMP 82, ST 88b]. For this use, there are some standard constructors used in equational specifications, namely, free-extension, quotient, restriction to the reachable subalgebra and derive along a signature morphism.

term	$Sig[term]$	$Mod[term]$	Context conditions
$F_\sigma^{eq}SP$	$\uparrow\sigma$	$\{Free_\sigma^{eq}(A) \mid A \in Mod[SP]\}$	$Sig[SP] = \downarrow\sigma$; $eq \subseteq Sen(\uparrow\sigma)$
$Q_{eq}SP$	$Sig[SP]$	$\{A/\sim_A^{eq} \mid A \in Mod[SP]\}$	$eq \subseteq Sen(Sig[SP])$
R_GSP	$Sig[SP]$	$\{Reach_G(A) \mid A \in Mod[SP]\}$	$G \subseteq Sorts(Sig[SP])$

Note that derive is not included since D_σ is already one of the six SBO's in ASL and therefore it enjoys institution-independent semantics in contrast to F_σ^{eq}, Q_{eq} and R_G.

These constructors are powerful enough to describe useful implementation relations [Ehr 81], they have simple model-theoretical semantics and they have been treated in the literature together with equational specifications for a long time. In this thesis, generic constructors as defined in 2.2.9 are preferred because of their institution-independent semantics and their explicit proof-theoretical significance, nevertheless *FQRD* constructors will still be considered in some sections where concrete examples are presented.

Abstraction

Although no abstraction operation has been defined for ASL above, ASL(**EQ**) is frequently extended to include an abstraction SBO which closes the class of models of a specification w.r.t. behavioural equivalence [SW 83, ST 87]. In the algebraic institution, this is often understood as defining some sorts of a specification to be *non-observable.*

The semantics of abstraction will be discussed in detail in chapter 5, both for the general case and for **EQ**.

Algebras as functors

Since early work by F.W.Lawvere [Law 63] (*cf.* [GTWW 75, Poi 86]), there has been an alternative categorical definition of an algebra.

Given an algebraic signature $\Sigma = \langle S, \Omega \rangle$, there is a category St_Σ with finite products, defined as follows:

1. objects are words $\omega \in S^*$,

2. morphisms are tuples of Σ-terms $\langle t_1, ..., t_n \rangle (x_1, ..., x_m) : s_1 ... s_m \rightarrow s'_1 ... s'_n$ where x_i is a variable of sort s_i for all $i = 1, ..., m$ and t_j is a term of sort s'_j over variables $x_1, ..., x_m$ for all $j = 1, ..., n$,

3. and composition is substitution

$$\langle t_1, ..., t_n \rangle (x_1, ..., x_m); \langle t'_1, ..., t'_p \rangle (y_1, ..., y_n)$$
$$= \langle t'_1[t_1/y_1, ..., t_n/y_n], ..., t'_p[t_1/y_1, ..., t_n/y_n] \rangle (x_1, ..., x_m)$$

Intuitively, the objects are sort names and the morphisms are the function names in Σ going from the parameter sorts to the result sort. Since morphisms must be closed under composition, all composite functions must also be included. Moreover, since functions in Σ may have no parameters (constants) or more than one parameter, sort names must be replaced by words of sort names and functions by tuples of functions.

Note that in the literature, except in [Poi 86], it is customary to consider the arrows in St_Σ reversed, that is going from the sort of the result to the sorts of the parameters. In this case St_Σ is closed under finite sums (co-products) instead of products, but this difference is irrelevant.

Now, Σ-algebras can be seen as product-preserving functors from St_Σ to the category of sets *Set*.

Theorem 2.3.7 (Law 63) *The category of Σ-algebras is isomorphic to the category of functors $F : St_\Sigma \to Set$ preserving finite products with natural transformations as morphisms.*

This functorial view of algebras will be used in chapter 5 in connection with the definition of behaviour, and also as a motivation for the category of specifications considered in chapter 7.

2.4 Derived syntax

Although the syntax of specifications has been fixed above, such a presentation of specifications is only useful for expressing inference rules or structural properties in a concise form. The actual writing of specifications becomes tedious and unreadable unless a derived language or at least a derived syntax is used.

Since its first appearance in [SW 83], *ASL* has been promoted as a kernel language upon which *real* specification languages can be defined. In this sense, the semantics of languages such as PLUSS [Gau 84, BGM 89] and Extended ML [ST 86] have been defined in terms of *ASL*. However, in the examples given in this thesis we want to use as little extra syntax as possible, that is why we shall only define a concrete syntax for a few common constructions in $ASL(\mathbf{EQ})$.

1. SP = `sorts` S {sort names}
   `operations` Ω {function names over sorts in S}
   `axioms` Φ {sentences over the signature $\langle S, \Omega \rangle$}
   `end`

   defines SP to be the specification $A_{\Phi \epsilon \langle S,\Omega \rangle}$ in $ASL(\mathbf{EQ})$ and any other institution with algebraic signatures. These specifications are commonly called *basic specifications.*

2. SP = `sorts` S
   `hidden_sorts` HS {hidden sort names}
   `operations` Ω {function names over sorts in S}
   `hidden_operations` $H\Omega$ {hidden function names over sorts $S \cup HS$}
   `axioms` Φ {sentences over $\langle S \cup HS, \Omega \cup H\Omega \rangle$}
   `end`

   defines SP to be a basic specification with a part of the signature hidden, $D_\iota A_\Phi \epsilon_{\langle S \cup HS, \Omega \cup H\Omega \rangle}$ where ι is the inclusion $\iota : \langle S, \Omega \rangle \hookrightarrow \langle S \cup HS, \Omega \cup H\Omega \rangle$.

3. Let SP' be a specification over a signature $\langle S', \Omega' \rangle$, then

   SP = `Enrich` SP' `by`
   `sorts` S
   `operations` Ω {functions names over sorts in $S \cup S'$}
   `axioms` Φ {sentences over $\langle S \cup S', \Omega \cup \Omega' \rangle$}
   `end`

   defines SP to be a specification $A_\Phi T_\iota SP'$ where ι is the inclusion of the signature $\langle S', \Omega' \rangle$ into the extended signature $\langle S \cup S', \Omega \cup \Omega' \rangle$. `Enrich` is defined for ASL over any institution having algebraic signatures.

   Enriched specifications may be written as $(A_\Phi T_\iota)SP'$ in order to emphasize each single enrichment. This can be formalized in terms of parameterized specifications as follows:

Definition 2.4.1 *An* **enrichment** *by Σ_e-axioms Φ_e w.r.t. a signature inclusion $\iota_e : \Sigma \hookrightarrow \Sigma_e$ is a parameterized specification $(\lambda X : \Sigma.\ A_{\Phi_e} T_{\iota_e} X)$ which produces a specification over Σ_e when it is applied to a specification over Σ. Enrichments are usually written as $(A_{\Phi_e} T_{\iota_e})$.*

This notation is particularly useful in the following case where two enrichments are considered. We also use this notation in institutions without algebraic signatures, where the combination of translation along a signature morphism and imposing axioms has an effect analogous to enrichment in **EQ**.

4. SP = **Enrich** SP' **by**
   Hidden {Hidden enrichment}
   sorts HS
   operations $H\Omega$
   axioms Φ_h
   in {Visible enrichment}
   sorts S
   operations Ω
   axioms Φ_v
   end

defines SP to be a specification $D_{\iota_h'}(A_{\Phi_v}\, T_{\iota_v})(A_{\Phi_h}\, T_{\iota_h})SP'$ where ι_h, ι_h' and ι_v are inclusions as follows, assuming $\langle S', \Omega' \rangle$ to be the signature of SP':

$$\iota_h : \langle S', \Omega' \rangle \hookrightarrow \langle S' \cup HS, \Omega' \cup H\Omega \rangle$$

$$\iota_v : \langle S' \cup HS, \Omega' \cup H\Omega \rangle \hookrightarrow \langle S' \cup HS \cup S, \Omega' \cup H\Omega \cup \Omega \rangle$$

$$\iota_h' : \langle S' \cup S, \Omega' \cup \Omega \rangle \hookrightarrow \langle S' \cup HS \cup S, \Omega' \cup H\Omega \cup \Omega \rangle$$

Note that enriching with the help of some auxiliary sorts and functions results in a different specification term than enriching a specification with a part of the signature hidden, which produces $D_{\iota_h'}(A_{\Phi_v \cup \Phi_h}\, T_{\iota_h;\, \iota_v})SP'$. Although both terms specify the same class of models, the former specification term is appropriate for those specification transformations defined in chapter 4.

5. Let SP' be a specification over a signature $\langle S', \Omega' \rangle$, G and S be disjoint sets of sorts included in S' and F be a subset of Ω' using sorts in $S \cup G$, then

   SP = `Reachable on` G `using` F `from` S
   SP'

   also written $M_{\langle G,F,S \rangle} SP'$, defines SP to have as models those models of SP' in which the carriers of sorts in G are finitely generated by the functions in F using the carriers of sorts in S. Formally, SP is $M_{\langle \sigma,\iota \rangle} SP'$ where ι and σ are signature inclusions as follows:

   $$\langle S, \emptyset \rangle \overset{\sigma}{\hookrightarrow} \langle S \cup G, F \rangle \overset{\iota}{\hookrightarrow} \langle S', \Omega' \rangle$$

   An $M_{\langle \sigma,\iota \rangle}$ operator in **EQ**, and in particular $M_{\langle G,F,S \rangle}$, is commonly called a **reachability constraint**. Often the same name is used in other institutions where $M_{\langle \sigma,\iota \rangle}$ is similarly related to reachable or finitely generated models.

   In chapter 6, constraints $M_{\langle \sigma,\iota \rangle}$ are treated as sentences $\ll \sigma, \iota \gg$ in an institution with constraints (Theorem 6.2.1). In particular, constraints $M_{\langle G,F,S \rangle}$ written in the form

   `Reachable on` G `using` F `from` S

   can be used as sentences in the `axioms` part of a specification.

6. SP = `Non_observable on` NO {Non-observable sorts}
   SP'

   defines SP to be a specification $T_o D_o SP$ where o is an abstraction morphism shifting sorts in NO from observable to non-observable. The definition of o and the meaning of $T_o D_o$ will be clear once abstraction morphisms and behaviours are defined in chapter 5.

2.5 Other institutions

In this thesis results at the institutional level are regarded as exceptions. Most work studies institution-independent aspects of verification while isolated sections are expressed mainly in terms of $ASL(\mathbf{EQ})$. However, other common institutions are also used.

Some results in chapter 6 and many examples through this thesis refer to institutions which enrich the algebraic institution **EQ** by adding other kinds of sentences and extending the satisfaction relation accordingly. In these cases, both the category of algebraic signatures SIG and the model functor Alg are as in **EQ**. Normally the sentences of **EQ** are enriched with conditional equations, disjunctions, negation, etc.

Technically, it is easier to define the institutions of first order logic **FOL** and first order logic with equality **FOLEQ**, then most institutions of interest are specializations of **FOLEQ** (as presented in [GB 84]). In the following a similar simplified presentation is given:

FOL

Given the category SIG of algebraic signatures, SIG_F is a category as follows:

- **Objects:** Pairs $\langle \Sigma, Q \rangle$ where $\Sigma = \langle S, \Omega \rangle$ is a standard algebraic signature, $\Sigma \in |SIG|$, and Q is a S-sorted set where each Q_s is a set of binary predicate symbols.

- **Morphisms:** Pairs $\langle \sigma, f \rangle : \langle \Sigma 1, Q1 \rangle \rightarrow \langle \Sigma 2, Q2 \rangle$ where $\sigma : \Sigma 1 \rightarrow \Sigma 2$ is an algebraic signature morphism and $f : Q1 \rightarrow Q2$ a function mapping each $r \in Q1_s$ to a predicate $f(r) \in Q2_{\sigma(s)}$ for all $s \in S$.

For any object $\Pi = \langle \Sigma, Q \rangle$ in $|SIG_F|$, we define a Π-structure $\mathcal{A}$ to consist of a Σ-algebra $\mathcal{A}_\Sigma$ and a relation $r_\mathcal{A} \subseteq |\mathcal{A}_\Sigma|_s \times |\mathcal{A}_\Sigma|_s$ for each $r \in Q_s$.

Let A and B be two Π-structures. A structure morphism $h : \mathcal{A} \to \mathcal{B}$ is a Σ-homomorphism $h : \mathcal{A}_\Sigma \to \mathcal{B}_\Sigma$ such that for all $r \in Q_s$ in Π and $(a, b) \in r_\mathcal{A}$, $(h_s(a), h_s(b)) \in r_\mathcal{B}$.

The structures over an object Π together with their structure morphisms form a category $Str(\Pi)$.

Given a morphism $\tau = \langle \sigma, f \rangle : \Pi \to \Pi'$ in SIG_F, the τ-reduct of a Π'-structure $\mathcal{A}'$ is defined as the Π-structure $\mathcal{A}'|_\tau$ such that $(\mathcal{A}'|_\tau)_\Sigma = (\mathcal{A}'_\Sigma)|_\sigma$ and $r_{\mathcal{A}'|_\tau} = f(r)_{\mathcal{A}'}$. Moreover, the τ-reduct of a Π'-structure morphism h' is the Π-structure morphism $h'|_\tau = h'|_\sigma$.

Hence $\_|_\tau$ is a functor from $Str(\Pi')$ to $Str(\Pi)$ and $Str : SIG_F \to Cat^{op}$ is the functor mapping each object Π in SIG_F to the category of structures $Str(\Pi)$ and each morphism τ in SIG_F to the reduct functor $\_|_\tau$.

The set of first order formulae over an object $\Pi = \langle \Sigma, Q \rangle$ in SIG_F and an S-sorted set of variables X is recursively defined as the smallest set $Fol(\Pi)$ such that:

1. for all $t1, t2 \in |T_\Sigma(X)|_s$ and $r \in Q_s$, $r(t1, t2)$ is in $Fol(\Pi)$.

2. `true` and `false` are in $Fol(\Pi)$.

3. $\neg F$ is in $Fol(\Pi)$, provided $F \in Fol(\Pi)$.

4. $F1 \wedge F2$ and $F1 \vee F2$ are in $Fol(\Pi)$, provided $F1, F2 \in Fol(\Pi)$.

5. $\forall x : s.\ F$ and $\exists x : s.\ F$ are in $Fol(\Pi)$, provided $F \in Fol(\Pi)$ and $x \in X_s$.

First order sentences over Π, $Fos(\Pi)$, are those formulae in $Fol(\Pi)$ without free variables, *i.e.* every variable in a sentence occurs in the scope of a quantifier which binds that variable.

Satisfaction between a first order Π-formulae and a Π-structure $\mathcal{A}$ w.r.t. a valuation $\nu : T_\Sigma(X) \to \mathcal{A}_\Sigma$ is defined as follows:

1. $\mathcal{A}, \nu \models r(t1, t2)$ iff $(\nu(t1), \nu(t2)) \in r_\mathcal{A}$.

2. $\mathcal{A}, \nu \models \texttt{true}$ and $\mathcal{A}, \nu \not\models \texttt{false}$.

3. $\mathcal{A}, \nu \models F1 \wedge F2$ iff $\mathcal{A}, \nu \models F1$ and $\mathcal{A}, \nu \models F2$.

4. $\mathcal{A}, \nu \models F1 \vee F2$ iff $\mathcal{A}, \nu \models F1$ or $\mathcal{A}, \nu \models F2$.

5. $\mathcal{A}, \nu \models \neg F$ iff $\mathcal{A}, \nu \not\models F$.

6. $\mathcal{A}, \nu \models \forall x : s.\ F$ iff for all $v \in |\mathcal{A}_\Sigma|_s$, $\mathcal{A}, \nu_{(x,v)} \models F$.

7. $\mathcal{A}, \nu \models \exists x : s.\ F$ iff there exists a value $v \in |\mathcal{A}_\Sigma|_s$, $\mathcal{A}, \nu_{(x,v)} \models F$.

where $\nu_{(x,v)} : T_\Sigma(X) \to \mathcal{A}_\Sigma$ is the valuation which maps x to v and the rest of the variables $x' \in X$ to $\nu(x')$.

Satisfaction between a first order Π-sentence φ and a Π-structure $\mathcal{A}$ is defined as follows:

$$\mathcal{A} \models \varphi \overset{\text{def}}{\iff} \mathcal{A}, \nu_i \models \varphi$$

where ν_i is the unique valuation $\nu_i : T_\Sigma \to \mathcal{A}_\Sigma$.

First order sentences can be translated along a morphism τ in SIG_F by translating the terms and variables involved in each sentence, analogously to the way that terms and variables in equations are translated along algebraic signature morphisms. These translations preserve satisfaction and therefore, SIG_F, Str, Fos and $\models$ form an institution.

We shall refer to this institution as the institution of first order logic, denoted **FOL**. Compared to the standard institution of multisorted first order logic (see [GB 84]), this is a simplified version restricting predicates to be binary and with arguments of the same sort.

First order logic with equality, **FOLEQ**, is another institution similar to **FOL** but with a distinguished infix predicate $=_s$ over each sort s in all signatures. This predicate is called *equality* and it must be preserved by signature morphisms, $\tau(=_s) = =_{\tau(s)}$. Moreover, every structure in **FOLEQ** must interpret the equality predicate as the identity relation.

If we restrict the predicates of a **FOLEQ** structure to only the distinguished equality predicate, structures and first order signatures are in a one-to-one correspondence with algebras and algebraic signatures respectively. In the following, we shall use the term **ALG** for the institution whose signatures are algebraic signatures, models are algebras, sentences are first order sentences using a single binary predicate $=$ over each sort and satisfaction is as in **FOLEQ**, *i.e.* $=$ must be interpreted as identity on each sort. In fact, taking **ALG** and limiting the logical connectives used in first order sentences we can get institutions such as conditional equational logic or equational logic.

Most examples, unless limitation to **EQ** is necessary, are given in **ALG** and the derived syntax defined in section 2.4 for *ASL*(**EQ**) is used for *ASL*(**ALG**). Occasionally, some examples in sections 5.4.2 and 6.4.3 use **FOL**.

2.6 Entailment in *ASL*

ASL, like any other specification language, needs a notion of implementation or refinement by means of which we can state that one specification refines/implements another specification. This question has been deeply studied in [ST 88b].

Here, we choose a notation closer to logic and say that $SP2$ **entails** $SP1$, written $SP2 \models SP1$, if $Sig[SP2] = Sig[SP1]$ and $Mod[SP2] \subseteq Mod[SP1]$. Including constructors and abstraction in our specification language allows complex relations of implementation such as the constructor and abstractor implementations described in [ST 88b] to be viewed as entailments. For example, $SP2 \models SP1$ iff $SP1 \rightsquigarrow SP2$ in the notation of [ST 88a].

This thesis is mostly dedicated to finding an inference system for entailment. Since this task depends critically on the complexity of the structure of the **antecedent** $SP2$ and the **consequent** $SP1$, we shall increase their complexities step by step. For example:

- $ASL \models ASL$ refers to entailments where the antecedent and the consequent are written using the six basic SBO's. This corresponds to so-called refinements in [ST 88b] and implementations in [SW 83].

- $k{+}ASL \models ASL$ refers to entailments where the antecedent can make use of the SBO's in ASL plus the constructors (generic constructors $\langle P, \Lambda \rangle$ related to a programming language and/or $FQRD$ constructors in $ASL(\mathbf{EQ})$), whereas the consequent can only use the six basic SBO's. This case includes constructor implementations as in [ST 88b].

- $k{+}ASL \models \alpha + ASL$ refers to entailments as in the previous case but allowing the consequent to include abstraction. This case includes abstractor implementations as in [ST 88b].

- $ASL \models A$ refers to entailments where the consequent is a set of sentences (a specification using only A_Φ and ϵ_Σ as specification building operations). This refers to theorem proving in ASL; in particular $SP \models \varphi$ is a special case where the consequent is $A_{\{\varphi\}}\epsilon_\Sigma$ for a certain signature Σ.

- $DATU \models ATU$ refers to entailments whose antecedent uses $D_\sigma, A_\Phi, T_\sigma$, U and ϵ_Σ whereas the consequent can only use A_Φ, T_σ U and ϵ_Σ.

At the end of the thesis there is an index referring to the different entailments considered and where they are treated. In general, this thesis progresses from the easy to the difficult cases.

Chapter 3

Theorem proving

3.1 Introduction

Proving theorems in the context of specifications can have two purposes. In the first place, proving that all models of a specification satisfy a certain property helps us to understand what the specification means. This is especially useful in specification design when we want to check that our specifications mean what we expect. Consider for example a specification of sets in terms of emptyset, singleton and union, as in figure 3–1. Taking the finitely generated sets (*Fin_Set*) we obtain a specification of finite sets; however, this might not be clear to the user.

Alternatively, sets can be specified in terms of $\emptyset$ and `insert` as if they were lists with two extra axioms identifying those lists which differ in the order of the elements or the number of occurrences of an element (commutativity and idempotence of insertion). We may wonder if in the specification *Fin_Set*, given above, an operation `insert` defined as

$\forall$ e:elem; S:set. insert(e, S) = {e} $\cup$ S

satisfies properties such as commutativity and idempotence. If this is the case we would feel more confident about our specification *Fin_Set*.

```
Set =       sorts elem, set
            operations
              ∅:  set
              _∪_:  set, set -> set
              {_}:  elem -> set
            axioms
              ∀ S:set. S ∪ ∅ = S
              ∀ S:set. S ∪ S = S
              ∀ S1, S2:set. S1 ∪ S2 = S2 ∪ S1
              ∀ S1, S2, S3:set. (S1 ∪ S2) ∪ S3 = S1 ∪ (S2 ∪ S3)
            end

Fin_Set =   Reachable on {set} using {∅, _∪_, {_}} from {elem}
              Set
```

Figure 3–1: Sets of elements

In the second place, theorem proving plays an important role in proving entailments between specifications and, therefore, in proving correctness of implementations.

In order to prove a specification entailment correct it must be proven that all the properties required in the consequent are satisfied by the antecedent. These properties are not exclusively axioms, they may include reachability constraints and other kinds of requirements depending on the specification language considered. Nevertheless, axioms are going to be an important part of the requirements and therefore, a large amount of theorem proving is commonly involved in proofs of correctness of entailments.

Consider for example the specifications *Set* and *Fin_Set* and an implementation by lists. If we take *Set*, the specification (consequent) is just a collection of axioms and correctness can be proven by proving each axiom in *Set* to be a theorem of

the implementation (antecedent). If we impose a reachability constraint on *Set*, we obtain *Fin_Set* which cannot be reduced to a collection of axioms (not even an infinite one); however, any proof of correctness has to include a proof that the axioms in *Set* hold in the implementation independently of how the reachability constraint is verified.

Because of the double use of theorem proving, we are not only interested in how to prove theorems from specifications but also to which extent a specification (*consequent*) can be reduced to a collection of axioms, which could then be shown to be a consequence of another specification, the *antecedent.* This idea is usually formalized in terms of reduction rules which transform structured specifications into plain lists of axioms.

In this chapter, we are concerned with theorem proving for $k{+}ASL$; in other words, we restrict the general problem $SP2 \models SP1$ to the case where $SP2$ is a specification in the language $k{+}ASL$ and the consequent $SP1$ is a set of axioms, *i.e.* has the form $A_{\Phi \epsilon \Sigma}$.

In the following sections inference rules for the different SBO's are given to allow us to infer theorems from specifications in $k{+}ASL$. Some reduction rules will also be given; however, it is not necessary to give many of these explicitly since the definition of an exact inference rule (see 3.3.11 below) characterizes inference rules which can be used to transform structured specifications into equivalent collections of axioms.

3.2 Proving $ATU \models A$

In this section we focus on antecedents in ATU. ATU is the simplest structured specification language since all specifications can be reduced to an equivalent collection of axioms, *i.e.* a specification of the form $A_\Phi \epsilon_\Sigma$; in this sense it can be said that ATU is equivalent to A.

The reduction can be easily done by repeatedly applying the following equivalences from left to right:

$T_\sigma A_\Phi \epsilon_\Sigma = A_{\sigma(\Phi)}\epsilon_{\Sigma'}$ for $\sigma : \Sigma \to \Sigma'$

$T_\sigma \epsilon_\Sigma = \epsilon_{\Sigma'}$ for $\sigma : \Sigma \to \Sigma'$

$A_{\Phi 1}\epsilon_\Sigma \ U \ A_{\Phi 2}\epsilon_\Sigma = A_{\Phi 1 \cup \Phi 2}\epsilon_\Sigma$

$\epsilon_\Sigma \ U \ SP = SP$

$A_{\Phi 1} A_{\Phi 2}\epsilon_\Sigma = A_{\Phi 1 \cup \Phi 2}\epsilon_\Sigma$

$SP \ U \ \epsilon_\Sigma = SP$

This fact has been presented in the literature under different names. These equivalences are considered as definitions of T and U in presentation semantics [EM 85][1] or in module algebra [BHK 86] but, in specification languages with model-oriented semantics such as ASL, they take the form of equivalences. Specifications of the form $A_\Phi \epsilon_\Sigma$ are called **flat specification** and the reduction from arbitrary structured specifications to flat specifications is called **flattening**.

The equivalence between ATU and A allows us to reduce theorem proving over ATU to theorem proving over A, transferring the inference to the underlying logic since $A_{\Phi 1}\epsilon_\Sigma \models A_{\Phi 2}\epsilon_\Sigma$ holds in $ASL(\mathbf{I})$ if and only if $\Phi 1 \models_\Sigma \Phi 2$ with the satisfaction relation $\models$ of $\mathbf{I}$. Hence, we can use the inference system $\vdash^{\mathbf{I}}$ available for that institution.

---

[1] Called *first level semantics* in [EM 85].

The structure provided by T and U can be removed before starting to prove theorems. However, the structure may be useful in guiding the theorem prover towards the premises needed in the proof of a theorem (see [SB 83] for details). For this reason, although we are glad to know that ATU-specifications can be reduced to A-specifications, we shall not always do that reduction in our proofs. We prefer to give an inference system strong enough to perform such reductions if the user wants but he can also ignore them if it is convenient.

This feature of T and U can be used for solving $ATU \models A$ but also for solving $ATU \models ATU$. This has no analogy in the other SBO's. Neither M_ι, D_σ nor abstraction can be reduced (eliminated) in a language of presentations; that is why verifying specification entailments involving them is difficult, particularly if M_ι, D_σ or abstraction appear in the consequent (see chapters 6, 4 and 5 respectively).

3.3 Proving $ASL \models A$

In this section we are concerned with proving theorems from ASL specifications. Some research in [ST 88a] proved the following inference rules to be sound:

$$\frac{SP1 \vdash \varphi}{SP1\ U\ SP2 \vdash \varphi} \qquad \frac{SP2 \vdash \varphi}{SP1\ U\ SP2 \vdash \varphi}$$

$$\frac{SP \vdash \varphi}{T_\sigma SP \vdash \sigma(\varphi)} \qquad \frac{SP \vdash \sigma(\varphi)}{D_\sigma SP \vdash \varphi} \qquad \frac{SP \vdash \varphi}{M_{(\sigma,\iota)} SP \vdash \varphi}$$

The main task of this section is to complete this list of rules and to provide a characterization of them in order to know how good they are and, in particular, if they give rise to reductions of specifications to presentations. First, we give some general definitions about how a theory relates to a class of models and, analogously, how an inference rule relates to a specification building operation.

3.3.1 Theories versus model classes

In general, a set of sentences Φ denotes the class of models satisfying Φ but some classes of models cannot be represented by a set of sentences and, in particular, this turns out to be the case for the model classes of some specifications in $ASL(\mathbf{EQ})$. In this section we study the possible relations between a class of Σ-models Γ and a theory Δ over Σ which tries to *capture* the properties of Γ in an institution $\mathbf{I} = \langle SIG, Mod, Sen, \models \rangle$.

Definition 3.3.1 *Given a class of Σ-models Γ, we define $Th[\Gamma]$ as*

$$Th[\Gamma] = \{\varphi \in Sen(\Sigma) \mid \Gamma \models_\Sigma \varphi\}$$

Given a set of Σ-sentences Φ, we define $Mod[\Phi]$ as

$$Mod[\Phi] = \{M \in |Mod(\Sigma)| \mid M \models_\Sigma \Phi\}$$

Then, the **closure** *of a set of Σ-sentences Φ under semantic entailment is the set of Σ-sentences $Cl(\Phi) = Th[Mod[\Phi]]$.*

A **theory** *Δ over Σ is a set of Σ-sentences closed under semantic entailment; i.e. $Cl(\Delta) = \Delta$.*

Corollary 3.3.2 *For every two theories Δ_1 and Δ_2, satisfaction between theories, $\Delta_1 \models_\Sigma \Delta_2$, is equivalent to theory inclusion, $\Delta_1 \supseteq \Delta_2$.*

The operators *Th* and *Mod* on classes of models and sets of sentences form a Galois connection [BG 80].

In the following we define some relations between a class of models and a theory which tries to *capture* its properties.

Definition 3.3.3 *Given a class of Σ-models Γ and a theory Δ over Σ,*

Δ *is* **sound** *w.r.t.* Γ *if* $\Delta \subseteq Th[\Gamma]$.

Δ *is* **complete** *w.r.t.* Γ *if* $\Delta \supseteq Th[\Gamma]$.

Δ *is* **M-sound** *w.r.t.* Γ *if* $Mod[\Delta] \supseteq \Gamma$.

Δ *is* **M-complete** *w.r.t.* Γ *if* $Mod[\Delta] \subseteq \Gamma$.

When the inclusions are proper we use the terms strictly sound, strictly complete, and so on.

Δ *is* **exact** *w.r.t.* Γ *if* $Mod[\Delta] = \Gamma$.

Definition 3.3.4 *Given a specification SP and a theory Δ over $Sig[SP]$, Δ is* **sound (complete, M-sound, M-complete or exact) w.r.t. to** *SP if Δ is sound (complete, M-sound, M-complete or exact) w.r.t. $Mod[SP]$.*

These definitions are related by the following propositions.

Proposition 3.3.5 *Δ is sound w.r.t. Γ iff Δ is M-sound w.r.t. Γ.*

Proposition 3.3.6 *If Δ is M-complete w.r.t. Γ then Δ is complete w.r.t. Γ.*

Corollary 3.3.7 *If Δ is exact w.r.t. Γ then Δ is sound and complete w.r.t. Γ.*

These follow from standard properties of Galois connections and from the definition of theory. For example:

Proof of Proposition 3.3.5 If Δ is sound w.r.t. Γ then $\Delta \subseteq Th[\Gamma]$. Applying *Mod* on both sides gives $Mod[\Delta] \supseteq Mod[Th[\Gamma]]$, and recalling that $Mod[Th[\Gamma]] \supseteq \Gamma$ we get $Mod[\Delta] \supseteq \Gamma$ which is the definition of M-sound.

If Δ is M-sound w.r.t. Γ then $Mod[\Delta] \supseteq \Gamma$. Applying *Th* on both sides, and recalling that $\Delta \subseteq Th[Mod[\Delta]]$ we get $\Delta \subseteq Th[\Gamma]$ which is the definition of sound. □

Note that proposition 3.3.5 is a consequence of the Galois connection since $\Gamma \subseteq Mod[Th[\Gamma]]$ and $\Phi \subseteq Th[Mod[\Phi]]$ for arbitrary classes of models Γ and sets of sentences Φ. The inclusion in the opposite direction does not always hold, *i.e.* there are generally more models in the closure $Mod[Th[\Gamma]]$ than in Γ. However, since theories are already closed under entailment $Th[Mod[\Delta]] \subseteq \Delta$ holds, hence proposition 3.3.6.

As a result of the difference between complete and M-complete we find that a theory Δ can relate to a class of models Γ in six different possible ways: Δ can be exact; sound and complete but not exact; sound but not complete; M-complete but not sound; complete but neither M-complete nor sound; or not related to Γ. But not all these cases can occur for the same class of models.

Definition 3.3.8 *A class of models* Γ *is axiomatizable if* $\Gamma = Mod[Th[\Gamma]]$.

Proposition 3.3.9 *A class of models* Γ *is axiomatizable iff every complete theory w.r.t.* Γ *is also M-complete w.r.t.* Γ.

Proof Assume every theory Δ complete w.r.t. Γ is also M-complete w.r.t. Γ. Take Δ to be $Th[\Gamma]$. Trivially, $Th[\Gamma] \subseteq Th[\Gamma]$, hence $Th[\Gamma]$ is complete w.r.t. Γ. By the assumption, $Th[\Gamma]$ must also be M-complete w.r.t. Γ, *i.e.* $Mod[Th[\Gamma]] \subseteq \Gamma$. Finally, since by definition of *Th* and *Mod* it holds that $\Gamma \subseteq Mod[Th[\Gamma]]$ we conclude that $Mod[Th[\Gamma]] = \Gamma$.

In the other direction, suppose Γ is axiomatizable. If Δ is a complete theory w.r.t. Γ then $\Delta \supseteq Th[\Gamma]$. Applying *Mod* on both sides, $Mod[\Delta] \subseteq Mod[Th[\Gamma]] = \Gamma$, hence Δ is also M-complete w.r.t. Γ. □

Now we can classify theories again. For an axiomatizable class of models a theory can be exact, sound (and M-sound) but not complete, complete (and M-complete) but not sound, or not related.

On the other hand, if a class of models is not axiomatizable all the relations are possible except the exact one.

3.3.2 Inference rules versus SBO's

Inference rules are rules which allow us to infer a theory from another theory.

†

Definition 3.3.10 *An* **inference rule** Λ *from* Σ_1 *to* Σ_2 *is a binary relation between sets of* Σ_1*-sentences and* Σ_2*-sentences.*

The **theory** Δ_{inf} **inferred** *from a theory* Δ *using* Λ *is* $\Delta_{inf} = Cl(\{\varphi | \exists \Phi_1 \subseteq \Delta.\ (\Phi_1, \varphi) \in \Lambda\})$.

Rules such as

$$\frac{SP \vdash \varphi}{T_\sigma SP \vdash \sigma(\varphi)}$$

used in [ST 88a] have an immediate interpretation as inference rules in this sense:

$$\Lambda = \{(\{\varphi\}, \sigma(\varphi)) \mid \varphi \text{ is a } Sig[SP]\text{-sentence}\}$$

Similarly, expressions such as $\xi SP \vdash \varphi$ (*provided* $P(\xi, \varphi)$) denote inference rules. In this case, the inferred sentences φ are independent of the specification to which ξ is applied. The inference rule denoted by such an expression is

$$\Lambda = \{(\emptyset, \varphi) \mid P(\xi, \varphi)\}$$

For example, $A_\Phi SP \vdash \varphi$ (*if* $\varphi \in \Phi$) denotes an inference rule whose inferred theory is the closure of Φ.

Note that inference rules infer *theories*, therefore, we do not consider the actual sentences that a rule can deliver but only their closure. If we say that a rule

$$\frac{SP \vdash f(\varphi)}{\xi SP \vdash g(\varphi)}$$

e definition of "inference rule" is non-standard. Basically, it captures sole aspect of an inference rule which is of interest in the lowing results.

is complete, this means, in practice, that it is complete when it is used together with rules such as

$$\frac{SP \vdash \varphi 1 \quad SP \vdash \varphi 2}{SP \vdash \varphi} \ (\varphi 1, \varphi 2 \vdash^{\mathbf{I}} \varphi)$$

for a sound and complete inference system $\vdash^{\mathbf{I}}$.

Now, we can extend the definitions given for theories w.r.t. specifications to definitions for inference rules w.r.t. SBO's:

Definition 3.3.11 *Given a unary SBO ξ we say that:*

An inference rule Λ is **sound** *w.r.t. ξ, if whenever it is applied to a theory Δ which is sound w.r.t. a specification SP (i.e. w.r.t. Mod[SP]), it yields a sound theory w.r.t. ξSP (i.e. w.r.t. Mod[ξSP]).*

An inference rule Λ is **M-complete** *w.r.t. ξ, if whenever it is applied to a theory Δ which is M-complete w.r.t. a specification SP, it yields an M-complete theory w.r.t. ξSP.*

An inference rule Λ is **complete** *w.r.t. ξ, if whenever it is applied to a theory Δ which is complete w.r.t. a specification SP, it yields a complete theory w.r.t. ξSP.*

An inference rule Λ is **exact** *w.r.t. ξ, if whenever it is applied to a theory Δ which is exact w.r.t. a specification SP, it yields an exact theory w.r.t. ξSP (equivalently, Λ is exact w.r.t. ξ if Λ is sound and M-complete w.r.t. ξ).*

For binary or n-ary SBO's we can easily extend the definitions of inference rule and inferred theory in order to infer a theory from several theories. Then, the definition of soundness (M-completeness, completeness or exactness) of the inference rule requires sound (M-complete, complete or exact) theories for each of the arguments.

With reference to the presentation of an inference rule, it is worth noticing that the union of inference rules between the same signatures is also an inference rule. Therefore, we can also characterize collections of inference rules as sound, complete, M-complete or exact, meaning that the union of them is sound, complete, M-complete or exact.

The inference rules given in [ST 88a] for U, T_σ, D_σ and $M_{\langle\sigma,\iota\rangle}$, plus some others for A_Φ can be characterized as follows:

Theorem 3.3.12

$\dfrac{SP1 \vdash \varphi}{SP1\ U\ SP2 \vdash \varphi}$ $\quad$ $\dfrac{SP2 \vdash \varphi}{SP1\ U\ SP2 \vdash \varphi}$ $\quad$ *Sound and M-complete (exact).*

$\dfrac{SP \vdash \varphi}{T_\sigma SP \vdash \sigma(\varphi)}$ $\quad$ *Sound and M-complete (exact).*

$\dfrac{SP \vdash \sigma(\varphi)}{D_\sigma SP \vdash \varphi}$ $\quad$ *Sound and complete.*

$\dfrac{SP \vdash \varphi}{M_{\langle\sigma,\iota\rangle} SP \vdash \varphi}$ $\quad$ *Sound.*

$\dfrac{SP \vdash \varphi}{A_\Phi SP \vdash \varphi}$ $\quad$ $A_\Phi SP \vdash \varphi$ *(if* $\varphi \in \Phi$*)* $\quad$ *Sound and M-complete (exact).*

Proof

- $\dfrac{SP1 \vdash \varphi}{SP1\ U\ SP2 \vdash \varphi}$ $\quad$ $\dfrac{SP2 \vdash \varphi}{SP1\ U\ SP2 \vdash \varphi}$ is a sound and M-complete inference rule w.r.t. U.

Let $\Delta 1$ and $\Delta 2$ be M-complete theories w.r.t. $SP1$ and $SP2$, *i.e.* $Mod[\Delta i] \subseteq Mod[SPi]$ for $i \in \{1,2\}$. The inferred theory is $\Delta_{inf} = Cl(\Delta 1 \cup \Delta 2)$. Then, by monotonicity of the intersection, $Mod[\Delta 1] \cap Mod[\Delta 2] \subseteq Mod[SP1] \cap Mod[SP2] = Mod[SP1\ U\ SP2]$, and by the definition of the models of a theory

$$Mod[\Delta 1] \cap Mod[\Delta 2] = Mod[\Delta 1 \cup \Delta 2] = Mod[Cl(\Delta 1 \cup \Delta 2)]$$

We conclude that Δ_{inf} is an M-complete theory for $SP1\ U\ SP2$.

Analogously, and considering the inclusion in the opposite way the inference rule can be proved M-sound and therefore sound.

- $\dfrac{SP \vdash \varphi}{T_\sigma SP \vdash \sigma(\varphi)}$ is a sound and M-complete inference rule w.r.t. T_σ.

Given an M-complete theory Δ for SP, the inferred theory has models $Mod[\Delta_{inf}] = \{A \in |Mod(\uparrow \sigma)| \mid A \models Cl(\sigma(\Delta))\}$ or, equivalently, $\{A \in |Mod(\uparrow\sigma)| \mid A \models \sigma(\Delta)\}$ which by the satisfaction condition is

$$\{A \in |Mod(\uparrow\sigma)| \mid A|_\sigma \models \Delta\}$$

Finally, by definition of the models of a theory, the same class of models can be expressed as $\{A \in |Mod(\uparrow\sigma)| \mid A|_\sigma \in Mod[\Delta]\}$ which, by M-completeness of Δ, is included in $\{A \in |Mod(\uparrow\sigma)| \ |A|_\sigma \in Mod[SP]\} = Mod[T_\sigma SP]$. We conclude that Δ_{inf} is an M-complete theory for $T_\sigma SP$.

Analogously, and considering the inclusion in the opposite way the inference rule can be proved M-sound and therefore sound.

- $\dfrac{SP \vdash \sigma(\varphi)}{D_\sigma SP \vdash \varphi}$ is a sound and complete inference rule w.r.t. D_σ.

Let be Δ a sound theory w.r.t. SP. The inferred theory is $\Delta_{inf} = Cl(\sigma^{-1}\Delta)$ where $\sigma^{-1}\Delta = \{\varphi_1 \in Sen(\downarrow\sigma) \mid \sigma(\varphi_1) \in \Delta\}$. Then,

$$
\begin{aligned}
Mod[SP] &\subseteq Mod[\Delta] && \text{since } \Delta \text{ is sound w.r.t. } SP \\
&\subseteq Mod[\sigma(\sigma^{-1}\Delta)] && \text{by definition of } \sigma^{-1}\Delta \\
&= \{A \in |Mod(\uparrow\sigma)| \mid |A \models \sigma(\sigma^{-1}\Delta)\} \\
&= \{A \in |Mod(\uparrow\sigma)| \mid A|_\sigma \models \sigma^{-1}\Delta\} && \text{by the satisfaction condition}
\end{aligned}
$$

Applying the σ-reduct functor to both sides

$$
\begin{aligned}
Mod[D_\sigma SP] &\subseteq \{A|_\sigma \mid A \in |Mod(\uparrow\sigma)| \text{ and } A|_\sigma \models \sigma^{-1}\Delta\} \\
&\subseteq \{A \in |Mod(\downarrow\sigma)| \mid A \models \sigma^{-1}\Delta\} \\
&= Mod[\sigma^{-1}\Delta] \\
&= Mod[\Delta_{inf}]
\end{aligned}
$$

therefore the rule is sound.

On the other hand, let Δ be a complete theory w.r.t. SP, $Th[Mod[SP]] \subseteq \Delta$, and consider an arbitrary sentence $\varphi \in Th[Mod[D_\sigma SP]]$. By definition of Th, every model of $D_\sigma SP$ satisfies φ. Hence, for all $M \in Mod[SP]$, $M|_\sigma \models \varphi$. Applying the satisfaction condition, $M \models \sigma(\varphi)$. Since M is an arbitrary model of SP, we can generalize to $Mod[SP] \models \sigma(\varphi)$. Hence, $\sigma(\varphi) \in Th[Mod[SP]] \subseteq \Delta$ and by definition of Δ_{inf}, $\varphi \in \Delta_{inf}$. Generalizing again, we obtain that $Th[Mod[D_\sigma SP]] \subseteq \Delta_{inf}$ as desired.

- $\dfrac{SP \vdash \varphi}{M_{\langle\sigma,\iota\rangle}SP \vdash \varphi}$ is a sound inference rule w.r.t. $M_{\langle\sigma,\iota\rangle}$.

For all sound theories Δ for SP, since M is a selector, $Mod[M_{\langle\sigma,\iota\rangle}SP] \subseteq Mod[SP] \subseteq Mod[\Delta] = Mod[\Delta_{inf}]$.

- $\dfrac{SP \vdash \varphi}{A_\Phi SP \vdash \varphi}$ $A_\Phi SP \vdash \varphi$ (if $\varphi \in \Phi$) is a sound and M-complete inference rule w.r.t. A_Φ.

For every M-complete Δ for SP, the inferred theory $\Delta_{inf} = Cl(\Delta \cup \Phi)$ has models $Mod[\Delta_{inf}] = \{A \in Mod[\Delta] | \; A \models \Phi\}$. Considering the definition of A_Φ and the M-completeness of Δ it follows that $Mod[\Delta_{inf}] \subseteq \{A \in Mod[SP] \mid A \models \Phi\} = Mod[A_\Phi SP]$.

Analogously, and considering the inclusion in the opposite way the inference rule can be proved M-sound and therefore sound.

Technically, our version of *ASL* differs from [ST 88a] in the definition of A_Φ as a SBO instead of using basic specifications. For this reason, we need an extra rule of inference for ϵ_Σ. In this case, since ϵ_Σ is a nullary SBO, we can directly present a theory instead of an inference rule.

Proposition 3.3.13 *The set of tautologies over Σ is an exact theory w.r.t. ϵ_Σ.*

Proof Trivial since all models satisfy tautologies.

The most interesting aspect of this characterization is that the completeness of the inference rule for D_σ does not entail M-completeness. This can be shown by an example from [BHK 86] which may be translated into *ASL* over the institution **ALG** as in figure 3–2.

The models of *NStNat* are the non-standard models of the natural numbers. It is well-known that there exist non-standard models of the natural numbers elementarily equivalent to the standard model, therefore the class of models denoted by *NStNat* is not axiomatizable in first order logic.

Summing up, we have a specification SP with an exact theory (SP is the specification *NStNat* before hiding `reach` and `inf`), and a non-axiomatizable specification of the form $D_\sigma SP$, the specification *NStNat*. In this situation, no exact rule for D_σ can exist since such a rule should infer an exact theory for $D_\sigma SP$ from an exact theory for SP.

```
NStNat =  sorts nat
          hidden_sorts
          operations
            0:  nat
            suc:  nat -> nat
          hidden_operations
            reach:  nat -> nat
            inf:  nat
          axioms
            ∀ x:nat. 0≠ suc(x)
            ∀ x,y:nat. (suc(x)=suc(y)) ⇒ (x=y)
            reach(0)=0
            ∀ x:nat. (reach(x)=0) ⇒ (reach(suc(x))=0)
            reach(inf)=suc(0)
          end
```

Figure 3–2: Non-standard natural numbers

In specification languages with (first order) theory-based semantics such as LARCH, the models of a specification analogous to *NStNat* include the standard model of the natural numbers against all intuition.

Looking at the inference rules given above, we may feel disappointed by the rule for $M_{\langle\sigma,\iota\rangle}$. The relation of $M_{\langle\sigma,\iota\rangle}$ to term generated models and therefore to induction seems clear, but no rule takes advantage of that. In fact, an induction rule cannot be treated at the same level as the other rules because it is not institution independent.

Induction, in contrast to the rules given above, assumes the existence of terms and therefore of sorts and algebraic-like signatures. Moreover, by its institution independent definition, $M_{\langle\sigma,\iota\rangle}$ only happens to impose finite term generation insofar as the morphisms in each category, $Mod(\Sigma)$, of models over a signature Σ are functions over the carriers of each sort. That means that our assumptions are not only on the nature of the signatures and the models but also on the morphisms between the models. Consider for example, a category of models over Σ whose objects are algebras, $|Alg(\Sigma)|$, but whose morphisms are not homomorphisms but an ordering according to the cardinality of the carriers. Then, $M_{\langle\sigma,\iota\rangle}$ does not have anything to do with term generated models or induction.

Nevertheless, we are especially interested in those cases where $M_{\langle\sigma,\iota\rangle}$ is related to induction. In the following we give the obvious rule for $M_{\langle\sigma,\iota\rangle}$ in an institution such as **FOLEQ**.

Induction rule

As mentioned in the preliminaries, most constraints $M_{\langle\sigma,\iota\rangle}$ are of the form $M_{\langle G,F,S\rangle}$ when they refer to an institution where constraints are interpreted as reachability constraints.

Proving inductive consequences has been extensively treated in the literature. We shall only give a brief account of the results.

A second order induction axiom can fully express the effect of a reachability constraint on one sort; *e.g.* in the case of arithmetic $M_{\langle\{nat\},\{0,suc\},\emptyset\rangle}$ amounts to

$$\forall P.\ (P(0) \wedge \forall x.\ (P(x) \Rightarrow P(suc(x)))) \Rightarrow (\forall x.\ P(x))$$

However, if the second order axiom is replaced by a first order schema

$$(A(0) \wedge \forall x.\ (A(x) \Rightarrow A(suc(x)))) \Rightarrow \forall x.\ A(x) \quad \textit{for all first order formulae } A$$

restricting A to be a predicate expressible in first order logic, completeness is lost by Gödel's incompleteness theorem.

This induction schema is an example of structural induction where the generated terms are ordered by the relation between a term and its immediate subterms. The idea of *translating* a reachability constraint into a structural induction schema does not rule out using other induction schemes such as complete induction:

$$\forall x.\ (\forall z.\ (z < x \Rightarrow A(z)) \Rightarrow A(x)) \Rightarrow \forall x.\ A(x) \quad \textit{for all first order formulae } A$$

which can be derived from the schema of structural induction (see [Men 71] page 126).

In general, structural induction is extensively used in computer science [Bur 69, BM 79, GG 88] but it is not always so simple as in the case of arithmetic since we must deal with multisorted structures with possibly mutually recursive generators. Frequently induction must be nested or simultaneously done on several sorts and often auxiliary predicates for the sorts of the parameters must be *guessed* by heuristics (see [BM 79] for details).

Consider for example a specification SP with sorts $\{s1, s2, s3, s4\}$ where the latter two are constrained by $M_{\langle G,F,S\rangle}$ with

$$G = \{s3, s4\} \qquad F = \{f1 : s1 \to s3, \quad f2 : s2, s3 \to s4, \quad f3 : s3, s4 \to s4\} \qquad S = \{s1, s2\}$$

In this case we can simultaneously prove predicates Q_{s3} and Q_{s4} by induction as follows:

$$\frac{\begin{array}{l} SP \vdash \forall x : s1.\ Q_{s3}(f1(x)) \\ SP \vdash \forall x : s2; y : s3.\ Q_{s3}(y) \Rightarrow Q_{s4}(f2(x, y)) \\ SP \vdash \forall y3 : s3; y4 : s4.\ (Q_{s3}(y3) \wedge Q_{s4}(y4)) \Rightarrow Q_{s4}(f3(y3, y4)) \end{array}}{M_{\langle G,F,S\rangle} SP \vdash (\forall y : s3.\ Q_{s3}(y)) \ \wedge \ (\forall y : s4.\ Q_{s4}(y))}$$

After these considerations a general structural induction rule for $M_{\langle G,F,S\rangle}$ can be defined as follows.

Proposition 3.3.14 *Given a specification SP over a set of sorts including disjoint sets S (basic sorts) and G (generated sorts), the rule*

$$\frac{SP \vdash \bigwedge_{s \in G} \bigwedge_{op \in F_s} \forall \overline{x} : S, \overline{y} : G.\ (\overline{Q}(\overline{y}) \Rightarrow Q_s(op(\overline{xy})))}{M_{\langle G,F,S\rangle} SP \vdash \bigwedge_{s \in G} \forall y : s.Q_s(y)}$$

is sound where Q is a G-indexed set of predicates, $\overline{x}$ a S-sorted sequence of variables, $\overline{y}$ a G-sorted sequence of variables, $\overline{xy}$ is the concatenation of $\overline{x}$ and $\overline{y}$, F_s those operations in F delivering a value of sort s, and $\overline{Q}$ the conjunction of the appropriate predicates in Q applied to a given G-sorted sequence of variables.

Proof We shall simultaneously prove that

$$\forall y : s.Q_s(y)$$

holds for the models of $M_{\langle G,F,S\rangle} SP$ for all $s \in G$ assuming that the premises of the rule hold for the models of SP.

Since s is a generated sort, by definition of $M_{\langle G,F,S\rangle}$ in **EQ**, every value a of sort s in G must be the value of a term $t_a \in |T_{\langle G \cup S,F\rangle}(\overline{x})|$ for an S-sorted set of

variables $\overline{x}$. Since t_a uses only the function symbols in F, we can substitute the conclusion $\forall\, y : s.Q_s(y)$ by the schema $\forall\, \overline{x} : S.\ Q_s(t(\overline{x}))$ for an arbitrary term t of sort s using only the function symbols in F.

Assuming that all the premises in the rule hold, we start structural induction on $t(\overline{x})$:

- Let the term $t(\overline{x})$ be a constant c of sort s. Then, $Q_s(c)$ holds since it is among the premises of the form $(\overline{Q(y)} \Rightarrow Q_s(op(\overline{xy})))$ for empty $\overline{x}$ and $\overline{y}$.

- Let $t(\overline{x})$ be a term $op(\overline{x_0}, t_1(\overline{x_1 y_1}), ..., t_n(\overline{x_n y_n}))$, with subterms t_1, ..., t_n of sorts s_1, ..., s_n in G, S-sorted sequences of variables $\overline{x_0}, ..., \overline{x_n}$ and G-sorted sequences of variables $\overline{y_1}, ..., \overline{y_n}$, and $op \in F$. By induction hypothesis, predicates Q hold for the respective subterms, *i.e.*

  $$\forall\, \overline{x_0}, \overline{x_1}, ..., \overline{x_n} : S.\ \overline{y_1}, ..., \overline{y_n} : G.\ Q_{s_1}(t_1(\overline{x_1 y_1})) \wedge ... \wedge Q_{s_n}(t_n(\overline{x_n y_n}))$$

  Moreover, considering the premise

  $$\forall\, \overline{x_0} : S.\ y_1 : s_1.\ ...y_n : s_n.\ (Q_{s_1}(y_1) \wedge ... \wedge Q_{s_n}(y_n)) \Rightarrow Q_s(op(\overline{x_0}, y_1, ..., y_n))$$

  in the rule, we conclude that

  $$\forall\, \overline{x_0}, \overline{x_1}, ..., \overline{x_n} : S.\ \overline{y_1}, ..., \overline{y_n} : G.\ Q_s(op(\overline{x_0}, t_1(\overline{x_1 y_1}), ..., t_n(\overline{x_n y_n})))$$

Finally, since $Q_s(t(\overline{x}))$ holds for all terms t using function symbols in F and variables in S, we conclude that

$$M_{(G,F,S)}SP \models \forall\, y : s.Q_s(y)$$

□

This rule looks very demanding since properties for generated sorts can only be proven simultaneously with a property for each generated sort. In practice,

if we are interested in proving $\forall x : s.\ P(x)$, a complete family Q with $Q_s = P$ can be chosen. In the example above, properties for $s3$ can be proven by taking $Q_{s4} = \mathit{true}$ since no value of $s3$ can be generated from the values of $s4$. However, in general other $Q_{s'}$ may be needed and theorem provers must undertake some synthesis of auxiliary predicates, see *e.g.* Boyer-Moore's theorem prover [BM 79].

Structural induction is a first order schema. In $ASL(\mathbf{EQ})$ such a schema cannot be expressed as it is above since no quantifiers or implications are available in **EQ**. In this case, structural induction has to be moved to the metalanguage.

Despite the fact that other induction schemes can be explicitly derived from structural induction, the system is bound to be incomplete due to basic results of mathematical logic. We shall call those systems which fail to be incomplete only because of induction **satisfactory systems**, in order to distinguish them from other – generally worse – incomplete systems.

3.4 Proving $k{+}ASL \models A$

In order to prove theorems from specifications with constructors, a few more inference rules are given. Generic constructors and $FQRD$-constructors are considered, however complete rules are never attained.

Generic constructors $\langle P, \Lambda \rangle$

Generic constructors come with a sound inference rule embedded in their definition:

Proposition 3.4.1 $\dfrac{SP \vdash \Phi}{\langle P, \Lambda\rangle SP \vdash \varphi}$ *provided* $(\Phi, \varphi) \in \Lambda$ *is a sound inference rule.*

Soundness is trivial by definition. The incompleteness of the rule is also clear since there is no way to know what the program P really does, apart from what is recorded in Λ.

The advantages of such constructors are methodological. Carrying Λ allows us to treat the program P as a black box: *Whatever the program P does, it will be unknown unless it is explicitly asserted in Λ.* This restricts greatly the use we can make of P but it makes sure that, whenever it is used, all that we need to know (prove) about it is in Λ.

FQRD-constructors

The standard algebraic constructors *FQRD* in **EQ** are the most well-known set of constructors. Although they make sense only in the context of an algebraic-like institution, some details are worth giving.

Contrary to generic constructors such as $\langle P, \Lambda\rangle$, constructors in *FQRD* are, essentially, model-theoretical operations. For this reason, inference rules for F, Q and R (D has already been treated at the institution-independent level) are scarce and may depend on awkward side conditions.

Proposition 3.4.2 $\dfrac{eq \vdash \forall X.\ t1 = t2}{Q_{eq} SP \vdash \forall X.\ t1 = t2}$ *is a sound inference rule.*

Proof By definition of quotient $A/\sim_{eq} \models eq$ for all $A \in Mod[SP]$, then by soundness of equational logic $eq \vdash \forall X.\ t1 = t2$ implies that $A/\sim_{eq} \models \forall X.\ t1 = t2$ for all $A \in Mod[SP]$, hence $Q_{eq} SP \models \forall X.\ t1 = t2$. □

Proposition 3.4.3 $\dfrac{SP \vdash \forall X.\ t1 = t2}{Q_{eq}SP \vdash \forall X.\ t1 = t2}$ *is a sound inference rule.*

Proof Let A be a model of SP and $\sim$ a congruence on A, and assume that

$$A \models \forall X.\ t1 = t2 \quad \text{but} \quad A/\sim \not\models \forall X.\ t1 = t2$$

Then, there must exist a valuation $\nu_\sim : T_{Sig[SP]}(X) \to A/\sim$ such that $A/\sim, \nu_\sim \not\models t1 = t2$. Define $\nu : T_{Sig[SP]}(X) \to A$ to be a valuation which delivers for each $x \in X$ a representative of $\nu_\sim(x)$, *i.e.* $[\nu(x)] = \nu_\sim(x)$.

Since congruences commute w.r.t. all operations in A, $\nu_\sim(t1) = [\nu(t1)]$ and consequently $[\nu(t1)] \neq [\nu(t2)]$ so their representatives must be different $\nu(t1) \neq \nu(t2)$, hence

$$A, \nu \not\models t1 = t2$$

Thus $A \not\models \forall X.\ t1 = t2$, which is a contradiction. □

Proposition 3.4.4 $\dfrac{SP \vdash \forall X.\ t1 = t2}{R_G SP \vdash \forall X.\ t1 = t2}$ *is a sound inference rule.*

Proof Let A be a model of SP and A' a subalgebra of A, and assume that

$$A \models \forall X.\ t1 = t2 \quad \text{but} \quad A' \not\models \forall X.\ t1 = t2$$

Then, there must exist a valuation ν' such that $A', \nu' \not\models t1 = t2$. Define $\nu : T_{Sig[SP]}(X) \to A$ to be a valuation which extends ν' to A, *i.e.* $\nu(x) = \nu'(x)$ for all $x \in X$.

Finally, since the interpretation of the function symbols in A and A' is the same when applied to values existing in the subalgebra A' we conclude that

$$\nu(t1) = \nu'(t1) \neq \nu'(t2) = \nu(t1)$$

Hence, $A, \nu \not\models t1 = t2$. Thus $A \not\models \forall X.\ t1 = t2$, which is a contradiction. □

Proposition 3.4.5 $\dfrac{eq \vdash \forall X.\ t1 = t2}{F_\sigma^{eq} SP \vdash \forall X.\ t1 = t2}$ *is a sound inference rule.*

Proof By definition, for $\sigma : \Sigma \to \Sigma'$, $Free_\sigma^{eq} : Alg(\Sigma) \to Mod[\langle \Sigma', eq \rangle]$. Then, it holds that $Mod[F_\sigma^{eq} SP] \subseteq Mod[\langle \Sigma', eq \rangle]$ and since $Mod[\langle \Sigma', eq \rangle] = Mod[Cl(eq)]$ it follows that all semantic consequences of eq hold in $Mod[F_\sigma^{eq} SP]$. By soundness of equational logic the rule is sound. □

A free extension $F_\sigma^{eq} SP$ is **sufficiently complete** if it does not add new values to the old sorts, although it can confuse (quotient) some of the old values. Formally, $F_\sigma^{eq} SP$ is sufficiently complete if the unit associated to the pair F_σ^{eq} and $\_|_\sigma$ for all $A \in Mod[SP]$ is a surjective homomorphism from A to $(Free_\sigma^{eq} A)|_\sigma$.

A free extension $F_\sigma^{eq} SP$ is **consistent** (or hierarchically consistent) if it preserves the old values of the old sorts along the extension, although it may add new ones. Formally, $F_\sigma^{eq} SP$ is consistent if the unit associated to the pair F_σ^{eq} and $\_|_\sigma$ for all $A \in Mod[SP]$ is a injective homomorphism from A to $(Free_\sigma^{eq} A)|_\sigma$.

Note that these definitions generalize the standard definitions of consistent and sufficiently complete free extension which refer exclusively to the reachable models of SP or, equivalently, to its initial model (see [Ber 87] for discussion).

Proposition 3.4.6 $\dfrac{SP \vdash \forall X.\ t1 = t2}{F_\sigma^{eq} SP \vdash \sigma(\forall X.\ t1 = t2)}$ *is a sound inference rule provided $F_\sigma^{eq} SP$ is sufficiently complete and consistent.*

Proof If a free extension is sufficiently complete and consistent then it preserves the meaning of the symbols in SP. Formally, the unit associated to the pair F_σ^{eq} and $\_|_\sigma$ for all $A \in Mod[SP]$ is an isomorphism from A to $(Free_\sigma^{eq} A)|_\sigma$

$$\forall A \in Mod[SP].\ (Free_\sigma^{eq} A)|_\sigma \cong A$$

Considering an arbitrary model $A \in Mod[SP]$ and an equation $\forall X.\ t1 = t2$, if $A \models \forall X.t1 = t2$ then $(Free_\sigma^{eq} A)|_\sigma \models \forall X.\ t1 = t2$ and, by the satisfaction condition, $Free_\sigma^{eq} A \models \sigma(\forall X.\ t1 = t2)$. □

3.5 Specification entailment

The main conclusion that can be extracted from the inference rules given in this chapter and their characterizations is the following:

Proofs of specification entailment based on reducing specifications to equivalent theories are of very limited use.

In fact, we can only expect an exact reduction of specifications to theories for ATU-specifications, since no other SBO has an exact rule. This result was already expected due to the greater expressive power of specifications in $ASL(\mathbf{I})$ than common theories in $\mathbf{I}$ for common institutions used in formal specification.

This result shows the most striking difference, from a proof theoretic standpoint, between *ASL* and other common specification languages where semantics are directly given in terms of presentations or theories.

Once the idea of reducing specifications to theories has been discarded, emphasis has been put on the characterization of some inference rules (mostly obtained from [ST 88a]).

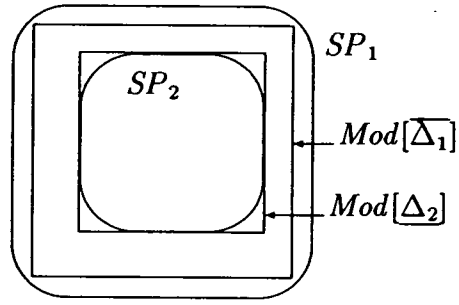
The definition of sound and M-complete inference rules helps us to analyze the nature of some inference rules and provides a better alternative than reducing specifications to exact theories; that is, reducing specifications to sound or M-complete theories.

We can use these reductions to prove $SP2 \models SP1$ as follows:

Theorem 3.5.1 (Basic rule) *Given an institution* $\mathbf{I}$, *if there exists an M-complete theory* $\overline{\Delta_1}$ *for* $SP1$, *a sound theory* $\underline{\Delta_2}$ *for* $SP2$ *and* $\underline{\Delta_2} \vdash_{\mathbf{I}} \overline{\Delta_1}$, *then* $SP2 \models SP1$.

Proof Since $\vdash_{\mathbf{I}}$ is sound, $\underline{\Delta_2} \vdash_{\mathbf{I}} \overline{\Delta_1}$ implies $\underline{\Delta_2} \models \overline{\Delta_1}$, which by corollary 3.3.2 means that $\underline{\Delta_2} \supseteq \overline{\Delta_1}$. Applying *Mod* we get $Mod[\underline{\Delta_2}] \subseteq Mod[\overline{\Delta_1}]$, and by definition of M-complete and M-sound (equivalent to sound) $Mod[SP_2] \subseteq Mod[\underline{\Delta_2}] \subseteq Mod[\overline{\Delta_1}] \subseteq Mod[SP_1]$. □

The situation where the theorem applies can be represented by the diagram on the right. Every figure represents a class of models; in particular, squares represent axiomatizable classes of models.



The theorem states that the inclusion of the two ovals can be proved by showing the inclusion of the two squares. In practice, this can be performed by a theorem prover using the inference system provided with the institution $\vdash^{\mathbf{I}}$ plus the inference rules presented in this chapter, in such a way that only sound rules are use for the antecedent $SP2$ and only M-complete rules are used for the consequent $SP1$.

Looking at the drawing the basic rule seems obvious but also the lack of anything better unless every class of models described by the specification language can be characterized by an exact theory. The situation can only be improved by increasing the expressive power of theories and/or weakening the expressiveness of the specification language.

Going back to the inference rules provided in this chapter, we find no M-complete rules apart from those which are also sound, and therefore exact. In principle, M-complete rules may not be sound and still be useful during the application of the *Basic rule* defined above. This is clear in the case of an abstraction

operation α where a rule

$$\frac{SP \vdash \varphi}{\alpha SP \vdash \varphi}$$

is not sound since some "non-observable" theorems in SP may not hold in the abstracted specification αSP, however, the rule is useful in the sense that an entailment

$$SP2 \models \alpha SP1$$

could be proven correct by showing that $SP2 \models SP1$, since the rule for abstraction is M-complete. Essentially, we prove that the antecedent satisfies more requirements than those needed.

Unfortunately, our research did not produce any interesting M-complete rule for D_σ or $M_{(\sigma,\iota)}$ even among unsound rules.

Restricting to the inference rules presented in this chapter, the rules in $\vdash^{\mathbf{I}}$ and the basic rule, entailments $SP2 \models SP1$ can only be proven to hold if $SP1$ is an ATU-specification. The limitation to ATU means, for institutions such as **EQ**, that we are unable to specify standard models of natural numbers or to use any hiding/abstraction mechanism in the consequent. In the context of a stepwise refinement of specification into programs, the antecedent of the first entailment is used as the consequent of the second entailment and so on,

$$SP_n \models SP_{n-1} \models ... \models SP2 \models SP1$$

hence the restriction to ATU-specifications is extended from the original specification of the problem $SP1$ to SP_{n-1}.

On the other hand, $SP2$ can be a specification in $k{+}ASL$ since the basic rule only requires sound inference rules for those SBO's used in the antecedent. The problem on this side comes from the incompleteness of some sound rules which may prevent some valid entailments from being derivable. This is particularly true for $M_{(\sigma,\iota)}$ and some model oriented constructors such as F_σ^{eq}, R_G and Q_{eq} in **EQ**.

All these considerations lead to a very limited collection of provable entailments $SP2 \models SP1$. In following chapters we shall gradually improve the situation but we can already identify which are the main problems.

The impossibility of using D_σ in the consequent is the first important problem. A solution will be found in chapters 4 and 5.

The impossibility of using $M_{\langle G,F,S\rangle}$ in the consequent despite having an induction rule is the second problem. A solution in **EQ** will be found in chapter 6.

However, the impossibility of using constructors in the consequent is not such an important problem. Constructors are normally thought of as pieces of a specification which have already been implemented during a refinement step (see constructor implementations in [ST 88b]). Therefore, later refinements should not *re-implement* constructors.

According to this view, constructors are mostly relevant in the antecedent. Constructors in the consequent will play a very limited role mostly related to decomposing large consequents for modular verification. These structural aspects of verification are addressed in chapter 7.

Chapter 4

Specifications with hidden parts

4.1 Introduction

In the last chapter we came across the particular nature of D_σ. D_σ is the only SBO having no exact inference rule despite having a sound and complete inference rule.

In the context of proving $SP2 \models SP1$, such a characterization means that using D_σ in $SP2$ has no unpleasant consequences since every valid theorem in $SP2$ (over $DATU$) can be inferred (completeness). However, D_σ cannot be eliminated from a specification (no exact rule) and therefore, if D_σ is used in $SP1$, proving $SP2 \models SP1$ cannot be reduced to simple theorem proving.

This technical problem happens to be a well-known problem in proofs of correctness of implementation steps. Specifications using D_σ are known in the literature as specifications with hidden parts (hidden functions and/or hidden sorts) [TWW 78], or specifications with a bias [Jon 80]. In any case, when such specifications act as consequent, *i.e.* when they express the requirements which must be met by the antecedent or implementation, verification becomes hard.

Many algebraic specifications of a system, especially if they are large, provide a lot of information describing functions and data types which are not interesting to

the user of the system or to the person implementing that system. This auxiliary structure (hidden parts) helps to define the functions and sorts of interest but they are not meant to be implemented.

Hidden parts are the standard way to express requirements in abstract model specifications. In VDM, for example, data types are specified by representing them in terms of some predefined types such as lists, sets, mappings and sequences. These predefined types are the means for expressing requirements but they are not intended to be implemented. Changing the representation of the specified data types in the implementation is known as data reification [Jon 80, Jon 86], and correctness proofs can be very difficult. For example, consider a `compiler` function which is specified by a simple interpreter and then implemented in a more efficient way without respecting any of the structures by means of which `compiler` was specified. The proof of correctness would be a great deal more difficult than the proof of correctness of a direct implementation of the interpreter.

Algebraic specifications became popular as a way to define the elements of a system without representing them in terms of more primitive concepts, avoiding the definition of any extra structure. However, it has been shown that hidden functions are in general necessary for specifying computable functions in equational logic [Maj 77, TWW 78, BBTW 81], therefore the use of hiding is not characteristic of a specification style but is a fundamental SBO for writing abstract specifications.

The necessity of using hidden functions in order to specify some computable functions proves that in some institutions some extra expressiveness is achieved using hiding; this results in the impossibility of flattening arbitrary specifications containing D_σ. Technically, the characterization of the inference rule for D_σ as complete but not M-complete witnesses this fact.

In some powerful logics, such as second order logic (SOL), the hiding of functions does not provide enhanced power. But in common logics used in formal specification, such as equational or first order logic, it certainly does. Intuitively,

hiding can be seen as a second order feature which can be added to weaker logics. Thus, correctness proofs in $DATU \models DATU$ are going to cause some of the same difficulties as the introduction of second order existential quantifiers does in SOL.

4.1.1 Example

The cases we are mostly interested in are those where some auxiliary functions/sorts are defined in order to specify some desired functions/sorts but the implementation does not follow the bias, *i.e.* the auxiliary structure is not implemented. With these cases, we expect to cover the most common difficulties involved in proofs of entailment w.r.t. specifications with hidden parts.

For example, suppose we want to add a function `mre: listnat -> nat` which computes the most repeated element in a list, to a specification *ListNat* of lists of natural numbers. Soon we realize that the available functions in *ListNat*, *i.e.*

`[]: list`	{empty list}
`_::_: nat, list -> list`	{adds an element to a list}
`car: list -> nat`	{returns the first element of a list}
`cdr: list -> list`	{removes the first element from a list}
`0: nat`	
`suc: nat -> nat`	
`_≥_: nat, nat -> bool`	
`true, false: bool`	

are not enough to conveniently define the new function. We proceed by defining an auxiliary function which counts the number of repetitions of a natural number in a list of natural numbers, as in figure 4–1.

Any efficient implementation of `mre` will not implement `count` itself but will instead implement a function which computes simultaneously the frequencies of

```
SP1 =    Enrich ListNat by
         Hidden
           sorts
           operations
             count:  listnat, nat -> nat
           axioms
             ∀ x:nat. count([], x) = 0
             ∀ x:nat; l:listnat. count(x::l, x) = suc(count(l,x))
             ∀ x1,x2:nat; l:listnat.
               (x1≠x2) ⇒ (count(x1::l,x2) = count(l,x2))
         in
           sorts
           operations
             mre:  listnat -> nat
           axioms
(1)          ∀ x:nat; l:listnat.  (count(l, mre(l)) ≥ count(l,x)) = true
         end
```

Figure 4–1: Lists with a `mre` (most repeated element) function

all elements in the list. In a stepwise refinement methodology, the specification in figure 4–2 leads in the direction of the desired implementation by defining `mre` in terms of a new function `frequencies: listnat -> set(nat x nat)` and some extra structure defining sets of pairs of natural numbers *SetPairsNat* (with sort name `set(nat x nat)`) and lists of pairs of natural numbers *ListPairsNat* (with sort name `list(nat x nat)`).

The function `frequencies` yields a set of pairs, where each pair contains a counter and an element of the argument list; for example

```
frequencies([1,3,2,1])={(1,3),(1,2),(2,1)}
```

Then we have that

```
car(sort(frequencies([1,3,2,1])))=(2,1)
```

and so `mre([1,3,2,1])=1`.

In the next refinement step `frequencies` can be made visible, *i.e.* forced to be implemented, or its definition may be changed to yield a table. Moreover picking the largest element need not be implemented using `car(sort(...))`. Eventually, after a few steps we obtain a correct and efficient implementation.

The above refinement step is proven correct by showing that the axiom (1) which defines `mre` in the specification *SP*1 can be deduced from the implementation, *SP*2, together with the definition of `count` (the hidden part of *SP*1); this repeats the "proof procedure" already used in [San 86]. Informally the proof proceeds by proving that:

1. The frequencies of the elements of a list are in the set created by `frequencies`.

```
∀ l:listnat; x:nat.
   (count(l,x)≠0) ⇒ (count(l,x),x)∈ frequencies(l) = true
```

SP2 = Enrich *ListNat* by

```
Hidden
  sorts
  operations
    ...
    {operations of SetPairsNat and ListPairsNat}
    ...
    sort:  set(nat x nat) -> list(nat x nat)
    frequencies:  listnat -> set(nat x nat)
  axioms
    ...
    {axioms of SetPairsNat and ListPairsNat}
    ...
    frequencies([])= ∅
    ∀ l:listnat; x:nat.
      (∃ i:nat.  (i,x)∈frequencies(l) = true) ⇒
      (∀ i:nat.  (i,x)∈frequencies(l) = true ⇒
      frequencies(x::l) = frequencies(l)\{(i,x)}∪{(i+1,x)})
    ∀ l:listnat; x:nat.
      (∄ i:nat.  (i,x)∈frequencies(l) = true) ⇒
      (frequencies(x::l) = frequencies(l)∪{(1,x)})
    ...
    { Axioms for sorting a set of pairs by the first
    element of the pair, in decreasing order }
    ...
in
  sorts
  operations
    mre:  listnat -> nat
  axioms
    mre([]) = 0
    ∀ l:listnat.  (l ≠ []) ⇒
      (mre(l) = proj2(car(sort(frequencies(l)))))
end
```

2. All counters in the set created by `frequencies` denote frequencies.

   ```
   ∀ i,x:nat; l:listnat.
     ((i,x)∈ frequencies(l) = true) ⇒ i = count(l,x)
   ```

3. By definition of `sort` and `car` the term `car(sort(s))` yields the pair in the set `s` with the highest counter.

   ```
   ∀ s:set(nat x nat).  (s≠∅) ⇒
     ((∀ i,y:nat. (i,y)∈s = true) ⇒ (proj1(car(sort(s)))≥i = true))
   ```

4. Similarly taking `s` to be the set of frequencies of a list `l`,

   ```
   car(sort(frequencies(l)))
   ```

   yields the pair with the highest frequency.

   ```
   ∀ l:listnat. (l=[]) ∨
     ((∀ x:nat. (i,x)∈frequencies(l) = true) ⇒
      (proj1(car(sort(frequencies(l))))≥i = true))
   ```

5. Hence, `mre(l)` is the most repeated element in `l` provided `l` is not an empty list, and zero when `l` is empty. Therefore, it is always the case that:

   ```
   ∀ x:nat; l:listnat.  count(l, mre(l)) ≥ count(l,x) = true
   ```

Apart from the technicalities of the proof, the interest of this example arises from the difficulty in generalizing this style of proof to a specification language with arbitrary use of hiding and justifying that it is sound.

The rest of the chapter is divided into six sections. Section 2 gives a formal presentation of the problem and some simple strategies are shown not to be satisfactory. Section 3 emphasizes the special structure of specifications such as $SP1$

and introduces two properties applicable to specifications of this form: *persistency* and *independence*. In section 4 the inheriting strategy, a formal version of the strategy used in the above example, is presented and proved sound for persistent hidden parts. In section 5 the inheriting strategy is proven complete for independent specifications. Section 6 generalizes the inheriting strategy to the case of $k{+}ASL \models DATU$. Finally, section 7 copes with some difficult cases falling out of the scope of the inheriting strategy.

4.2 Proving $DA \models DA$

A general setting in which to study entailment proofs between specifications with hidden parts is $ASL \models ASL$. For the time being, we shall ignore reachability constraints (see chapter 6) and focus on $DATU \models DATU$.

However, the problem can be simplified to $DA \models DA$ without loss of generality since considering some transformation rules in addition to those for ATU in section 3.2 allows the elimination of T and U in $DATU$.

Proposition 4.2.1 *The following specification transformations are sound for all specifications SP, $SP1$ and $SP2$, morphisms σ and $\sigma 1$, and set Φ of sentences*

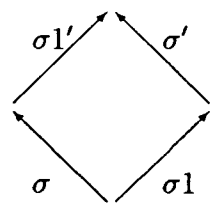
$$A_\Phi D_\sigma SP = D_\sigma A_{\sigma(\Phi)} SP$$

$$T_\sigma D_{\sigma 1} SP = D_{\sigma 1'} T_{\sigma'} SP$$

$$SP1 \ U \ D_\sigma SP2 = D_\sigma (T_\sigma SP1 \ U \ SP2)$$

$$D_\sigma SP1 \ U \ SP2 = D_\sigma (SP1 \ U \ T_\sigma SP2)$$

where $\sigma 1'$ and σ' are defined by the following pushout diagram:



Proof

1. $A_\Phi D_\sigma SP = D_\sigma A_{\sigma(\Phi)} SP$

   The proof can be presented as a chain of equalities between classes of models:

   $Mod[A_\Phi D_\sigma SP]$
   $= \{A|_\sigma \mid A \in Mod[SP] \wedge A|_\sigma \models \Phi\}$
   $= \{A|_\sigma \mid A \in Mod[SP] \wedge A \models \sigma(\Phi)\}$ by the satisfaction condition.
   $= Mod[D_\sigma A_{\sigma(\Phi)} SP]$

2. $T_\sigma D_{\sigma 1} SP = D_{\sigma 1'} T_{\sigma'} SP$

   The proof can be presented as a chain of equalities between classes of models:

   $Mod[T_\sigma D_{\sigma 1} SP]$
   $= \{A \mid A|_\sigma \in Mod[D_{\sigma 1} SP]\}$
   $= \{A \mid \exists B \in Mod[SP].\ B|_{\sigma 1} = A|_\sigma\}$
   $= \{C|_{\sigma 1'} \mid C|_{\sigma'} \in Mod[SP]\}$ since for every A and B such that $B|_{\sigma 1} = A|_\sigma$, there is a C such that $C|_{\sigma 1'} = A$ and $C|_{\sigma'} = B$ *i.e.* $C = A \oplus B$.
   $= \{C|_{\sigma 1'} \mid C \in Mod[T_{\sigma'} SP]\}$
   $= Mod[D_{\sigma 1'} T_{\sigma'} SP]$

3. $SP1\ U\ D_\sigma SP2 = D_\sigma(T_\sigma SP1\ U\ SP2)$

   The proof can be presented as a chain of equalities between classes of models:

   $Mod[SP1\ U\ D_\sigma SP2]$
   $= \{A \mid A \in Mod[SP1] \wedge (\exists B \in Mod[SP2].\ B|_\sigma = A)\}$
   $= \{B|_\sigma \mid B \in Mod[SP2] \wedge B \in Mod[T_\sigma SP1]\}$
   $= Mod[D_\sigma(T_\sigma SP1\ U\ SP2)]$

4. $D_\sigma SP1 \ U \ SP2 = D_\sigma(SP1 \ U \ T_\sigma SP2)$

   Proven analogously to 3 above.

□

Theorem 4.2.2 *Every specification SP over DATU is equivalent to a specification of the form $D_\sigma A_\Phi \epsilon_\Sigma$, where the reduction from SP to $D_\sigma A_\Phi \epsilon_\Sigma$ is computable.*

Proof It is enough to consider the reduction rules from ATU to A, the commutativity rules for D_σ given above (considered from left to right) and the rule

$$D_\sigma D_{\sigma'} SP = D_{\sigma;\sigma'} SP$$

where $\uparrow \sigma = \downarrow \sigma'$. Then, every specification SP in $DATU$ can be transformed first into a specification $D_\sigma SP'$ for some signature morphism σ and a specification SP' in ATU (using the commutativity rules for D_σ) and, from that, it can be transformed into a specification $D_\sigma A_\Phi \epsilon_\Sigma$ using the rules in section 3.2.

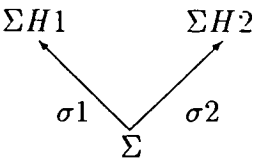
□

These proofs are among those presented in [EWT 83]. Nevertheless, in [EWT 83] the result is only stated for the particular institution they use. Results in [Bre 89] also repeat a similar proof, however all results are unnecessarily restricted to algebraic signatures and to D_σ with injective signature morphisms. For this reason, proofs have been given again instead of using a corollary to the normal form results in [Bre 89] or [EWT 83].

Taking advantage of these results, $DATU \models DATU$ is reduced without loss of generality to the following task:

Task 1: $D_{\sigma 2} A_{\Phi 2} \epsilon_{\Sigma H2} \models D_{\sigma 1} A_{\Phi 1} \epsilon_{\Sigma H1}$

Given two finite presentations $\Phi 1$ *and* $\Phi 2$ *over signatures* $\Sigma H1$ *and* $\Sigma H2$ *which are two extensions of* Σ, *prove that all models over* Σ *obtained by restricting models of* $\Phi 2$ *can be obtained by restricting models of* $\Phi 1$.

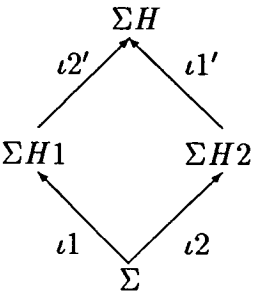


Note: In order to simplify notation signature morphisms $\sigma 1$ and $\sigma 2$ are assumed to be inclusions $\iota 1$ and $\iota 2$. However, no advantage of this assumption will be taken unless stated.

Given such a task there are only two *reasonable* things one may try to do: prove that $\Phi 1$ follows from $\Phi 2$ in an appropriate signature, or prove that all visible consequences of $\Phi 1$ follow from $\Phi 2$. Unfortunately, the first approach only works for some trivial cases and the second one is unsound.

First naïve approach.

In this approach we want to mix in our reasoning sentences over $\Sigma H1$ and over $\Sigma H2$ without confusing their auxiliary symbols; therefore, we consider the pushout signature ΣH and try to prove that $\iota 1'(\Phi 2) \models \iota 2'(\Phi 1)$ (we will also write this as $\Phi 2 \models_{\Sigma H} \Phi 1$).



In an institution such as first order logic, this amounts, by the Craig interpolation lemma, to the existence of a finite set of sentences Φ over Σ such that

$$(2) \quad \Phi 2 \models_{\Sigma H2} \Phi \quad \textit{and} \quad \Phi \models_{\Sigma H1} \Phi 1$$

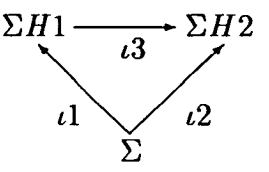
Turning around the argument, a refinement $D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H2} \models D_{\iota 1} A_{\Phi 1} \epsilon_{\Sigma H1}$ can only be proven correct in this fashion if there exists an intermediate flat specification $A_{\Phi} \epsilon_{\Sigma}$ such that Φ satisfies (2).

However, if this is the case, the symbols in the hidden part of $\Phi 1$ are rather trivial since their properties can be inferred from sentences in Φ which do not mention them.

This approach offers surprisingly little. We may think that at least those implementations which proceed by implementing the hidden part should be provably correct. For example, given a the specification $SP1$ in figure 4–1 as consequent we propose $SP3$ in figure 4–3 as antecedent.

In this case the proof is easier because the implementation follows the bias of the specification, and we can directly prove that $\Phi 2 \models_{\Sigma H2} \Phi 1$.

The difference with the approach just proposed is the existence of a morphism $\iota 3$ (identity in this example) which is missing in the general case.



On the the other hand, the existence of $\iota 3$ does not guarantee that a refinement can be proved correct in this fashion since the same symbols may have different meanings in the two specifications.

For example, consider the result returned by `count` in the specification $SP4$ (see figure 4–4). In this case, the result of `count` is double the result returned by `count` in $SP1$. Therefore, although $SP4 \models SP1$ is a correct entailment and $Sig[SP4] = Sig[SP1]$, the *bias* is not followed and the proof method above does not work.

Second naïve approach

More realistic is the second approach where we prove that:

$$\Phi 2 \models_{\Sigma H2} \iota 1^{-1}(Cl(\Phi 1))$$

*SP*3 = Enrich *ListNat* by
Hidden
sorts
operations
count: listnat, nat -> nat
axioms
...
{ Axioms of count as in *SP*1}
...
in
sorts
operations
mre: listnat -> nat
axioms
mre([])= 0
∀ x:nat; l:listnat.
(count(l,x)+1 ≤ count(l,mre(l)) = true ⇒
mre(x::l) = mre(l))
∀ x:nat; l:listnat.
(count(l,x)+1 > count(l,mre(l)) =true ⇒
mre(x::l) = x)
end

Figure 4–3: Lists with a mre function, version 3

```
SP4 =    Enrich ListNat by
         Hidden
           sorts
           operations
             count:  listnat, nat -> nat
           axioms
             ∀ x:nat. count([],x) = 0
             ∀ x:nat; l:listnat. count(x::l, x) = count(l,x) + 2
             ∀ x1,x2:nat; l:listnat.
               (x1≠x2) ⇒ (count(x1::l,x2) = count(l,x2))
         in
           sorts
           operations
             mre:  listnat -> nat
           axioms
             ...
             { Axioms of mre as in SP3}
             ...
         end
```

Figure 4–4: Lists with a `mre` function, version 4

where $\iota 1^{-1}$ forgets the sentences which mention symbols not in Σ. Unfortunately, due to the difference in expressive power between flat specifications (even if they are infinite) and presentations with hidden parts studied in [Maj 77, TWW 78], this proof strategy is, in general, unsound [BHK 86, Far 89]. For example (from [BHK 86]), suppose we are asked to prove

$$Nat \models NStNat$$

where *Nat* is a specification of the natural numbers including the standard interpretation alongside the non-standard interpretations, whereas *NStNat* (figure 3–2) specifies the class of the non-standard models of the natural numbers. It is well-known that all the first-order properties of the non-standard models of the natural numbers are satisfied by the standard models; however the entailment is incorrect.

In many cases this strategy is sound but $\iota 1^{-1}(Cl(\Phi 1))$ is infinite and not finitely presentable, making the proof very hard. Even worse, if the strategy is sound and $\iota 1^{-1}(Cl(\Phi 1))$ happens to be equivalent to a finite presentation Φ, we may be able to prove that $\Phi 2 \models \Phi$ but it may be difficult to prove the equivalence between Φ and $\iota 1^{-1}(Cl(\Phi 1))$, in particular this may be as difficult as the original task.

Consider for example the specification of `max: listnat -> nat` which yields the maximum element of a list of natural numbers, by means of an auxiliary function `sort: listnat -> listnat`, as in figure 4–5.

The function `max` can be directly specified using the available operations in *ListNat*:

(3)
```
max:  listnat -> nat
max([]) = 0
∀ l:listnat;x:nat.
  (max(l)>x = true ⇒ max(x::l) = max(l))
∀ l:listnat;x:nat.
  (max(l)≤x = true ⇒ max(x::l) = x)
```

```
SP5 =    Enrich ListNat by
         Hidden
           sorts
           operations
             sort:  listnat -> listnat
           axioms
             ...
             { Axioms for sorting }
             ...
         in
           sorts
           operations
             max:  listnat -> nat
           axioms
             max([]) = 0
             ∀ l:listnat. l≠[] ⇒ max(l) = car(sort(l))
         end
```

Figure 4–5: Lists with a max function

Then, the visible consequences of $SP5$ can be finitely axiomatized by adding (3) to *ListNat* so that an implementation $D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H2}$ can be proven correct by proving $\Phi 2 \models ListNat \cup (3)$. In this case, the strategy turns out to be sound.

But, in fact, there is still the problem of proving that the proposed axiomatization (3) is correct, in other words, that $ListNat \cup (3) \models SP5$, but this is just another instance of our general task in the particular case that $\Sigma H2$ is equal to Σ, which can be as difficult as the task we started with (see e.g. histogram example in [San 86]).

A strategy based on second order existential quantifiers

A hiding operator, D_ι with an inclusion ι, can be seen to a certain extent as a second order existential quantifier. If we can quantify over functions it is very easy to specify functions such as `mre` in figure 4–1 by writing a sentence such as the following:

$\exists^2$ `count: listnat, nat -> nat. {Axioms for count}` $\wedge$ `{Axioms for mre}`

Keeping this analogy in mind, a new strategy resembling the introduction rule for $\exists^2$ in SOL can be found. The introduction rule of $\exists^2$ says that before being able to assert that a function with a certain property exists, an explicit function with that property must be given; then, introducing $\exists^2$ discharges (hides) this function.

In the implementation of `mre` via `frequencies`, the $\exists^2$-strategy amounts to explicitly including `count` in the implementation in spite of the fact that it is not needed. Once `count` is in the implementation, the proof can easily be performed ignoring the hiding.

Formally this strategy can be presented as the following rule:

Proposition 4.2.3 *Given signature morphisms* $\sigma 1 : \Sigma \rightarrow \Sigma H1$, $\sigma 2 : \Sigma \rightarrow \Sigma H2$ *and* $\sigma : \Sigma H1 \rightarrow \Sigma H$, *and sets of sentences* $\Phi 1 \subseteq Sen(\Sigma H1)$, $\Phi 2 \subseteq Sen(\Sigma H2)$ *and* $\Phi \subseteq Sen(\Sigma H)$, *then*

$$\frac{D_\sigma(A_\Phi T_{\sigma 1;\sigma})D_{\sigma 2}A_{\Phi 2}\epsilon_{\Sigma H2} \models A_{\Phi 1}\epsilon_{\Sigma H1} \qquad D_{\sigma 2}A_{\Phi 2}\epsilon_{\Sigma H2} \models D_{\sigma 1;\sigma}(A_\Phi T_{\sigma 1;\sigma})D_{\sigma 2}A_{\Phi 2}\epsilon_{\Sigma H2}}{D_{\sigma 2}A_{\Phi 2}\epsilon_{\Sigma H2} \models D_{\sigma 1}A_{\Phi 1}\epsilon_{\Sigma H1}}$$

is sound.

Proof Soundness follows immediately from the transitivity of $\models$ and the monotonicity of $D_{\sigma 1}$ w.r.t. model class inclusion (entailment), since the conclusion is the composition of the premises after applying $D_{\sigma 1}$ to both sides of the first premise.

□

Note that $(A_\Phi T_{\sigma 1;\sigma})$ is an enrichment to the antecedent (implementation) which must be guessed. The second premise can be seen as a side condition guaranteeing the persistency of $(A_\Phi T_{\sigma 1;\sigma})$ w.r.t. $D_{\sigma 2}A_{\Phi 2}\epsilon_{\Sigma H2}$ and $\langle id_\Sigma, (\sigma 1;\sigma)\rangle$, and the first premise is the desired proof ignoring the hiding provided by $D_{\sigma 1}$. The following theorem shows that this rule can solve our task (Task 1).

Theorem 4.2.4 (Wir 91) *Given signature morphisms* $\sigma 1 : \Sigma \rightarrow \Sigma H1$ *and* $\sigma 2 : \Sigma \rightarrow \Sigma H2$, *a set* $\Phi 1$ *of* $\Sigma H1$*-sentences and a set* $\Phi 2$ *of* $\Sigma H2$*-sentences such that*

$$D_{\sigma 2}A_{\Phi 2}\epsilon_{\Sigma H2} \models D_{\sigma 1}A_{\Phi 1}\epsilon_{\Sigma H1}$$

then, there exists a signature ΣH, *a morphism* $\sigma : \Sigma H1 \rightarrow \Sigma H$ *and a set* Φ *of* ΣH*-sentences such that*

$$D_\sigma(A_\Phi T_{\sigma 1;\sigma})D_{\sigma 2}A_{\Phi 2}\epsilon_{\Sigma H2} \models A_{\Phi 1}\epsilon_{\Sigma H1}$$

and

$$D_{\sigma 2}A_{\Phi 2}\epsilon_{\Sigma H2} \models D_{\sigma 1;\sigma}(A_\Phi T_{\sigma 1;\sigma})D_{\sigma 2}A_{\Phi 2}\epsilon_{\Sigma H2}$$

Proof Take σ and Φ to be $id_{\Sigma H1}$ and $\Phi 1$ respectively and substitute in the conclusion of the theorem

$$(A_{\Phi 1}\, T_{\sigma 1}) D_{\sigma 2} A_{\Phi 2} \epsilon_{\Sigma H2} \models A_{\Phi 1} \epsilon_{\Sigma H1} \qquad D_{\sigma 2} A_{\Phi 2} \epsilon_{\Sigma H2} \models D_{\sigma 1}(A_{\Phi 1}\, T_{\sigma 1}) D_{\sigma 2} A_{\Phi 2} \epsilon_{\Sigma H2}$$

The first entailment holds trivially since any model of the antecedent has to satisfy $\Phi 1$ and therefore it is also a model of $A_{\Phi 1} \epsilon_{\Sigma H1}$.

In order to prove the second entailment, consider an arbitrary model $A \in Mod[D_{\sigma 2} A_{\Phi 2} \epsilon_{\Sigma H2}]$. According to the assumption, $D_{\sigma 2} A_{\Phi 2} \epsilon_{\Sigma H2} \models D_{\sigma 1} A_{\Phi 1} \epsilon_{\Sigma H1}$, we know that $A \in Mod[D_{\sigma 1} A_{\Phi 1} \epsilon_{\Sigma H1}]$. By definition of $D_{\sigma 2}$ and $D_{\sigma 1}$, there exists a $\Sigma H2$-model B and a $\Sigma H1$-model C such that

$$A = B|_{\sigma 2} = C|_{\sigma 1} \qquad B \models_{\Sigma H2} \Phi 2 \qquad C \models_{\Sigma H1} \Phi 1$$

thus $A \in Mod[D_{\sigma 1}(A_{\Phi 1}\, T_{\sigma 1}) D_{\sigma 2} A_{\Phi 2} \epsilon_{\Sigma H2}]$.

□

Considering this theorem and the normal form result in theorem 4.2.2 for specifications in *DATU*, we may think that $DATU(\mathbf{I}) \models DATU(\mathbf{I})$ has a sound and complete inference system, provided $\vdash^{\mathbf{I}}$ is sound and complete. However, this is not the case unless *persistency, i.e.* $D_{\sigma 2} A_{\Phi 2} \epsilon_{\Sigma H2} \models D_{\sigma 1;\sigma}(A_{\Phi}\, T_{\sigma 1;\sigma}) D_{\sigma 2} A_{\Phi 2} \epsilon_{\Sigma H2}$, can be proven. A similar result found in [Wir 91] asserts the existence of a complete inference system for $DATU \models DATU$ but that relies on the fact that persistency is treated as a side condition instead of a premise.

In a more general form the $\exists^2$-strategy can be presented as:

$$\frac{PSP2 \models SP1 \qquad SP2 \models D_{\sigma 1} PSP2}{SP2 \models D_{\sigma 1} SP1}$$

where P is a parameterized specification $(\lambda\, X :\downarrow \sigma 1.\ SP)$ taking specifications over $\downarrow \sigma 1$ and producing specifications over $\uparrow \sigma 1 = Sig[SP1]$.

Analogously to the rule above, soundness of this rule follows immediately from the transitivity of $\models$ and the monotonicity of $D_{\sigma 1}$ w.r.t. entailment.

In fact, the first rule corresponds to the particular case in which $SP2$ and $SP1$ are specifications $D_{\sigma 2} A_{\Phi 2} \epsilon_{\Sigma H 2}$ and $A_{\Phi 1} \epsilon_{\Sigma H 1}$, and P is a parameterized specification $(\lambda\, X :\downarrow \sigma 1.\ D_\sigma A_\Phi T_{\sigma 1;\sigma} X)$. Nevertheless, for specifications in $DATU$ the normalization results in theorem 4.2.2 allow the first rule to be used without loss of generality.

The drawback of the $\exists^2$-strategy is the need to guess Φ, or P in general. In the following sections a technique is presented for taking Φ to be a part of $\Phi 1$ in cases such as `count` in the specification of `mre`. But, in general, guessing Φ amounts to implementing the hidden part of the consequent specification in order to prove correctness, and this is indeed a high price for not following the specification bias.

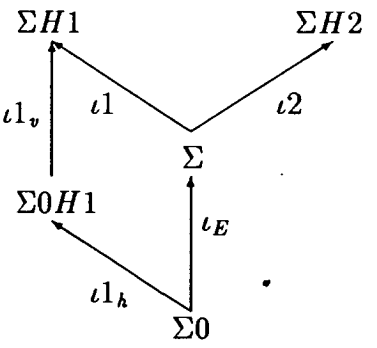
We conclude that given the general task of proving entailments between specifications with hidden parts, naïve techniques fail to handle examples such as the one in section 4.1.1, and indeed there are very few cases where entailment can be proven. Alternatively, a complete strategy exists but it requires the prover to do a lot of work. In the following section we present some restrictions to the general case needed to handle all the examples presented, using a method like the one sketched for the example at the end of section 4.1.1.

4.3 Persistent and independent enrichments

The main difference between the general case as it has been presented in the last section and the examples, is the existence of a structure in the axioms of the examples which allows us to distinguish some of them as *defining the hidden part*

and some others as *using the hidden part in order to define a visible enrichment.* Therefore, we shall assume that our task carries such a structure.

In general, we shall distinguish within the visible part, Σ, those visible symbols in terms of which the hidden symbols are defined, $\Sigma 0$, the inclusion along the hidden enrichment, $\iota 1_h$, and finally the inclusion along the (visible) enrichment $\iota 1_v$ on *top* of the hidden enrichment.



In the example $SP1$ in figure 4–1, $\Sigma 0$ is the signature of *ListNat*, $\iota 1_h$ adds `count`, and $\iota 1_v$ adds `mre` producing the whole signature $\Sigma H1$, which is then constrained by hiding `count` along $\iota 1$.

In general, we shall require signature morphisms $\iota 1_h$, $\iota 1_v$, ι_E and $\iota 1$ as in the figure to form a pushout diagram. In the case where all morphisms are inclusions, $\Sigma H1$ is the union of Σ and $\Sigma 0H1$, and $\Sigma 0$ is their intersection.

Such a structure of the signature must be imposed upon the sentences so that, instead of having a single set of sentences $\Phi 1$ over $\Sigma H1$ and consequent $D_{\iota 1}\Phi 1 \epsilon_{\Sigma H1}$, we have three sets of sentences:

- $\Phi 1_0$: set of sentences over $\Sigma 0$.
- $\Phi 1_h$: set of sentences over $\Sigma 0H1$ specifying the hidden part.
- $\Phi 1_v$: set of sentences over $\Sigma H1$ specifying a visible enrichment using the hidden part.

Therefore the task becomes:

Task 2: $D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H2} \models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$

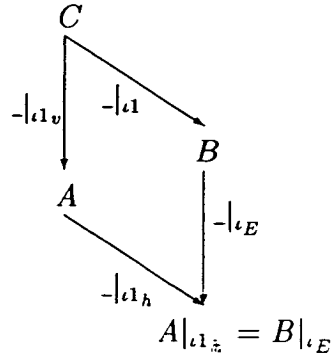
Now, we must establish the conditions under which such a decomposition capture the informal ideas of *specifying the hidden part* and *using the hidden part for*

specifying a visible enrichment. In order to do that, we define the concepts of *persistent* and *independent* enrichment.

Definition 4.3.1 *An* **enrichment** *by Σ_e-axioms Φ_e w.r.t. a signature inclusion $\iota_e : \Sigma \hookrightarrow \Sigma_e$ is a parameterized specification $(\lambda X : \Sigma.\ A_{\Phi_e} T_{\iota_e} X)$ which produces a specification over Σ_e when it is applied to a specification over Σ. Enrichments are usually written as $(A_{\Phi_e} T_{\iota_e})$.*

Given a signature inclusion $\iota_e : \Sigma \hookrightarrow \Sigma_e$ and a set of Σ_e-sentences Φ_e, an enrichment $(A_{\Phi_e} T_{\iota_e})$ (or $\lambda X : \Sigma.\ A_{\Phi_e} T_{\iota_e} X$) is **persistent** *w.r.t. a Σ-specification SP iff $(A_{\Phi_e} T_{\iota_e})$ is persistent SP and $\langle id_\Sigma, \iota_e \rangle$, i.e. for all $A \in Mod[SP]$ there is a model $B \in Mod[(A_{\Phi_e} T_{\iota_e})SP]$ which extends A, $B|_{\iota_e} = A$.*

Given signature morphisms ι_E, $\iota 1_v$, $\iota 1$ and $\iota 1_h$ forming a pushout diagram as in the figure above, and sets of sentences $\Phi 1_h$ over $\Sigma 0 H1$ and $\Phi 1_v$ over $\Sigma H1$, an enrichment $(A_{\Phi 1_h} T_{\iota 1_h})$ is **independent** *w.r.t. a $\Sigma 0$-specification SP and an enrichment $(A_{\Phi 1_v} T_{\iota 1_v})$ iff for all $A \in Mod[(A_{\Phi 1_h} T_{\iota 1_h})SP]$ and $B \in Mod[D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h})SP]$ such that $B|_{\iota_E} = A|_{\iota 1_h}$, there exists $C \in Mod[(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h})SP]$ such that $C|_{\iota 1_v} = A$ and $C|_{\iota 1} = B$.*



Remarks

1. An enrichment to a presentation is a theory extension, and a persistent enrichment is a particular kind of conservative extension, in the terminology of [Kei 77, BP 90].

2. If $\iota 1_h$ and $\iota 1_v$ are inclusions, we can require $\iota 1$ and ι_E to be inclusions as well without loss of generality. Moreover, if the pushout of inclusions exists, inclusions $\iota 1$ and ι_E are unique given $\iota 1_h$ and $\iota 1_v$.

3. Often translations T_ι where ι is an inclusion can be inferred from the context. In this case, they can be omitted so that Φ_e is used instead of $(A_{\Phi_e}\, T_{\iota_e})$ for arbitrary enrichments. Moreover, we write $[\Phi 1_h]$ to denote hidden enrichments so that $D_{\iota 1}(A_{\Phi 1_v}\, T_{\iota 1_v})(A_{\Phi 1_h}\, T_{\iota 1_h})SP$ can be written as $\Phi 1_v[\Phi 1_h]SP$.

   We shall use this short notation in some explanations, though the full notation is kept in the results and their proofs.

4. Since *Mod* is assumed to preserve finite co-limits, according to the amalgamation lemma, there is a unique model C which is the amalgamated union $A \oplus B$ (w.r.t. $\_|_{\iota 1_h}$ and $\_|_{\iota_E}$). Therefore, we can define independent enrichments by saying that: $\Phi 1_h$ *is independent w.r.t. SP and* $\Phi 1_v$ *iff the class of models of* $\Phi 1_v \Phi 1_h SP$ *is closed under amalgamated unions between models of* $\Phi 1_h SP$ *and models of* $\Phi 1_v[\Phi 1_h]SP$.

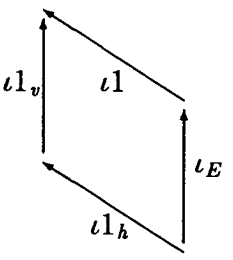
From an intuitive point of view, persistency of a hidden enrichment $\Phi 1_h$ w.r.t. a specification SP ensures that no models of SP is indirectly excluded by some requirements of $\Phi 1_h$, *i.e.* $[\Phi 1_h]SP$ is equivalent to SP or, as expressed in the $\exists^2$-strategy, $SP \models [\Phi 1_h]SP$. Independence of a hidden enrichment $\Phi 1_h$ w.r.t. specification SP and a later visible enrichment $\Phi 1_v$ means that the choice of a particular model in the hidden part does not exclude any of the models of the overall specification $\Phi 1_v[\Phi 1_h]SP$.

In next section we shall see how these conditions relate to entailment proofs, but we show first how independence is related to persistency.

Independence versus persistency

We might think that a more *natural* condition to place upon a visible enrichment on top of a hidden enrichment is to require persistency for the visible enrichment, at least w.r.t. the hidden symbols. This notion is formalized as relative persistency.

Definition 4.3.2 *Given signature morphisms ι_E, $\iota 1_v$, $\iota 1$ and $\iota 1_h$ forming a pushout diagram as in the figure, an enrichment $(A_{\Phi 1_v} T_{\iota 1_v})$ is* **relatively persistent** *w.r.t. a previous enrichment $(A_{\Phi 1_h} T_{\iota 1_h})$ and a specification SP, iff for all $A \in Mod[(A_{\Phi 1_h} T_{\iota 1_h})SP]$ for which there exists $B \in Mod[D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h})SP]$ such that $B|_{\iota_E} = A|_{\iota 1_h}$, there exists $C \in Mod[(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h})SP]$ such that $C|_{\iota 1_v} = A$.*



This property states persistency of the second enrichment w.r.t. that part of the signature added during the first (hidden) enrichment. Technically, C extends A as in the definition of persistent enrichment, but now A is not an arbitrary model; hence, relative persistency is weaker that persistency. Intuitively, the axioms of $\Phi 1_v$ cannot further constrain the symbols to be hidden but can add new requirements on the visible symbols defined in SP.

It is not difficult to see that independence entails relative persistency.

Proposition 4.3.3 *If an enrichment $(A_{\Phi 1_h} T_{\iota 1_h})$ is independent w.r.t. a specification SP and a later enrichment $(A_{\Phi 1_v} T_{\iota 1_v})$, then $(A_{\Phi 1_v} T_{\iota 1_v})$ is relatively persistent w.r.t. $(A_{\Phi 1_h} T_{\iota 1_h})$ and SP.*

Proof Definitions 4.3.1 and 4.3.2 define independence and relative persistency in the context of four signature morphisms ι_E, $\iota 1_v$, $\iota 1$ and $\iota 1_h$ forming a pushout

diagram as in the figure above, and sets of sentences $\Phi 1_h$ over $\Sigma 0 H1$ and $\Phi 1_v$ over $\Sigma H1$.

By definition of independence, for all $A \in Mod[(A_{\Phi 1_h} T_{\iota 1_h})SP]$ and $B \in Mod[D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h})SP]$ such that $B|_{\iota_E} = A|_{\iota 1_h}$, there exists $C \in Mod[(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h})SP]$ such that $C|_{\iota 1_v} = A$, hence relative persistency holds.

□

But, independence is more than relative persistency of the visible enrichment. Consider the specifications:

```
Ex11 =    sorts s
          operations a,b:  s
          axioms a≠b
          end

Ex12 =    Enrich Ex11 by
          Hidden
            sorts
            operations h:  s
            axioms
          in
            sorts
            operations v:  s
            axioms v=h
          end
```

Here, both enrichments are persistent but the hidden enrichment fails to be independent because there exists an algebra $\{a = 1, b = 2, v = 1, h = 2\}$ which is not a model of the overall specification ($Ex12$ without considering h hidden) but which can be obtained as the amalgamated union of a model of $Ex12$ $\{a = 1, b =$

$2, v = 1\}$ and a model of the $Ex11$ with the hidden enrichment $\{a = 1, b = 2, h = 2\}$.

Since neither relative persistency nor persistency (as in $Ex12$) are enough to guarantee independence, we may think that persistency is weaker than independence as relative persistency has been shown to be. However, independence of $(A_{\Phi 1_h} T_{\iota 1_h})$ w.r.t. SP and $(A_{\Phi 1_v} T_{\iota 1_v})$ does not guarantee persistency of $(A_{\Phi 1_v} T_{\iota 1_v})$ w.r.t. $(A_{\Phi 1_h} T_{\iota 1_h})SP$. Consider for example:

```
Ex21 =    sorts s
          operations a,b:  s
          axioms
          end

Ex22 =    Enrich Ex21 by
          Hidden
            sorts
            operations h:  s
            axioms
          in
            sorts
            operations v:  s
            axioms
              v=a
              v=b
          end
```

In this case the hidden enrichment is independent: given models for $\{s;\ a, b, v\}$ and $\{s;\ a, b, h\}$ if $v = a = b$ in the first and both agree in $\{s;\ a, b\}$ then their amalgamated union is a model of the overall specification ($Ex22$ without considering h hidden). But the visible enrichment is not persistent since it requires $a = b$.

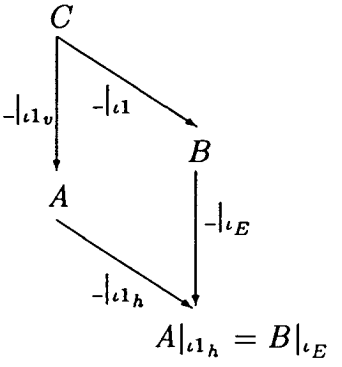
Summing up, independence and persistency are unrelated. Relative persistency is a weak notion of persistency which is necessary for independence.

Independence versus persistent functors

It is very common to find cases like count in $SP1$ where the hidden part is *totally* defined; *i.e.* $(A_{\Phi 1_h} T_{\iota 1_h})$ is a persistent functor over the models of SP. Then independence follows automatically:

Proposition 4.3.4 *Given a specification* $(A_{\Phi 1_h} T_{\iota 1_h})SP$ *such that for all* $A \in Mod[SP]$ *there exists exactly one (up to isomorphism)* $B \in Mod[(A_{\Phi 1_h} T_{\iota 1_h})SP]$ *such that* $B|_{\iota 1_h} = A$, *then for every enrichment* $(A_{\Phi 1_v} T_{\iota 1_v})$ *and morphisms* $\iota 1$ *and* ι_E *forming a pushout diagram as in the earlier figure,* $(A_{\Phi 1_h} T_{\iota 1_h})$ *is independent w.r.t.* SP *and* $(A_{\Phi 1_v} T_{\iota 1_v})$.

Proof For any given model B of $D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h})SP$, there is (up to isomorphism) a unique $A \in Mod[(A_{\Phi 1_h} T_{\iota 1_h})SP]$ such that $A|_{\iota 1_h} = B|_{\iota_E}$. On the other hand, by definition of $D_{\iota 1}$ there exists a $C \in Mod[(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h})SP]$ such that $C|_{\iota 1} = B$. Since $C|_{\iota 1_h ; \iota 1_v} = A|_{\iota 1_h}$, by the uniqueness of A we conclude that C is also an extension of A, $C|_{\iota 1_v} = A$. □



The relation between persistent functors and amalgamated unions is not new. If we consider a specification language where specifications only denote isomorphic classes of models and hidden enrichments are required to be persistent functors (such as basic and parameterized specifications in the initial approach [EM 85]) the extension lemma establishes a similar connection.

Persistency in some particular cases

If we consider models to be algebras without empty carriers and signature morphisms to be inclusions, persistency of a enrichment $(A_{\Phi 1_h} T_{\iota 1_h})$ w.r.t. SP depends exclusively on the relationship between SP and the presentation $\Phi 1_h$. Hence, enrichments $(A_\emptyset T_\iota)$ for an inclusion ι are persistent w.r.t. any specification SP over the appropriate signature.

On the other hand, if a non-injective morphism σ is considered, persistency of $(A_\Phi T_\sigma)$ w.r.t. SP also depends on σ. For example, let $s1$ and $s2$ be two sorts which are mapped to the same sort $s12$ by an algebraic signature morphism σ; then no model $A \in Mod[SP]$ with different carriers $|A|_{s1}$ and $|A|_{s2}$ can be extended to a model of $T_\sigma SP$, or to a model of $(A_\Phi T_\sigma)SP$ for any Φ.

Similarly, if models of SP include algebras with empty carriers, extending the signature of the specification may add constants and functions preventing some carriers from being empty, hence violating persistency.

4.4 Inheriting strategy — soundness

As we noticed in section 4.1.1, some proofs of specification entailment can be carried out by *importing* the hidden part of the consequent into the antecedent and then proving that the visible part of the consequent follows from the *enriched* antecedent. The machinery introduced in the last section provides the means to formalize such a proof strategy.

The notation corresponding to task 2 is enhanced with a new pushout as in figure 4–6, so that signature ΣH combines the hidden symbols of the antecedent and the consequent.

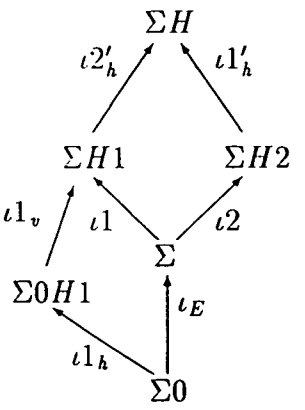


Figure 4–6: Signature morphisms in Task 2.

Now for task 2

$$D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2} \models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$$

abbreviated as $D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2} \models \Phi 1_v[\Phi 1_h]\Phi 1_0$, we can offer the following rule:

Theorem 4.4.1 (Inheriting strategy) *Given presentations* $\Phi 1_0$ *over* $\Sigma 0$, $\Phi 1_h$ *over* $\Sigma 0H1$, $\Phi 1_v$ *over* $\Sigma H1$ *and* $\Phi 2$ *over* $\Sigma H2$, *the rule*

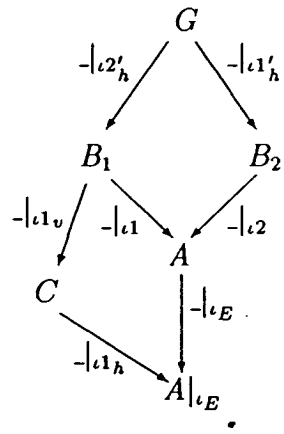
$$\frac{\Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v \qquad \Phi 2 \models_{\Sigma H} \Phi 1_0}{D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2} \models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}}$$

is sound, provided the signature morphisms are arranged according to the two pushouts of figure 4-6 and $(A_{\Phi 1_h} T_{\iota 1_h})$ *is a persistent enrichment w.r.t.* $A_{\Phi 1_0} \epsilon_{\Sigma 0}$.

Proof For an inconsistent presentation $\Phi 2$ the theorem holds trivially.

Otherwise, let A be a model of $D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2}$; then there exists $B_2 \models \Phi 2$ such that $B_2|_{\iota 2} = A$, and $C \in Mod[(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}]$ such that $C|_{\iota 1_h} = A|_{\iota_E}$. C is guaranteed to exist because, by the rule's second premise $\Phi 2 \models_{\Sigma H} \Phi 1_0$ so $A|_{\iota_E} \models \Phi 1_0$ and, by persistency of $(A_{\Phi 1_h} T_{\iota 1_h})$, $A|_{\iota_E}$ can be extended to an algebra C as required.

Now, we consider amalgamated unions $B_1 = A \oplus C$ and $G = B_1 \oplus B_2$ according to the diagram at the right.



By the satisfaction condition it can be shown that $G \models_{\Sigma H} \Phi 2 \cup \Phi 1_h$. Then from the rule's first premise $G \models \Phi 1_v$, therefore $G|_{\iota 2'_h} \in Mod[(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}]$ and finally $A = G|_{\iota 1;\, \iota 2'_h} \in Mod[D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}]$. □

Persistency is a sufficient condition for the soundness of the rule but it is not necessary. That is, a correct entailment

$$D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2} \models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$$

may satisfy the premises of the rule and $(A_{\Phi_h} T_{\iota 1_h})$ not be persistent w.r.t. $A_{\Phi 1_0} \epsilon_{\Sigma 0}$.

The crucial point for considering persistency as an adequate condition has to do with the quantification it involves, *i.e.* which parts of which specifications are involved in the condition for soundness among $\Phi 1_0, \Phi 1_h, \Phi 1_v$ and $\Phi 2$. For example, let

$$\forall \Phi 1_0, \Phi 1_h, \Phi 1_v, \Phi 2.$$
$$Condition(\Phi 1_0, \Phi 1_h, \Phi 1_v, \Phi 2) \iff Rule(\Phi 1_0, \Phi 1_h, \Phi 1_v, \Phi 2)$$

mean that *Condition* is a necessary and sufficient condition for soundness of the rule. In our case we have (by theorem 4.4.1 and theorem 4.4.3 below)

$$\forall \Phi 1_0, \Phi 1_h.$$
$$Persistent\ (A_{\Phi 1_h} T_{\iota 1_h})\ wrt\ A_{\Phi 1_0} \epsilon_{\Sigma 0} \iff \forall \Phi 1_v, \Phi 2.\ Rule(\Phi 1_0, \Phi 1_h, \Phi 1_v, \Phi 2)$$

which means that persistency is adequate when we look for a condition on $\Phi 1_0$ and $\Phi 1_h$ independently of what $\Phi 1_v$ and $\Phi 2$ might be. From a methodological point of

view, this is satisfactory since we would expect that specifications (consequents) are written before their implementations (antecedents) and that inner enrichments are written before outer ones; thus, at the time of writing the specification of the hidden part $\Phi 1_h$, only $\Phi 1_0$ is known.

Other sufficient conditions for soundness can be found which involve different parts of the specifications, *e.g.*

Proposition 4.4.2 (Inheriting strategy 2) *Given presentations* $\Phi 1_0$ *over* $\Sigma 0$, $\Phi 1_h$ *over* $\Sigma 0H1$, $\Phi 1_v$ *over* $\Sigma H1$ *and* $\Phi 2$ *over* $\Sigma H2$, *the rule*

$$\frac{\Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v \qquad \Phi 2 \models_{\Sigma H} \Phi 1_0}{D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2} \models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}}$$

is sound, provided the signature morphisms are arranged according to the two pushouts of figure 4-6 and $(A_{\Phi 1_h} T_{\iota 1_h})$ *is a persistent enrichment w.r.t.* $D_{\iota_E; \iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2}$.

Proof Looking at the proof of theorem 4.4.1 we realize that only models $A|_{\iota_E}$ for $A \in Mod[D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2}]$ need to be extended to $\Sigma 0H1$, thus persistency of $(A_{\Phi 1_h} T_{\iota 1_h})$ w.r.t. $D_{\iota_E; \iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2}$ guarantees soundness. □

This condition is weaker than persistency of $(A_{\Phi 1_h} T_{\iota 1_h})$ w.r.t. $A_{\Phi 1_0} \epsilon_{\Sigma 0}$ since it requires persistency over fewer models as $Mod[D_{\iota_E; \iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2}] \subseteq Mod[A_{\Phi 1_0} \epsilon_{\Sigma 0}]$. It is also an analogous condition to that required in the $\exists^2$-strategy which generalizes the inheriting strategy. Nevertheless, this soundness condition asks for requirements on the antecedent (implementation side), and that may not methodologically desirable. This condition is used in the following chapters where hiding is combined with abstraction and reachability constraints; however, it is not an adequate condition to consider when designing specifications which are to appear as consequents in specification entailment proofs.

To conclude we prove, as promised above, that persistency is indeed necessary as a condition on $\Phi 1_0$ and $\Phi 1_h$.

Theorem 4.4.3 *Given finite presentations* $\Phi 1_0$ *over* $\Sigma 0$ *and* $\Phi 1_h$ *over* $\Sigma 0 H 1$, *if the rule*

$$\frac{\Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v \qquad \Phi 2 \models_{\Sigma H} \Phi 1_0}{D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2} \models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}}$$

is sound for all $\Phi 2$ *and* $\Phi 1_v$, *then* $(A_{\Phi 1_h} T_{\iota 1_h})$ *is persistent w.r.t.* $A_{\Phi 1_0} \epsilon_{\Sigma 0}$, *provided signatures and morphisms are arranged as in figure 4–6.*

Proof Given a non-persistent $(A_{\Phi 1_h} T_{\iota 1_h})$ w.r.t. $A_{\Phi 1_0} \epsilon_{\Sigma 0}$, there exists a model of $A_{\Phi 1_0} \epsilon_{\Sigma 0}$ which cannot be extended to a model of $(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$, hence

$$A_{\Phi 1_0} \epsilon_{\Sigma 0} \not\models D_{\iota 1_h}(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$$

then if we choose $\Phi 2$ and $\Phi 1_v$ such that $\Phi 2 = (\iota_E; \iota 2)(\Phi 1_0)$ and $\Phi 1_v = \emptyset$ we get

$$D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2} \not\models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$$

while the premises of the rule hold trivially. □

So far, we have presented the inheriting strategy as a rule for proving specification entailments involving consequents with a persistent hidden enrichment. Now, we shall show that the inheriting strategy is *good enough* for proving all correct specification entailments involving consequents with an independent hidden enrichment.

4.5 Inheriting strategy — completeness

In this section we show how the inheriting strategy is sufficient for proving refinements of specifications with an independent hidden enrichment. Later the converse will be proven as well, confirming from a proof-theoretic standpoint that independence is not just intuitively reasonable but also very convenient.

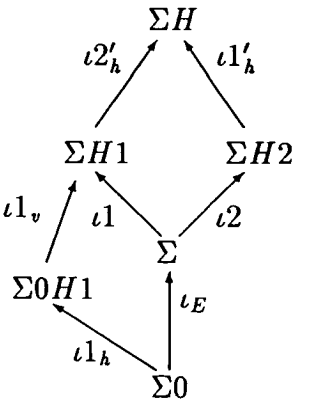
Theorem 4.5.1 *Given a correct entailment*

$$D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2} \models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$$

where $(A_{\Phi 1_h} T_{\iota 1_h})$ *is independent w.r.t.* $A_{\Phi 1_0} \epsilon_{\Sigma 0}$ *and* $(A_{\Phi 1_v} T_{\iota 1_v})$*, then it is always the case that*

$$\Phi 2 \models_{\Sigma H} \Phi 1_0 \quad \text{and} \quad \Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v$$

provided signatures and morphisms are arranged as in the figure on the right.



Proof The first entailment (1) $\Phi 2 \models_{\Sigma H} \Phi 1_0$ holds from the composition of the two following entailments:

$$D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2} \models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$$

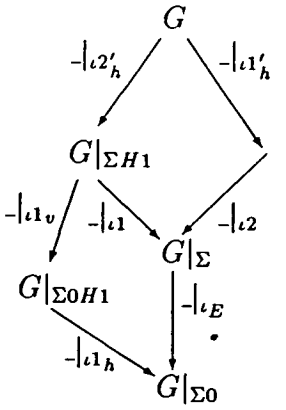
$$D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0} \models T_{\iota_E} A_{\Phi 1_0} \epsilon_{\Sigma 0}$$

where the first entailment is an assumption of the lemma and the second holds trivially.

In order to prove that $\Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v$, we assume the contrary (2) $\Phi 2, \Phi 1_h \not\models_{\Sigma H} \Phi 1_v$ and prove that the hidden enrichment cannot be independent.

By (1) and (2) there exists a ΣH-algebra G such that

$$G \models_{\Sigma H} \Phi 1_0 \cup \Phi 2 \cup \Phi 1_h \qquad G \not\models_{\Sigma H} \Phi 1_v$$



Taking the appropriate reducts it is clear that

$$G|_{\Sigma 0 H1} \in Mod[(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}]$$

$$G|_{\Sigma} \in Mod[D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2}]$$

$$G|_{\Sigma H1} \notin Mod[(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}]$$

and since the entailment is correct we can also conclude that

$$G|_{\Sigma} \in Mod[D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}]$$

Now, taking the amalgamated union $G|_{\Sigma 0 H1} \oplus G|_{\Sigma} = G|_{\Sigma H1}$ we come to a contradiction with independence, since according to it $G|_{\Sigma H1}$ should be a model of $(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$ but it is not. □

With theorems 4.4.1 and 4.5.1 we have the basis for a sound and complete inference system for specifications with persistent and independent hidden enrichments. Moreover, we give another theorem justifying the choice of independence as an *adequate* property w.r.t. the completeness of the rule, just as we gave theorem 4.4.3 to justify the choice of persistency w.r.t. soundness. We start with a new definition:

Definition 4.5.2 *A model A of a specification SP is* **abstract implementable** *iff there exists a flat specification $A_\Phi \epsilon_\Sigma$ such that $A_\Phi \epsilon_\Sigma \models SP$ and $A \in Mod[A_\Phi \epsilon_\Sigma]$.*

Abstract implementability heavily depends on the expressiveness of the logical system used in specifications. In FOLEQ we cannot expect it to hold in general, but if we allow code as axioms, then all computable models are abstract implementable.

In the following theorem, the necessity of independence as a condition for completeness depends critically on the abstract implementability of one model.

Theorem 4.5.3 *Given finite presentations* $\Phi 1_0$ *over* $\Sigma 0$, $\Phi 1_h$ *over* $\Sigma 0 H1$ *and* $\Phi 1_v$ *over* $\Sigma H1$, *if* $D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H2} \models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$ *implies that* $\Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v$ *for all finite presentations* $\Phi 2$ *over* $\Sigma H2$, *then* $(A_{\Phi 1_h} T_{\iota 1_h})$ *is independent w.r.t.* $A_{\Phi 1_0} \epsilon_{\Sigma 0}$ *and* $(A_{\Phi 1_v} T_{\iota 1_v})$, *provided signatures and morphisms are arranged as in figure 4-6 and all models of* $D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$ *are abstract implementable.*

Proof Given a specification $(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$ such that $(A_{\Phi 1_h} T_{\iota 1_h})$ is not independent w.r.t. $A_{\Phi 1_0} \epsilon_{\Sigma 0}$ and $(A_{\Phi 1_v} T_{\iota 1_v})$, there exist A and C such that

$$C \in Mod[(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}] \quad A \in Mod[D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}]$$

$$A|_{\Sigma 0} = C|_{\Sigma 0}$$

$$A \oplus C \notin Mod[(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}]$$

But, if A is abstract implementable there exists a flat specification $A_{\Phi_A} \epsilon_{\Sigma}$ such that

$$A \in Mod[A_{\Phi_A} \epsilon_{\Sigma}] \quad A_{\Phi_A} \epsilon_{\Sigma} \models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$$

and by the amalgamation lemma $A \oplus C$ must satisfy $\Phi 1_h$, $\Phi 1_0$ and Φ_A

$$A \oplus C \models_{\Sigma H} \Phi 1_h \cup \Phi_A$$

but, since $A \oplus C$ is not a model of $(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$ we know that

$$A \oplus C \not\models_{\Sigma H} \Phi 1_v$$

hence, $\Phi_A, \Phi 1_h \not\models_{\Sigma H} \Phi 1_v$.

Finally, taking $\Phi 2$ to be $\iota 2(\Phi_A)$ we have both

$$D_{\iota 2} A_{\Phi_2} \epsilon_{\Sigma H2} \models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$$

and

$$\Phi 2, \Phi 1_h \not\models_{\Sigma H} \Phi 1_v$$

□

The fact that A might be not abstract implementable does not matter for our purpose. We are only concerned with showing that no weaker condition than independence can be required in order to obtain completeness for the rule. Then the above lemma must be understood as saying: *Independence is a necessary condition for completeness, for an arbitrary logical system.*

Independence is not necessary for the success of the inheriting strategy. In fact, for any consequent $D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$ where independence does not hold, there exist an antecedent $A_{\Box} \epsilon_{\Sigma}$ where $\Box$ is the contradiction, such that $A_{\Box} \epsilon_{\Sigma} \models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$ can be proven using the inheriting strategy.

Nevertheless, in practice, the lack of independence of $(A_{\Phi 1_h} T_{\iota 1_h})$ w.r.t. $A_{\Phi 1_0} \epsilon_{\Sigma 0}$ and $(A_{\Phi 1_v} T_{\iota 1_v})$ arises because some elements defined in the visible enrichment are "closely" dependent on the interpretation of the hidden elements, as in specification *Ex*12 above. In such cases, *each* model of the visible part corresponds to only *one* or a *few* possible models of the hidden part; therefore, *each visible model* excludes some possible interpretations of the hidden symbols. In this situation, the models of the antecedent $Mod[D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2}]$, unless empty, contain a model A for which there exists a model $C \in Mod[(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}]$ such that $A|_{\iota_E} = C|_{\iota 1_h}$ but $A \oplus C \not\models_{\Sigma H} \Phi 1_v$ as in the proof above, hence the inheriting strategy fails.

In conclusion, although independence is not necessary from a formal point of view, it is necessary nearly always in practice.

4.6 Proving $k{+}ASL \models DATU$

The results for task 2 are good but it remains to be seen how they can be used in a more general context such as $k{+}ASL \models DATU$.

We start by relaxing the form of the antecedent specification from $D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2}$ to an arbitrary specification over $k{+}ASL$.

4.6.1 Generalizing the antecedent

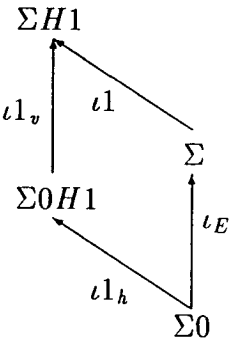
Unlike specifications in $DATU$, it is not always clear how an arbitrary specification $SP2$ in $k{+}ASL$ can be reduced to an equivalent specification of the form $D_{\iota 2} A_{\Phi 2} \epsilon_{\Sigma H 2}$. Sometimes, we can expect to obtain a sound theory $\Phi 2$ w.r.t. $SP2$ such that $\Phi 2, \Phi 1_h \models_{\Sigma H} \Phi 1_v$ and $\Phi 2 \models_{\Sigma H} \Phi 1_0$. Then, by soundness of the inheriting strategy (where $\iota 2$ is the identity on Σ) it follows that:

$$SP2 \models A_{\Phi 2} \epsilon_{\Sigma} \models D_{\iota 1}(A_{\Phi 1_v} T_{\iota 1_v})(A_{\Phi 1_h} T_{\iota 1_h}) A_{\Phi 1_0} \epsilon_{\Sigma 0}$$

Alternatively, we can generalize the inheriting strategy so that the hidden enrichment can be inherited by $SP2$ leading to an enriched antecedent

$$(A_{\iota 1_v(\Phi 1_h)} T_{\iota 1}) SP2$$

This generalized strategy can be presented leaving premises in the form of specification entailments, as in the $\exists^2$-strategy.



Proposition 4.6.1 *Given presentations* $\Phi 1_0$ *over* $\Sigma 0$, $\Phi 1_h$ *over* $\Sigma 0 H1$, $\Phi 1_v$ *over* $\Sigma H1$ *and a specification* $SP2$ *over* Σ, *the rule*

$$\frac{(A_{\iota 1_v(\Phi 1_h)}\, T_{\iota 1})SP2 \models (A_{\Phi 1_v}\, T_{\iota 1_v})(A_{\Phi 1_h}\, T_{\iota 1_h})A_{\Phi 1_0}\epsilon_{\Sigma 0} \qquad SP2 \models D_{\iota 1}(A_{\iota 1_v(\Phi 1_h)}\, T_{\iota 1})SP2}{SP2 \models D_{\iota 1}(A_{\Phi 1_v}\, T_{\iota 1_v})(A_{\Phi 1_h}\, T_{\iota 1_h})A_{\Phi_0}\epsilon_{\Sigma 0}}$$

is sound.

Soundness follows immediately from the soundness of the $\exists^2$-strategy. On the other hand, completeness considerations like those exhibited by the inheriting strategy in the last section are not applicable.

4.6.2 Generalizing the consequent

Similarly we can generalize each component of the consequent specification so that $D_{\iota 1}(A_{\Phi 1_v}\, T_{\iota 1_v})(A_{\Phi 1_h}\, T_{\iota 1_h})A_{\Phi 1_0}\epsilon_{\Sigma 0}$ is substituted by a specification in $\lambda + DATU$ of the form

$$D_{\iota 1}\, P_v\, P_h\, SP1$$

where parameterized specifications P_v for $(\lambda\, X : \Sigma 0 H1.\ P_v(X))$ and P_h for $(\lambda\, X : \Sigma 0.\ P_h(X))$ take the place of the visible and hidden enrichment respectively, and the specification $SP1$ takes the place of $A_{\Phi 1_0}\epsilon_{\Sigma 0}$.

Applying the $\exists^2$-strategy we obtain a rule as follows:

$$\frac{P'_h SP2 \models P_v\, P_h\, SP1 \quad SP2 \models D_{\iota'}\, P'_h\, SP2}{SP2 \models D_{\iota 1}\, P_v\, P_h\, SP1}$$

where P'_h for $(\lambda\, X : \Sigma.\ P'_h(X))$ is an arbitrary parameterized specification and ι' is a signature morphism $\iota' : Sig[SP2] \to Sig[P'_h SP2]$.

We can say that the inheriting strategy is being used when P'_h is an "extension" of P_h. It is not the purpose of this section to discuss in general the *extensibility* of parameterized specifications which has been a popular topic of discussion in the initial approach [TWW 78, EM 85, Ore 87] and also in *ASL* [SST 90]. Here, we

just consider that the inheriting strategy is a particular instance of the $\exists^2$-strategy where P'_h is obtained by *extending* P_h from parameter specifications over $\Sigma 0$ to parameter specifications over Σ.

4.6.3 *DATU** consequents

Despite the previous generalizations it is clear that consequents will not always have a form like:

$$D_{\iota 1} P_v P_h SP1$$

Among the specifications in *DATU* we shall distinguish those which restrict the use of D_σ to the context of persistent and independent hidden enrichments and call that sub-language *DATU**.

Using the $\exists^2$-strategy (described in section 4.2) and the rules reducing arbitrary specifications in *DATU* to the form $D_\iota A_{\Phi \epsilon \Sigma}$ (see theorem 4.2.2) we have a complete inference system for $DATU \models DATU$, provided $\vdash^{\mathbf{I}}$ is sound and complete and persistency can be decided.

Nevertheless, restricting to specification entailments in $DATU \models DATU^*$ has the advantage that they can be proven using the inheriting strategy instead of the $\exists^2$-strategy. This makes a very significant difference to provers which do not need to *guess* a convenient enrichment at each application of the $\exists^2$-strategy. Moreover, since persistency and independence of the hidden enrichments are implicitly required, $DATU \models DATU^*$ has a complete inference system.

The $\_^*$ notation will be used in other chapters below in order to require the same restrictions on the use of D_σ in *DATU** in more complex specification languages such as *ASL** and *k+ASL**.

U* includes only those specifications of the form (after
nalization) $\Phi 1_v [\Phi 1_h] \Phi 1_o$ with $\Phi 1_h$ being a persistent
independent hidden enrichment.
er this condition, DATU ⊨ DATU* has a complete inference
em.

4.7 Difficult cases

Inheriting the hidden part from the consequent to the antecedent is a fruitful strategy in those cases where a distinguished set of axioms defines the hidden part precisely enough. Persistency and independence ensure that. Nevertheless, when these side conditions are not satisfied we are driven into a more general strategy such as the $\exists^2$-strategy.

In this section some guidance is given for those cases where the consequent has the form

$$D_{\iota 1} P_v P_h SP1$$

but either persistency of P_h w.r.t. $SP1$ or independence of P_h w.r.t. $SP1$ and P_v fail.

4.7.1 Failing independence

Many times independence fails in specifications with a persistent hidden part. This is normally the case for specifications whose auxiliary functions or sorts are not defined with enough detail. We may not bother defining very precisely an auxiliary function because for each of its possible realizations there are some valid models of the visible part. However, this leads to a failure of independence. Consider for example, the specification of a scheduling function as in figure 4–7.

Although `scheduler` seems to be hidden, the function `next` gives some information about which choice of the function has been assumed. This dependency of a visible function on a hidden function which is not completely defined prevents the hidden enrichment from being independent.

```
Schdler =  Enrich ListNat by
           Hidden
             sorts
             operations
               scheduler:  listnat -> listnat
             axioms
               ...
               { Axioms requiring 'scheduler' to deliver
               some arbitrary permutation of the parameter.  }
               ...
           in
             sorts
             operations
               next:  listnat -> nat
             axioms
               ∀ l:listnat. next(l)=car(scheduler(l))
           end
```

Figure 4–7: Specification of a scheduler

If `scheduler` were completely defined as being, for example, a function to sort a list in descending order then the hidden enrichment would become independent automatically (see proposition 4.3.4).

On the other hand, if `next` is a less completely defined function, independence can hold without changing `scheduler`. For example, suppose the only requirement on `next(l)` is to yield a value of the list `scheduler(l)`:

$$\forall\ \texttt{l:listnat.}\ \texttt{next(l)} \in \texttt{scheduler(l)} = \texttt{true}$$

Now, the hidden enrichment is independent because no realization of `next` excludes a possible behaviour of `scheduler`.

Using the $\exists^2$-strategy amounts to requiring the completion of the definition of the hidden part as far as needed to attain independence.

Consider an antecedent $SP2$ where `next` produces the maximum of the list. The entailment $SP2 \models \mathit{Schdler}$ cannot be proven correct by the inheriting strategy. The lack of independence means that we are not inheriting enough axioms to complete the proof.

The $\exists^2$-strategy can be seen as a procedure which forces the prover to complete the definition of the hidden part as far as necessary to complete the proof. For example, choosing Φ in the $\exists^2$-strategy to be the definition of `scheduler` as a sorting function in descending order would suffice. In general Φ can be chosen to be $\Phi 1_A$ — as in the inheriting strategy — plus some extra axioms completing the definition of the hidden symbols.

In consequence, we can say that the hidden symbols can be used in the visible part insofar as they are adequately defined. If some hidden symbols are inadequately defined w.r.t. how they are used, the hidden enrichment lacks independence and the definition of the hidden symbols generally needs to be completed

during the proof of correctness (see discussion at the end of section 4.5 about independence being generally needed for the success of the inheriting strategy).

4.7.2 Failing persistency

Non-persistent hidden parts are a more serious problem although not so frequent in practice.

A non-persistent hidden part cannot be inherited to the antecedent because this could cause some of the antecedent's models to be excluded from consideration in the proof, leading to a situation where an incorrect entailment is proven correct.

Non-persistent hidden enrichments can be considered errors in the sense that the existence of some auxiliary functions and sorts should be *transparent* to the rest of the specification.

Applying the $\exists^2$-strategy amounts to asking for a new persistent definition, Φ, of the hidden part. In other words, such a strategy is equivalent to going back to the specification, writing a correct (persistent) definition, Φ, of the hidden part, leaving the old $\Phi 1_h$ as additional requirements on the visible part — added to $\Phi 1_v$ or $\Phi 1_0$ — and starting the proof of correctness according to the inheriting strategy.

Chapter 5

Abstracted specifications

5.1 Introduction

In this chapter we are concerned with specifications using abstraction and behavioural abstraction. Unlike the other SBO's in *ASL*, the definition of `abstract` has been controversial (c.f. [SW 83, ST 87]) as has also been the general notion of abstraction in the field of algebraic specifications (c.f. [GGM 76, Rei 87, MG 85, NO 88]) and its relationship to the notion of abstract implementation (c.f. [Hoa 72, Ehr 81, EKMP 82, BBC 86]).

Since early work in [GGM 76], the interpretation of an abstract data type as an algebra (abstracted) up to isomorphism has been seriously debated. The problem is essentially linked to the distinction between observable and non-observable sorts.

If all sorts are observable then abstraction up to isomorphism is appropriate in the sense that it fully captures the idea of a system seen as a black box. The sorts of the algebra represent the different kinds of input/output values while the functions (their graphs) represent the relation of the input to the output.

However, algebras, when denoting data types, contain at least one sort representing the values of the data type in question (distinguished phylum in [GH 78]). The values of this data type are, in general, abstract entities which do not correspond to any kind of input/output value (observable value for short), e.g. states

in an automaton. Abstraction w.r.t. isomorphism only admits as models of a data type algebras with isomorphic carriers. So, in the case of the specification of an automaton which recognizes a particular language, automata with different configurations of states cannot be models of the same specification, even if they recognize the same language.

In order to bring into algebraic specification an idea similar to equivalence of automata, the notion of *behaviour* of an algebra w.r.t. some observable sorts has been defined (among others in [GGM 76, MG 85]). Then, two algebras are behaviourally equivalent if they have the same behaviour[1], that is if they agree on all computations from observable sorts to observable sorts. Applying behavioural abstraction to a specification closes its class of models with respect to behavioural equivalence.

Behavioural abstraction can be generalized to non-algebraic institutions by explicitly indicating those sentences which are observable, as does `abstract` in [ST 87, ST 88a]. Then, behavioural abstraction becomes the particular case in which the observable sentences are all the equations between terms of observable sorts.

In this chapter we shall propose a new account for abstraction and in particular behavioural abstraction according to the following informal requirements.

1. Abstraction should be defined as an institution-independent operation, like the other SBO's in *ASL*.

---

[1]Most frequently, behavioural equivalence is defined without an explicit definition of behaviour.

2. Behavioural abstraction should be a particular instance of abstraction for the case in which models are algebras and observations are defined in terms of observable sorts.

3. The user should be able to decide if a sort is observable or not when defining a specification. Moreover, any specification can be further abstracted, whenever it is used in a bigger context, for example by turning some observable sorts into non-observable sorts.

4. Abstraction in *ASL* should be related to D_σ since both operations "hide", in an informal sense, some details of a specification which do not need to be implemented.

This is realized by considering an institution morphism from the *original* institution to another *abstracted* institution where $T_\sigma D_\sigma$ happens to mean abstraction in some cases or exactly what it meant in the original institution in others.

But before giving details of this approach, some others are surveyed showing which of the above requirements are not satisfied.

Historical survey

Without pretending to be exhaustive, a few approaches are sketched trying to capture the range of alternatives among specification languages with tight or loose semantics, and among the latter distinguishing those with a notion of institution from the rest.

Probably the simplest way to deal with behavioural abstraction is to ignore it, considering specifications to denote isomorphic classes of algebras and proposing a relation of abstract implementation as in [GTW 76, EKMP 82, Ehr 81]. In this

case, those models which are behaviourally equivalent to a model of the specification without being isomorphic to it, are expected to be valid implementations. For example, many algebras which fail to be models of a specification are among the valid implementations because there exists a homomorphism from them to a model of the specification (quotient step in [EKMP 82, Ehr 81]).

Often abstract implementation are such that by considering an appropriate restriction of the signature, taking a subalgebra and performing a quotient[2], any valid implementation can be turned into a model isomorphic to the initial model of the specification. Therefore, as in [Hoa 72], a value can be represented by several values in the implementation but never the other way round. In an example such as the automaton specification referred above, if it is the case that the automaton denoted by the initial model of the specification (up to isomorphism) is not minimal then some behaviourally equivalent models will not be valid implementations. Hence, this approach fails the second requirement.

Some other approaches genuinely concerned with behaviour and still faithful to initial semantics have been developed [Rei 81, Rei 87, MG 85, NO 88].

Reichel's work explicitly introduces abstraction via *canons of behaviour*. Essentially, two algebras are behaviourally equivalent if they have a common quotient via congruences which leave their observable carriers untouched. This definition takes good care of specifications such as the automaton example above, but junk in non-observable sorts must be preserved in the implementation. Hence the second requirement does not hold.

In [MG 85] a definition for behaviour of an algebra is presented as a new algebra whose operations are the observations of the former one, *i.e.* those sequences of

---

[2]A subalgebra must be chosen before taking the quotient. Reversing the order of the operations results in a weaker notion of implementation [EKMP 82].

old operations computing observable values from observable values (a new concise categorical definition will be given below). Behavioural abstraction is obtained by including as models of an abstracted specification all algebras having the same behaviour. In [Niv 87, NO 88] a similar setup is defined by changing the notion of homomorphism in such a way that isomorphism means behavioural equivalence. Both works lead to the same notion of behaviour and make it compatible with initiality. Nivela's work goes further, taking into account the most common constructions related to initial semantics, but what we consider more relevant is that in her approach behaviour is a concept embedded into a new institution instead of being defined separately. These approaches fail in the first requirement since their notions of behaviour are intimately linked to **EQ** (the standard institution of total algebras and equational logic).

Another way to avoid the problems posed by the initial approach of [GTW 76] is to consider the model designated by a specification to be not the initial one but the final one [GGM 76, Kam 83, Wan 79]. In this way a notion of implementation based on homomorphism will succeed in examples such as the automaton specification mentioned above since the final model is precisely the minimal automaton recognizing the required language. However, apart from technical problems related to the existence of final models, this approach violates the third requirement since there is no explicit way to apply or not to apply finality.

Specification languages with loose semantics fit more comfortably with the problem of defining behaviour. Since the models of a specification do not need to be isomorphic, one possibility is to change the definition of satisfaction for equations between non-observable values to mean equivalence w.r.t. all visible contexts; hence, the equality symbol means indistinguishability (see for instance [Hen 88, Gog 90]). Problems with behaviour and loose semantics usually appear when look-

ing at proof techniques or when trying to generalize the idea of abstraction to an arbitrary institution.

First, we consider cases such as [WB 89] where equality is treated as in standard first order logic. Following [Men 71], equality is an ordinary predicate symbol satisfying some extra axioms, namely, reflexivity, transitivity, symmetry and extensionality w.r.t. any well-formed formula. Normal models are those in which the equality symbol is interpreted as identity, but other models are possible. In [WB 89] multi-sorted first order logic with equality is used in this way, so that for many specifications their class of models is automatically closed up to behavioural equivalence. The advantages are proof-theoretical but from the point of view of abstraction this approach fails the third requirement since the user cannot decide in which sorts equality should mean identity, and it fails the second requirement as well since no model of an specification with an inequality $a \neq b$ can give the same value to a and b, even if a and b belong to a non-observable sort.

Other approaches making an explicit distinction between observable and non-observable sorts when using equations lead to [Gog 90] where this is concisely presented as a new institution for behaviour. The only inconvenience is its commitment to **EQ** which makes the first requirement fail.

Heading towards a more general definition of behaviour, [SW 83], [Wir 86] and [ST 87] propose to consider observable terms and observable sentences as an alternative to observable sorts. [Wir 86] accompanies generality with some proof theoretical results but only [ST 87] presents a institution independent definition: *Observational abstraction.* An approach based on an observability predicate *Obs()* such as [Wir 86] does not satisfy our third requirement properly since a given specification cannot be further abstracted later by any means, contrary to what an abstraction operation in [ST 87], a closure as proposed below, the parameter passing mechanism in [Gog 90] or even a *Non-Obs()* assertion can do.

An $\texttt{abstract}_{Obs}$ operation as in [ST 87] defines abstraction w.r.t. an arbitrary

set of sentences *Obs* called observations. Models of an abstracted specification $\texttt{abstract}_{Obs}SP$ include all those models over $Sig[SP]$ which satisfy the same observations as a model of SP. In particular, observable sorts can be defined in terms of observations by including enough sentences in *Obs* so that the carrier of an observable sort is completely determined up to isomorphism. This approach satisfies our requirements except for the last one. Unfortunately, this new operation does not relate (observational) abstraction to D_σ, the other abstraction mechanism in *ASL*.

In our proposal we aim to preserve the good properties of **abstract** but shifting the general problem of abstraction from the specification language into the underlying institution as in [NO 88] and [Gog 90], so that **abstract** becomes a derived operation: a particular instance of $T_\sigma D_\sigma$.

Proof techniques

Despite the multitude of approaches to behavioural abstraction, the available proof techniques for proving behavioural equivalence are essentially two: *explicit definition of congruences* and *proving agreement on all observations.*

Consider for example the classic implementation of a stack by an array and a pointer. The axiom

```
∀ S:stack; x:elem. pop(push(x,S)) = S
```

is not satisfied in the implementation as such. The first proof technique suggests to formalize the appropriate congruence on array-pointer pairs which corresponds to *stack equality*, e.g. all arrays with the same pointer which agree in the values of the array for indexes below the pointer are considered equal (see for example [Wan 82, WB 89]). The second proof technique substitutes the problematic axiom by its infinite set of observable consequences, *i.e.*

```
∀ S:stack; x:elem. top(pop(push(x,S))) = top(S)
∀ S:stack; x:elem. top(push(y,pop(push(x,S)))) = top(push(y,S))
...
```

and proves that all these hold in the implementation. By considering the recursive definition of observation, an induction schema can be used (see context induction in [Hen 88]).

All the approaches to behavioural abstraction can be classified depending on which proof technique they use, if any.

Model theoretical notions of implementation such as in [Hoa 72] and behavioural inclusion such as that in [Sch 87] (page 223) consider, by definition, that an abstract implementation must satisfy the (infinite) set of observable theorems of the specification. However, the main result of such approaches is precisely showing that an appropriate kind of congruence characterizes all such implementations. Hence, the latter method is normally used in the proofs.

For example, given a notion of observation such as equalities between closed terms of observable sorts, we obtain a notion of behavioural equivalence between algebras, *i.e.* two algebras are behaviorally equivalent if they satisfy the same observations. Then, a model-theoretical construction between algebras, *e.g.* correspondences in [Sch 87] and abstraction functions in [Hoa 72], is proposed in such a way that if the construction can be done between two algebras then they will be behaviourally equivalent. Defining an abstraction function amounts to using a congruence, *e.g.* in the case of stacks, all arrays with equal values below the pointer are mapped to the same *abstract stack*. Then, these characterizations can be understood as a proof of soundness of a technique such as the construction of an abstraction function (or correspondence).

Algebraic specification approaches such as [Ehr 81, EKMP 82] do not use an explicit notion of abstraction, but a notion of implementation which embeds it (see historical survey above). In this case a *quotient step* is included in the implementation notion. This quotient involves a congruence relating those values of the implementation which represent the same abstract value. Hence these approaches use *explicit definition of congruences.*

Also some loose approaches justify the use of congruences taking a first order interpretation of equality, *e.g.* [Wan 82, WB 89].

In contrast, those approaches taking definitions of abstraction in terms of observations or observable sorts without exploring any further, are committed to proving that antecedent and consequent agree on all observations. Therefore, notions of abstraction such as in [NO 88, MG 85, Gog 90] support, by default, *proving agreement on all observations.* We have also in this group: [ST 87] which gives an explicit rule for `abstract`, [Wir 86] which extends the calculus of equational implications with some explicit rules for handling the *Obs()* predicate, [Hen 88] which provides context induction to deal with an infinite number of observations and [Lin 87] which uses proofs by consistency.

In this chapter, since abstraction is treated as a particular meaning for $T_\sigma D_\sigma$ we shall revisit the proof techniques for hiding presented in the last chapter, which give rise to the two well-known proof techniques just surveyed.

5.2 Abstractions

At a very general level, we view the abstraction operation in *ASL* as $T_\sigma D_\sigma$ in an *abstracted* institution. In this sense, the *abstracted* institution is the enrichment of an (*original*) institution by some new signatures and signature morphisms and the corresponding models and sentences. This extra structure corresponds, intuitively, to abstract models (behaviours), abstract signatures (behaviour signatures) and observations.

Imagine the case of the institution for equational logic **EQ**. We can define a new institution **Beh(EQ)** where signatures are either standard algebraic signatures such as $\langle S, \Omega \rangle$ or abstracted signatures containing an explicit set of observable sorts, *e.g.* $\langle \langle S, \Omega \rangle, O \rangle$ where $O \subseteq S$, as is standard in behavioural abstraction. Models over the abstracted signatures are *behaviours* as defined in [MG 85] instead of algebras, and the reduct functor $\_|_o$ for a signature morphism $o : \langle \langle S, \Omega \rangle, O \rangle \rightarrow \langle S, \Omega \rangle$ maps each $\langle S, \Omega \rangle$-algebra to its behaviour. Sentences over an abstracted signature are those sentences of the original institution which cannot distinguish between two algebras with the same behaviour. The result is an *abstracting* institution morphism α : **EQ** $\rightarrow$ **Beh(EQ)** from the original institution **EQ** to the abstracted institution **Beh(EQ)**.

The abstraction operation in *ASL* is $T_o D_o$. Therefore, the essence of abstraction is the reduct functor $\_|_o$ which maps to the same behaviour — model of $Mod(\langle \langle S, \Omega \rangle, O \rangle)$ — all those algebras in $Alg(\langle S, \Omega \rangle)$ which are behaviourally equivalent. Thus, $T_o D_o$ performs the closure of the models of a specification SP over $\langle S, \Omega \rangle$ w.r.t. behavioural equivalence.

$$Mod[T_o D_o SP] = \{A \in Alg(\langle S, \Omega \rangle) \mid A|_o = B|_o \text{ and } B \in Mod[SP]\}$$

In the following sections we shall study some abstracted institutions and how proof techniques for T and D can be used in $\alpha + ASL \models \alpha + ASL$, but first we shall devote some time to a general definition of abstraction.

Definition 5.2.1 *Given two institutions,*

$$\mathbf{I} = \langle SIG, Mod, Sen, \models \rangle \qquad \alpha\mathbf{I} = \langle \alpha SIG, \alpha Mod, \alpha Sen, \models^{\alpha} \rangle$$

and an institution morphism $\langle \alpha_{SIG}, \alpha_{Mod}, \alpha_{Sen} \rangle$,

$$\alpha_{SIG} : SIG \rightarrow \alpha SIG$$
$$\alpha_{Mod} : Mod^{op} \Rightarrow \alpha_{SIG}^{op}; \alpha Mod^{op}$$
$$\alpha_{Sen} : \alpha_{SIG}; \alpha Sen \Rightarrow Sen$$

then $\alpha\mathbf{I}$ *is an* **abstracted institution** *w.r.t. the* **original institution I** *if* α_{SIG} *is a full and faithful functor (i.e. there exists a full subcategory of* αSIG *isomorphic to SIG in Cat) and there exists a functor* $C : \alpha SIG \rightarrow SIG$ *and natural transformations* $\mathbf{i} : \alpha_{SIG}; C \Rightarrow Id_{SIG}$ *and* $\mathbf{o} : Id_{\alpha SIG} \Rightarrow C; \alpha_{SIG}$ *forming an adjunction* $\langle C, \alpha_{SIG}, \mathbf{o}, \mathbf{i} \rangle$.

Morphisms such as $\langle \alpha_{SIG}, \alpha_{Mod}, \alpha_{Sen} \rangle$ *are then called* **abstracting institution morphisms.**

Functor α_{SIG} maps the original signatures and signature morphisms in SIG to their corresponding signatures and signature morphisms in the full subcategory of αSIG isomorphic to SIG; functor C yields the original signature (signature morphism) corresponding to each abstract signature (signature morphism)[3]; natural transformation i gives for each original signature Σ a morphism $i_{\Sigma} : C(\alpha_{SIG}(\Sigma)) \rightarrow$

[3]The name C is chosen to mean *concrete*, as opposite to *abstract*.

Σ; and natural transformation $\mathbf{o}$ gives for each signature Π in the abstracted institution a morphism $o_\Pi : \Pi \to \alpha_{SIG}(C(\Pi))$, that is, if α_{SIG} is an inclusion, o_Π is a morphism from Π to its corresponding original signature.

A few facts follow directly from the definition (see [Mac 71]).

Fact 5.2.2 *Given an arbitrary abstracting institution morphism*

$$\langle \alpha_{SIG}, \alpha_{Mod}, \alpha_{Sen} \rangle : \mathbf{I} \to \alpha\mathbf{I}$$

with adjunction $\langle C, \alpha_{SIG}, \mathbf{o}, \mathbf{i} \rangle$, *then*

1. *for every signature* Σ *in SIG,* $\Sigma \cong C(\alpha_{SIG}(\Sigma))$, (see [Mac 71] page 88)
2. C *preserves colimits,*
3. α_{SIG} *preserves limits,*
4. *for all* $\Sigma \in |SIG|$, *all morphisms* $h : \Pi \to \alpha_{SIG}(\Sigma)$ *in* αSIG *can be uniquely decomposed into* $h = o_\Pi; \alpha_{SIG}(\sigma)$ *where* $\sigma : C(\Pi) \to \Sigma$ *is a morphism in SIG and* $o_\Pi : \Pi \to \alpha_{SIG}(C(\Pi))$ *is the unit for* Π.

In practice, SIG can be chosen to be a full subcategory of αSIG so that $\langle C, \alpha_{SIG}, \mathbf{o} \rangle$ is a reflection. Then, Σ can be thought of as identical to $C(\alpha_{SIG}(\Sigma))$ and $\alpha_{SIG}(\Sigma)$; morphisms $i_\Sigma : C(\alpha_{SIG}(\Sigma)) \to \Sigma$ as identity morphisms id_Σ in SIG, and morphisms $o_\Pi : \Pi \to \alpha_{SIG}(C(\Pi))$ as morphisms in αSIG mapping an arbitrary Π to its corresponding original signature $C(\Pi)$, *e.g.* for $C(\Pi) = \Sigma$, $o_\Pi : \Pi \to \Sigma$ in αSIG.

Notational conventions The original institution is denoted by $\mathbf{I}$ and the abstracted institution by $\alpha\mathbf{I}$ as in the definition. The unit of C is usually denoted by $\mathbf{o}$ and unit morphisms o_Π are called **abstraction morphisms**. Functor α_{SIG} and

the morphisms i_Σ, corresponding to the counit $\mathbf{i}$, will be systematically dropped since they can be cancelled to the left and to the right, in other words we shall proceed as if (as is frequent in practice) $\Sigma = \alpha_{SIG}(\Sigma) = C(\alpha_{SIG}(\Sigma))$ and $i_\Sigma = id_\Sigma$. The symbol Σ will be reserved to denote objects of SIG (original signatures) and Π for arbitrary or proper objects of αSIG (abstracted signatures).

This definition of abstraction allows to view some known institutions as abstracted institutions, *e.g.* the institution of first order logic in section 5.4.2 below. However, we normally think of the abstracted institution as an enrichment of a given original one. The following section formalizes this construction.

5.2.1 Quotient abstracted institutions

Following [ST 87], an abstraction operation on a specification SP over Σ yields another specification $\alpha_{\equiv}SP$ over the same signature whose models are the closure of the original class of models w.r.t. a given signature-sorted equivalence relation $\equiv$ among the models[4]. For each Σ-specification SP

$$Mod[\alpha_{\equiv}SP] = \{A \in |Mod(\Sigma)| \mid A \equiv_\Sigma B,\ B \in Mod[SP]\}$$

Keeping this general idea in mind we define the following enrichment to a given institution.

Definition 5.2.3 *Given an institution* $\mathbf{I} = \langle SIG, Mod, Sen, \models \rangle$, *and the following:*

---

[4]In [ST 87] $\equiv$ is not sorted by the signatures, but this difference is not important.

1. *A category αSIG with a full inclusion*[5] $\alpha_{SIG} : SIG \to \alpha SIG$.

2. *A reflection $\langle C, \alpha_{SIG}, \mathbf{o} \rangle$ with $C : \alpha SIG \to SIG$ and $\mathbf{o} : Id_{\alpha SIG} \Rightarrow C; \alpha_{SIG}$.*

3. *An $|\alpha SIG|$-sorted equivalence relation $\equiv$ among the models of $\mathbf{I}$ such that $\equiv_\Pi \subseteq |Mod(C(\Pi))| \times |Mod(C(\Pi))|$ for all $\Pi \in |\alpha SIG|$.*

$\forall A, B \in |Mod(C(\pi_2))| . \; A \equiv_{\pi_2} B \Rightarrow A|_{C(h)} \equiv_{\pi_1} B|_{C(h)}$ for all $h : \pi_1 \to \pi_2$ in αS

4. *An $|\alpha SIG|$-sorted set Obs of sentences in $\mathbf{I}$ such that $Obs_\Pi \subseteq Sen(C(\Pi))$ for all $\Pi \in |\alpha SIG|$.*

$\forall \varphi \in Sen(C(\pi_1)) . \; \varphi \in Obs_{\pi_1} \Rightarrow C(h)(\varphi) \in Obs_{\pi_2}$ for all $h : \pi_1 \to \pi_2$ in αS

Functors $\alpha Mod : \alpha SIG \to Cat^{op}$ and $\alpha Sen : \alpha SIG \to Set$, and a relation $\models^\alpha_\Pi \in |\alpha Mod(\Pi)| \times \alpha Sen(\Pi)$ for each $\Pi \in |\alpha SIG|$ are defined as follows[6]*:*

- *$\alpha Mod(\Sigma) = Mod(\Sigma)$ if Σ is one of the original signatures. Otherwise, $\alpha Mod(\Pi) = Mod(C(\Pi))/\equiv_\Pi$.* †

  $\alpha Mod(\sigma) = Mod(\sigma)$ if σ is one of the original signature morphisms. For every unit morphism $o_\Pi : \Pi \to C(\Pi)$ in αSIG, $\alpha Mod(o_\Pi)$ is a mapping from every model M in $Mod(\Sigma)$ to its equivalence class $[M]$ in $Mod(\Sigma)/\equiv_\Pi$, also written $M|_{o_\Pi}$.

- *$\alpha Sen(\Sigma) = Sen(\Sigma)$ if Σ is one of the original signatures. Otherwise, $\alpha Sen(\Pi) = Obs_\Pi$.*

  $\alpha Sen(\sigma) = Sen(\sigma)$ if σ is one of the original signature morphisms. For every unit morphism $o_\Pi : \Pi \to C(\Pi)$ in αSIG, $\alpha Sen(o_\Pi)$ is an inclusion from Obs_Π into $Sen(\Sigma)$, usually called o_Π as well.

---

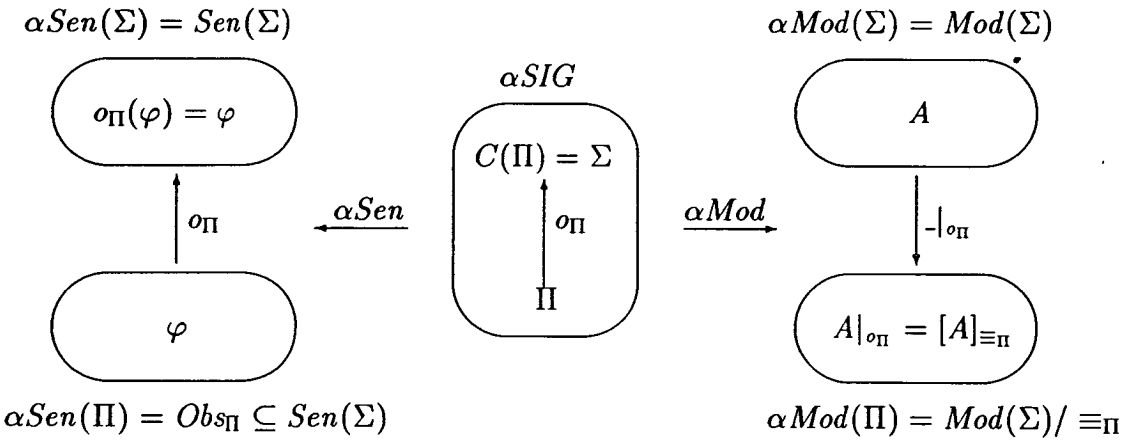
[5]Functor α_{SIG} is defined to be a full inclusion instead of an arbitrary faithful and full functor in order not to overload the notation.

[6]Proof that these functors are well-defined is included in the following proposition.

† the category of models over a signature is considered discrete.

- $\models^{\alpha}_{\Sigma} = \models_{\Sigma}$ *if* Σ *is one of the original signatures. Otherwise,* $M \models^{\alpha}_{\Pi} \varphi$ *iff for all* $A \in M$, $A \models_{C(\Pi)} \varphi$ *in* **I**.

If $\langle \alpha SIG, \alpha Mod, \alpha Sen, \models^{\alpha} \rangle$ *satisfies the satisfaction condition then it is called a* **quotient abstracted institution of I,** *denoted by* $\alpha^{\equiv}_{Obs}\mathbf{I}$.



In the above definition αMod and αSen are only defined on the objects of αSIG, the original morphisms and the unit morphisms. The following proposition shows that to be enough.

Proposition 5.2.4 *Given a situation as in the previous definition,* αMod *and* αSen *are well-defined functors over all morphisms in* αSIG.

Proof In their definition αMod and αSen are explicitly defined over all objects, original morphisms and unit morphisms in αSIG. Now, we must prove that they are implicitly defined for an arbitrary morphism $h : \Pi 1 \to \Pi 2$ in αSIG.

Since $\langle C, \alpha_{SIG}, \mathbf{o} \rangle$ is a reflection, the unit morphisms $o_{\Pi 1}$ and $o_{\Pi 2}$ make the following diagram commute in αSIG.

ions on Obs and ≡ allow αMod and αSen to be defined over $h : \Pi 1 \to \Pi 2$ a

$\alpha Mod(h)([A]_{\equiv \Pi 2}) = [A|_{C(h)}]_{\equiv \Pi 1}$, for all $A \in |Mod(C(\Pi 2))|$

$\alpha Sen(h)(\varphi) = Sen(C(h))(\varphi)$, for all $\varphi \in Obs\,\Pi 1$

functors. In the following we prove their uniqueness.

$$\begin{array}{ccc} \Pi 1 & \xrightarrow{o_{\Pi 1}} & C(\Pi 1) \\ \Big\downarrow h & & \Big\downarrow C(h) \\ \Pi 2 & \xrightarrow{o_{\Pi 2}} & C(\Pi 2) \end{array}$$

Consider two possible functors αMod_1 and αMod_2 which satisfy the requirements given for αMod.

For all unit morphisms o_Π we know that αMod_1 and αMod_2 agree and since they are functors

$$\begin{array}{ccc} \alpha Mod(\Pi 1) & \xleftarrow{-|_{o_{\Pi 1}}} & Mod(C(\Pi 1)) \\ \alpha Mod_1(h) \Big\uparrow \Big\uparrow \alpha Mod_2(h) & & \Big\uparrow \alpha Mod(C(h)) \\ \alpha Mod(\Pi 2) & \xleftarrow{-|_{o_{\Pi 2}}} & Mod(C(\Pi 2)) \end{array}$$

we have that

$$-|_{o_{\Pi 2}};\ \alpha Mod_1(h) = -|_{o_{\Pi 2}};\ \alpha Mod_2(h)$$

And finally, since $-|_{o_{\Pi 2}}$ is a surjective mapping onto $\alpha Mod(\Pi 2)$ and therefore epi and left-cancellable, the equation is reduced to

$$\alpha Mod_1(h) = \alpha Mod_2(h)$$

hence αMod is well-defined.

The same reasoning applies to αSen with the difference that the inclusions $\alpha Sen(o_\Pi) : Sen(C(\Pi)) \hookleftarrow \alpha Sen(\Pi)$ go in the opposite direction and they are not epi but mono. □

Since αMod and αSen are guaranteed to be well-defined we only need a condition between $\equiv$ and Obs characterizing when $\langle \alpha Sig, \alpha Mod, \alpha Sen, \models^\alpha \rangle$ is indeed a quotient abstracted institution.

Theorem 5.2.5 *Given a situation as in the previous definition*

$$\langle \alpha Sig, \alpha Mod, \alpha Sen, \models^{\alpha} \rangle$$

satisfies the satisfaction condition iff for all signatures Π *in* αSIG, *models* $A, B \in |Mod(C(\Pi))|$ *and sentences* $\varphi \in \alpha Sen(\Pi)$,

$$A \equiv_{\Pi} B \quad \Rightarrow \quad (A \models_{C(\Pi)} \varphi \Leftrightarrow B \models_{C(\Pi)} \varphi)$$

Proof Assuming the satisfaction condition to hold and considering an abstraction morphism $o_{\Pi} : \Pi \rightarrow \Sigma$ in αSIG, *i.e.* $C(\Pi) = \Sigma$, we have that for all $\varphi \in \alpha Sen(\Pi)$ (implicitly $o_{\Pi}(\varphi) = \varphi \in Sen(\Sigma)$)

$$\forall A \in Mod(\Sigma).\ (A|_{o_{\Pi}} \models^{\alpha}_{\Pi} \varphi \Leftrightarrow A \models_{\Sigma} \varphi)$$

Since $\_|_{o_{\Pi}}$ maps A to its equivalence class $[A]_{\equiv_{\Pi}} \in Mod(\Sigma)/\equiv_{\Pi}$, by definition of $\models^{\alpha}_{\Pi}$ this can be rewritten as

$$\forall A \in Mod(\Sigma).\ ((\forall B \in [A]_{\equiv_{\Pi}}.\ B \models_{\Sigma} \varphi) \Leftrightarrow A \models_{\Sigma} \varphi)$$

Taking the implication from right to left we can produce the following sequence of equivalences

$$\begin{aligned}
&\forall A \in Mod(\Sigma).\ ((\forall B \in [A]_{\equiv_{\Pi}}.\ B \models_{\Sigma} \varphi) \Leftarrow A \models_{\Sigma} \varphi) \\
&\iff \forall A \in Mod(\Sigma).\ (\forall B \in [A]_{\equiv_{\Pi}}.\ B \models_{\Sigma} \varphi) \vee (A \not\models_{\Sigma} \varphi) \\
&\iff \forall A \in Mod(\Sigma).\ (\forall B \in Mod(\Sigma).\ (A \equiv_{\Pi} B \Rightarrow B \models_{\Sigma} \varphi)) \vee (A \not\models_{\Sigma} \varphi) \\
&\iff \forall A \in Mod(\Sigma).\ (\forall B \in Mod(\Sigma).\ (A \not\equiv_{\Pi} B) \vee (B \models_{\Sigma} \varphi)) \vee (A \not\models_{\Sigma} \varphi) \\
&\iff \forall A, B \in Mod(\Sigma).\ (A \not\equiv_{\Pi} B) \vee (B \models_{\Sigma} \varphi) \vee (A \not\models_{\Sigma} \varphi) \\
&\iff \forall A, B \in Mod(\Sigma).\ (A \equiv_{\Pi} B \Rightarrow (A \models_{\Sigma} \varphi \Rightarrow B \models_{\Sigma} \varphi))
\end{aligned}$$

Considering that $A \equiv_{\Pi} B$ is equivalent to $B \equiv_{\Pi} A$, we can apply this fact twice on the same pair of models to obtain

$$\forall A, B \in Mod(\Sigma).\ (A \equiv_{\Pi} B \quad \Rightarrow \quad (A \models_{\Sigma} \varphi \Leftrightarrow B \models_{\Sigma} \varphi))$$

for all $\varphi \in \alpha Sen(\Pi)$.

In the other direction, $\_|_\sigma$ satisfies the satisfaction condition whenever σ is an original morphism because the satisfaction condition is already satisfied in **I**.

For each abstraction morphism $o_\Pi : \Pi \rightarrow \Sigma$, *i.e.* $C(\Pi) = \Sigma$, and models $A, B \in Mod(\Sigma)$, we assume that for all sentences $\varphi \in \alpha Sen(\Pi)$

$$A \equiv_\Pi B \quad \Rightarrow \quad (A \models_\Sigma \varphi \Leftrightarrow B \models_\Sigma \varphi)$$

Taking the implication from left to right and writing explicitly the quantifiers over models we have that for all sentences $\varphi \in \alpha Sen(\Pi)$

$$\forall A, B \in Mod(\Sigma).\ (A \equiv_\Pi B \quad \Rightarrow \quad (A \models_\Sigma \varphi \Rightarrow B \models_\Sigma \varphi)))$$

The same chain of equivalences as before can be applied to obtain

$$\forall A \in Mod(\Sigma).\ ((\forall B \in [A]_{\equiv_\Pi}.\ B \models_\Sigma \varphi) \Leftarrow A \models_\Sigma \varphi)$$

Implication in the opposite direction holds trivially since $A \in [A]_{\equiv_\Pi}$, hence

$$\forall A \in Mod(\Sigma).\ ((\forall B \in [A]_{\equiv_\Pi}.\ B \models_\Sigma \varphi) \Leftrightarrow A \models_\Sigma \varphi)$$

Using the definition of $\models^\alpha_\Pi$ and $\_|_{o_\Pi}$ we get

$$\forall A \in Mod(\Sigma).\ (A|_{o_\Pi} \models^\alpha_\Pi \varphi \Leftrightarrow A \models_\Sigma \varphi)$$

for all $\varphi \in \alpha Sen(\Pi)$, as desired.

Finally we have to prove that the satisfaction condition holds for an arbitrary morphism $h : \Pi 1 \rightarrow \Pi 2$ in αSIG.

Since $\langle C, \alpha_{SIG}, \mathbf{o} \rangle$ is a reflection, the unit morphisms $o_{\Pi 1}$ and $o_{\Pi 2}$ make the following diagram commute in αSIG.

$$\begin{array}{ccc} \Pi 1 & \xrightarrow{o_{\Pi 1}} & \Sigma 1 = C(\Pi 1) \\ \Big\downarrow h & & \Big\downarrow \sigma = C(h) \\ \Pi 2 & \xrightarrow{o_{\Pi 2}} & \Sigma 2 = C(\Pi 2) \end{array}$$

Since αSen is a functor the corresponding diagram in the category of sets also commutes, *i.e.*

$$\alpha Sen(h); \alpha Sen(o_{\Pi 2}) = \alpha Sen(o_{\Pi 1}); Sen(\sigma)$$

Taking an arbitrary sentence $\varphi \in Sen(\Pi 1)$, we get

$$\alpha Sen(o_{\Pi 2})(\alpha Sen(h)(\varphi)) = Sen(\sigma)(\alpha Sen(o_{\Pi 1})(\varphi))$$

and since $\alpha Sen(o_{\Pi 1})$ and $\alpha Sen(o_{\Pi 2})$ are inclusions, this is reduced to $\alpha Sen(h)(\varphi) = Sen(\sigma)(\varphi)$; that is to say, in the standard shorter notation:

$$h(\varphi) = \sigma(\varphi)$$

Now, for every model $M \in |Mod(\Sigma 2)|$ since the satisfaction condition holds in the original institution, we have that

$$M|_\sigma \models_{\Sigma 1} \varphi \Leftrightarrow M \models_{\Sigma 2} h(\varphi)$$

Since the satisfaction condition holds w.r.t. unit morphisms (proved above), we know also that

$$(M|_\sigma)|_{o_{\Pi 1}} \models^\alpha_{\Pi 1} \varphi \Leftrightarrow M|_\sigma \models_{\Sigma 1} \varphi$$

and similarly

$$M|_{o_{\Pi 2}} \models^\alpha_{\Pi 2} h(\varphi) \Leftrightarrow M \models_{\Sigma 2} h(\varphi)$$

At the same time, since αMod is a functor the following diagram commutes guaranteeing that $(M|_{o_{\Pi 2}})|_h = (M|_\sigma)|_{o_{\Pi 1}}$.

$$\begin{array}{ccc} \alpha Mod(\Pi 1) & \xleftarrow{-|_{o_{\Pi 1}}} & Mod(\Sigma 1) \\ \uparrow{\scriptstyle -|_h} & & \uparrow{\scriptstyle -|_\sigma} \\ \alpha Mod(\Pi 2) & \xleftarrow{-|_{o_{\Pi 2}}} & Mod(\Sigma 2) \end{array}$$

Combining the last three equivalences with this equation we conclude that:

$$(M|_{o\Pi 2})|_h \models^{\alpha}_{\Pi 1} \varphi \Leftrightarrow M|_{o\Pi 2} \models^{\alpha}_{\Pi 2} h(\varphi)$$

Now, it suffices to note that $\alpha Mod(o_{\Pi 2})$ (written $\_|_{o\Pi 2}$) is a full functor. Therefore, every model in $|\alpha Mod(\Pi 2)|$ is a $o_{\Pi 2}$-reduct of at least one model in $|Mod(\Sigma 2)|$. Hence, any equivalence holding for an arbitrary reduct $M|_{o\Pi 2}$ also holds for an arbitrary $A \in |\alpha Mod(\Pi 2)|$, *i.e.*

$$A|_h \models^{\alpha}_{\Pi 1} \varphi \Leftrightarrow A \models^{\alpha}_{\Pi 2} h(\varphi)$$

□

Finally, we can easily show quotient abstract institutions to be abstracted institutions as defined in page 135.

Proposition 5.2.6 *Given a quotient abstract institution $\alpha^{\equiv}_{Obs}\mathbf{I}$, there exists a abstracting institution morphism from* $\mathbf{I}$ *to* $\alpha^{\equiv}_{Obs}\mathbf{I}$.

Proof Given institutions $\mathbf{I} = \langle SIG, Mod, Sen, \models \rangle$ and

$$\alpha^{\equiv}_{Obs}\mathbf{I} = \langle \alpha SIG, \alpha Mod, \alpha Sen, \models^{\alpha} \rangle$$

as those in definition 5.2.3, we consider the tuple $\langle \alpha_{SIG}, \alpha_{Mod}, \alpha_{Sen} \rangle$ defined as:

1. $\alpha_{SIG} : SIG \rightarrow \alpha SIG$ is the full inclusion from the signatures in $\mathbf{I}$ to the signatures in $\alpha^{\equiv}_{Obs}\mathbf{I}$.

2. $\alpha_{Mod} : Mod^{op} \Rightarrow \alpha^{op}_{SIG}$; αMod^{op} is defined for all signatures $\Sigma \in |SIG|$ as the identity functor

$$\alpha_{Mod}\Sigma = Id_{Mod(\Sigma)}$$

3. $\alpha_{Sen} : \alpha_{SIG}; \alpha Sen \Rightarrow Sen$ is defined for all signatures $\Sigma \in |SIG|$ as the identity function

$$\alpha_{Sen}\Sigma = Id_{Sen(\Sigma)}$$

Using these definitions it is not difficult to verify that all the conditions required by institution morphisms hold and, in particular, the satisfaction condition

$$M \models_\Sigma \alpha_{Sen}\Sigma(\varphi) \quad \text{iff} \quad \alpha_{Mod}\Sigma(M) \models^{\alpha}_{\alpha_{SIG}(\Sigma)} \varphi$$

for all $\Sigma \in |SIG|$, $M \in \alpha Mod(\Sigma)$ and $\varphi \in \alpha Sen(\alpha_{SIG}(\Sigma))$, becomes

$$M \models_\Sigma \varphi \quad \text{iff} \quad M \models^{\alpha}_{\Sigma} \varphi$$

for all $\Sigma \in |SIG|$, $M \in \alpha Mod(\Sigma)$ and $\varphi \in \alpha Sen(\Sigma)$. Furthermore, considering the definitions of αSen and αMod over original signatures in definition 5.2.3, we obtain a trivial statement:

$$M \models_\Sigma \varphi \quad \text{iff} \quad M \models_\Sigma \varphi$$

for all $\Sigma \in |SIG|$, $M \in Mod(\Sigma)$ and $\varphi \in Sen(\Sigma)$.

Finally, since reflections are adjunctions and inclusion functors are always faithful, we conclude that

$$\langle \alpha_{SIG}, \alpha_{Mod}, \alpha_{Sen} \rangle : \mathbf{I} \to \alpha^{\equiv}_{Obs}\mathbf{I}$$

is an abstracting institution morphism.

□

From now on, we shall drop the subscripts for abstraction morphisms so that an abstraction morphism $o : \Pi \to \Sigma$ implicitly refers to a unit morphism $o_\Pi : \Pi \to \alpha_{SIG}(C(\Pi))$ whose target signature is an original signature Σ.

In a quotient abstracted institution $\alpha^{\equiv}_{Obs}\mathbf{I}$ and for an abstraction morphism $o : \Pi \to \Sigma$, the closure operator $T_o D_o$ means abstraction in the sense of $\alpha_\equiv$ in [ST 87]; i.e. for any Σ-specification SP

$$Mod[T_o D_o SP] = \{A \in |Mod(\Sigma)| \mid A \equiv_\Pi B,\ B \in Mod[SP]\}$$

In general, the closure operator $T_\sigma D_\sigma$ for a signature morphism $\sigma : \Sigma 1 \rightarrow \Sigma 2$ is equivalent to an abstraction with respect to the kernel induced by the reduct functor $\_|_\sigma$, i.e. $Ker(\_|_\sigma) \subseteq |Mod(\Sigma 2)| \times |Mod(\Sigma 2)|$ and $T_\sigma D_\sigma SP = \alpha_{Ker(\_|_\sigma)} SP$ for any $\Sigma 2$-specification SP. The construction of $\alpha^{\equiv}_{\overline{Obs}}\mathbf{I}$ is just a way to force the kernel for abstraction morphisms to be a given equivalence relation $\equiv$.

We can conclude that abstraction in ASL is defined as follows:

Definition 5.2.7 *Abstraction in* $ASL(\alpha^{\equiv}_{\overline{Obs}}\mathbf{I})$ *is a derived SBO* $\alpha_{\equiv\Pi}$ *defined as*

$$\alpha_{\equiv\Pi} = \lambda X : \Sigma.\ T_o D_o X$$

for each abstraction morphism $o : \Pi \rightarrow \Sigma$.

5.2.2 Observational abstractions

Following [ST 87], an abstraction operation $\alpha_\equiv$ is called observational abstraction w.r.t. a signature-sorted set of sentences Obs if $\equiv$ is a signature-sorted equivalence $\equiv^{Obs}$ such that for each signature Σ and for every $A, B \in Mod(\Sigma)$,

$$A \equiv^{Obs}_\Sigma B \iff (\forall \varphi \in Obs_\Sigma.\ A \models \varphi \Leftrightarrow B \models \varphi)$$

This concept can be easily defined for quotient abstracted institutions as follows:

Definition 5.2.8 *A quotient abstracted institution* $\alpha^{\equiv}_{\overline{Obs}}\mathbf{I}$ *with reflection* $\langle C, \alpha_{SIG}, \mathbf{o}\rangle$ *is called* **observational**, *and denoted by* $\alpha_{Obs}\mathbf{I}$, *if for each signature* Π *and models* $A, B \in Mod(C(\Pi))$, *the equivalence relation* $\equiv_\Pi$ *satisfies*

$$A \equiv_\Pi B \ \text{ iff } \ \forall \varphi \in \alpha Sen(\Pi).\ (A \models_{C(\Pi)} \varphi \Leftrightarrow B \models_{C(\Pi)} \varphi)$$

Note that observations are ordinary sentences which cannot discriminate between two equivalent models (this requirement comes from theorem 5.2.5). Observational abstractions occur in the particular case that the equivalence relation

$\equiv_\Pi$ is exactly the one induced by the observations Obs_Π which, by definition of a quotient abstracted institution, is $\alpha Sen(\Pi)$.

In this case $A \equiv_\Pi B$ can be tested by checking if A and B agree on the observations Obs_Π. For this reason, observational abstraction is related to the existence of proof techniques for entailment when the consequent contains abstraction, *i.e.* for $DATU \models \alpha + DATU$, as for example in proving entailment w.r.t. a behaviourally abstracted specification.

In a non-observational abstraction we cannot know if two Σ-models which satisfy the same observations are equivalent or not w.r.t. $\equiv_\Pi$ for an abstraction morphism $o : \Pi \rightarrow \Sigma$. This makes it impossible, at the specification level, to decide if $SP2 \models \alpha_{\equiv_\Pi} SP1$ is correct. For example, suppose $\equiv_\Pi$ is the identity relation and Obs_Π is empty; then all Σ-models satisfy the observable theorems of a specification $\alpha_{\equiv_\Pi} SP1$ but none of them is a model of $\alpha_{\equiv_\Pi} SP1$ unless it is also a model of $SP1$.

5.3 Proving satisfaction of all observable theorems

In this section we study in which cases $SP2 \models \alpha_{\equiv_\Pi} SP1$ can be proven by showing all observable theorems of the consequent $SP1$ to hold in the antecedent $SP2$. This relies, essentially, on the provability of behavioural equivalence between two models by *proving agreement on all observations*, but some side conditions render these proofs particularly interesting.

In terms of specification entailment, we intend to prove $SP2 \models \alpha_{\equiv_\Pi} SP1$ by proving that there is a sound theory $\Delta 2$ w.r.t. $SP2$ and an M-complete theory $\Delta 1$

w.r.t. $SP1$ such that

$$\Delta 2 \models_\Sigma \Delta 1 \cap Obs_\Pi$$

which is equivalent to requiring the inference rule

$$\frac{SP \models \varphi}{\alpha_{\equiv_\Pi} SP \models \varphi} \quad \varphi \in Obs_\Pi$$

to be M-complete w.r.t. $\alpha_{\equiv_\Pi}$.

Substituting $\alpha_{\equiv_\Pi}$ by its definition $T_o D_o$, we realize that the inference rule just introduced for abstraction is the composition of the inference rules given for T_σ and D_σ in chapter 3, taking into consideration that $\varphi \in Obs_\Pi$ if $o(\varphi) = \varphi$.

$$\frac{SP \vdash \sigma(\varphi)}{D_\sigma SP \vdash \varphi} \qquad \frac{SP \vdash \varphi}{T_\sigma SP \vdash \sigma(\varphi)}$$

Unfortunately, the rule for D_σ is not M-complete and, therefore, the rule for $\alpha_{\equiv_\Pi}$ is not M-complete unless some side conditions are met.

Definition 5.3.1 *A quotient abstracted institution $\alpha^{\equiv}_{\overline{Obs}}\mathbf{I}$ is called* **well-behaved** *if for every abstraction morphism $o : \Pi \to \Sigma$, theory Δ over Σ and partition $\langle O, \overline{O} \rangle$ of the set of Π-observations (i.e. $O \cap \overline{O} = \emptyset$ and $O \cup \overline{O} = Obs_\Pi$), it is the case that whenever there exists a model $A \in Mod(\Sigma)$ such that*

$$A \models_\Sigma (\Delta \cap Obs_\Pi) \cup O \quad \text{and} \quad A \not\models_\Sigma \overline{O}$$

then, there exists also $B \in Mod(\Sigma)$ such that

$$B \models_\Sigma \Delta \cup O \quad \text{and} \quad B \not\models_\Sigma \overline{O}$$

Intuitively, we can imagine Δ to be an abstracted specification and A a model satisfying the observable theorems of that specification, *i.e.* $A \models \Delta \cap Obs_\Pi$. Then, in a well-behaved institution, there must exist a model B of Δ which is observationally equivalent to A, hence A is a model of the abstracted Δ.

In particular, a well-behaved abstracted institution requires observations to be rich enough so that if a theory Δ is inconsistent with a certain partition $\langle O, \overline{O} \rangle$ of the observations, then this partition should also be inconsistent with the observable part of the theory $\Delta \cap Obs_\Pi$.

Lemma 5.3.2 *Let $\alpha^{\equiv}_{Obs}\mathbf{I}$ be a well-behaved institution and $o : \Pi \to \Sigma$ an abstraction morphism. If Δ is an inconsistent theory over Σ then $\Delta \cap Obs_\Pi$ is also inconsistent.*

Proof If there exists A such that $A \models_\Sigma \Delta \cap Obs_\Pi$ then we can partition Obs_Π into $\langle Obs_A, \overline{Obs_A} \rangle$ with $Obs_A = \{\varphi \in Obs_\Pi \mid A \models_\Sigma \varphi\}$ and $\overline{Obs_A} = Obs_\Pi \setminus Obs_A$. Applying the definition of well-behaved institution, Δ must be consistent. □

Theorem 5.3.3 *Given a well-behaved observational institution $\alpha_{Obs}\mathbf{I}$, the inference rule*

$$\frac{SP \models \varphi}{D_o SP \models \varphi} \quad \varphi \in Obs_\Pi$$

for every abstraction morphism $o : \Pi \to \Sigma$ is M-complete.

Proof Let Δ be an M-complete theory for SP, *i.e.* $Mod[\Delta] \subseteq Mod[SP]$. The inferred theory is $\Delta_{\text{inf}} = \Delta \cap Obs_\Pi$.

If Δ is inconsistent so is Δ_{inf} by the previous lemma, and $Mod[\Delta_{\text{inf}}] \subseteq Mod[D_o SP]$ holds trivially.

Otherwise, it is enough to show that for all Π-models M such that $M \models^{\alpha}_{\Pi} \Delta_{\text{inf}}$ there exists a model $B \in Mod[SP]$ such that $B \in M$, since applying reducts $M = B|_o \in Mod[D_o SP]$ as desired.

In an observational institution, M is characterized by a partition $\langle Obs_M, \overline{Obs_M} \rangle$ of the set of observations Obs_Π depending on whether they are satisfied or they are not satisfied in M, *i.e.* $Obs_M \cup \overline{Obs_M} = Obs_\Pi$ and

$$M = \{A \in Mod(\Sigma) \mid A \models_\Sigma Obs_M, A \not\models_\Sigma \overline{Obs_M}\}$$

Moreover, by construction, M is an equivalence class of Σ-models w.r.t. $\equiv_{Obs}$ and therefore it cannot be empty. Hence, there exists a Σ-model $A \in M$ and, since $M \models^{\alpha}_{\Pi} \Delta_{inf}$,

$$A \models_\Sigma (\Delta \cap Obs_\Pi) \cup Obs_M \quad \text{and} \quad A \not\models_\Sigma \overline{Obs_M}$$

In a well-behaved institution, there must exist a Σ-model B such that $B \models_\Sigma \Delta \cup Obs_M$ and $B \not\models_\Sigma \overline{Obs_M}$, therefore $B \in M$. At the same time, $B \in Mod[\Delta]$ and by M-completeness of Δ, $B \in Mod[SP]$ as required.

□

Since the soundness of the rule without the side condition has already been proven (see section 3.3), the rule is exact.

Corollary 5.3.4 *In a well-behaved observational institution $\alpha_{Obs}\mathbf{I}$, any specification SP in $\alpha + ATU$ is equivalent to a presentation $\Phi_{SP} \subseteq Sen(Sig[SP])$ which can be obtained by adding an equivalence*

$$\alpha_{\equiv_\Pi} A_\Phi \epsilon_\Sigma = A_{Cl(\Phi) \cap Obs_\Pi} \epsilon_\Sigma$$

to those given for ATU in section 3.2.

Unfortunately, the theory $Cl(\Phi) \cap Obs_\Pi$ may not have a finite presentation even if $Cl(\Phi)$ has. This is the case of the common specification of stacks where a non-observable axiom

```
pop(push(x,S)) = S
```

gives rise to an infinite set of observable consequences

```
top(pop(push(x,S))) = top(S)
top(push(y,pop(push(x,S)))) = top(push(y,S))
...
```

which cannot be expressed by an equivalent finite presentation. This problem arises also in first order logic [Sch 91] and prevents abstraction in a well-behaved observational institution from being trivial. In practice, well-behaved observational abstractions allow entailment proofs by *proving satisfaction of all observable theorems*, but they do not tell us how to perform such proofs. In the end, we must rely on appropriate proof techniques such as context induction [Hen 88] or proof by consistency [Lin 87].

Comparing our proof results for abstraction in well-behaved observational institutions to those in [ST 87, ST 88a], a few comments can be made.

Defining abstraction to be the closure $T_o D_o$ allows us to give the inference rule for abstraction in terms of those for D_σ and T_σ, so that no extra rule is needed.

Well-behaviour of an abstracted institution is not such a new requirement. Fact 13 in [ST 87] shows that in order to be able to reduce abstracted first order presentations to equivalent first order presentations, observations are required to be closed under negation and conjunction. Well-behaviour can be derived from this closure.

Using an institution-independent notion of negation and conjunction, we can extend that result to an arbitrary quotient abstracted institution. Consider for example the definition of closure under negation and conjunction given in [Tar 86a] for an arbitrary institution, then the following result follows:

Proposition 5.3.5 *Given an institution* $\alpha^{\equiv}_{Obs}\mathbf{I}$ *with a reflection* $\langle C.\alpha_{SIG}, \mathbf{o}\rangle$, *if for every* $\Pi \in |\alpha SIG|$ *each* Obs_Π *is closed under negation and (possibly infinitary) conjunction, then* $\alpha^{\equiv}_{Obs}\mathbf{I}$ *is well-behaved.*

Proof Let $o : \Pi \rightarrow \Sigma$ be an abstraction morphism, Δ a theory over Σ, $\langle O, \overline{O} \rangle$ a partition of Obs_Π and A a model such that $A \models_\Sigma (\Delta \cap Obs_\Pi) \cup O$ and $A \not\models_\Sigma \overline{O}$.

If $\Delta \cup O$ is consistent, there exists a model $B \models_\Sigma \Delta \cup O$. Moreover, for any observation $\overline{\varphi} \in \overline{O}$ its negation φ must belong to O, since A must satisfy either φ or $\overline{\varphi}$. Then, B cannot satisfy $\overline{\varphi}$ because $B \models_\Sigma O$; hence $B \not\models_\Sigma \overline{O}$.

If $\Delta \cup O$ is inconsistent, we come to a contradiction in each possible case:

1. If Δ is inconsistent then the contradiction $\Box$ (negation of the conjunction of an empty set of sentences $\neg(\wedge\emptyset)$) is in Δ and therefore in $\Delta \cap Obs_\Pi$ since Obs_Π is closed under negation and conjunction. Hence A satisfying $\Delta \cap Obs_\Pi$ cannot exist.

2. If O is inconsistent then A satisfying O cannot exist.

3. Otherwise, there exists a set of sentences $\Phi \subseteq O$ in contradiction with Δ. Since Obs_Π is closed under infinite conjunctions there exists $\varphi_\Phi \in Obs_\Pi$ equivalent to Φ and, since it is closed under negation, $\neg\varphi_\Phi \in Obs_\Pi$ as well. Since $\Delta \cup \Phi$ is inconsistent, Δ must include $\neg\varphi_\Phi$, hence $\neg\varphi_\Phi \in \Delta \cap Obs_\Pi$. Finally $(\Delta \cap Obs_\Pi) \cup O$ is inconsistent since $\varphi_\Phi \in O$.

$\Box$

If the logic of the sentences is compact, closure under possibly infinitary conjuctions can be replaced by closure under finite conjunctions, like in the case of first order logic in fact 13 of [ST 87].

Another example in [ST 87] where observations are open formulae fails to be well-behaved. That proposal is intuitively sound since it includes as observable all values of an observable sort, even those which are not reachable – note that if observations are first order sentences we cannot refer individually to non-reachable

values. Unfortunately, such a modification leads to a logic of open formulae which is no longer compact — see the counterexample in [ST 87] where an infinite set of observations are in contradiction with a first order sentence while no finite subset is.

In this chapter we are not so much concerned with the problem of finding a realistic set of observations which is at the same time well-behaved, as was the case in [ST 87]. Our goal is to develop a certain proof theory for specifications with abstraction, and in this sense observational and well-behaved institutions are those allowing entailment proofs to be performed by *proving satisfaction of all observable theorems.*

In the following we review some common abstracted institutions for behavioural abstraction, and analyze them for observationality and well-behaviour.

5.3.1 Behavioural abstraction and EQ

Probably the most well-known example of abstraction is behavioural abstraction of an equational specification w.r.t. some non-observable sorts. In the following we present some abstracted institutions related to this idea.

Beh(EQ)

This abstracted institution matches the standard intuition of behavioural abstraction w.r.t. some non-observable sorts (as in [MG 85, NO 88, ST 87, Gog 90] reviewed above).

First a category of signatures of behaviour is defined (as explicitly given in [Gog 90] and implicitly standard long before):

Definition 5.3.6 *Given the category SIG of algebraic signatures,* **signatures of behaviour** *form a category Beh(SIG) as follows:*

- **Objects:** *Pairs* $\langle \Sigma, O \rangle$ *where* Σ *is an object of SIG and* O *a set of sorts in* Σ.

- **Morphisms:** *Arrows* $\sigma : \langle \Sigma 1, O1 \rangle \rightarrow \langle \Sigma 2, O2 \rangle$ *where* $\sigma : \Sigma 1 \rightarrow \Sigma 2$ *is a morphism in SIG such that* $\sigma(O1) \subseteq O2$. *Identities and composition are as in SIG.*

According to this definition there is a full subcategory of $Beh(SIG)$ which is isomorphic to SIG, namely the full subcategory containing signatures $\langle \Sigma, Sorts(\Sigma) \rangle$. Hence, there is a full and faithful functor $Beh_{SIG} : SIG \rightarrow Beh(SIG)$ which maps each signature Σ to its equivalent $\langle \Sigma, Sorts(\Sigma) \rangle$ and is the identity on signature morphisms.

Behaviours of algebras w.r.t. some observable sorts can be defined as in [MG 85], or equivalently, a concise definition can be given in terms of the functorial interpretation of an algebra.

Definition 5.3.7 *Given an algebraic signature* Σ *and an algebra*

$$A : St_\Sigma \rightarrow Set$$

then, the **behaviour of an algebra** A **w.r.t. a set of sorts** O *in* Σ *is a functor*

$$M : St_\Sigma(O) \rightarrow Set$$

where $St_\Sigma(O)$ *is the minimal full subcategory of* St_Σ *containing all the sorts in* O *and being closed under products. Then,* M *is the restriction of* A *to such a subcategory.*

A behaviour M is also an algebra since, like A, it is a product preserving functor from a category with finite products to Set. However, the signature of M is no longer Σ but a generally infinite signature with a function name for each Σ-term using variables over observable sorts and delivering a value of an observable sort, as explicitly defined in [MG 85].

Now we are able to define an abstracted institution w.r.t. **EQ** using signatures in $Beh(SIG)$, considering *behaviours* to be the models of the abstracted signatures, and equations between terms over observable sorts to be the sentences of the abstracted signatures.

Technically, we shall define the new abstracted institution as a quotient abstracted institution, repeating the construction given in definition 5.2.3. In order to perform such a construction original signatures are required to form a full subcategory of αSIG; since this is not the case for SIG we are going to consider SIG' to be the full subcategory of signatures in αSIG of the form $\langle \Sigma, Sorts(\Sigma)\rangle$ and $\mathbf{EQ}' = \langle SIG', Alg', Eq', \models'\rangle$ to be the obvious equational institution defined over SIG', *i.e.* $Alg(\Sigma) = Alg'(\langle \Sigma, Sorts(\Sigma)\rangle)$, $Eq(\Sigma) = Eq'(\langle \Sigma, Sorts(\Sigma)\rangle)$ and $\models_\Sigma = \models'_{\langle \Sigma, Sorts(\Sigma)\rangle}$.

Lemma 5.3.8 *Let Beh_{SIG} be the full inclusion from SIG' to $Beh(SIG)$, $C : Beh(SIG) \to SIG'$ a functor defined as follows*

$$C(\langle \Sigma, O\rangle) = \langle \Sigma, Sorts(\Sigma)\rangle \qquad C(\sigma) = \sigma$$

and $\mathbf{o} : Id_{Beh(SIG)} \Rightarrow C$; *$Beh_{SIG}$ a natural transformation such that for every $Beh(SIG)$-signature $\langle \Sigma, O\rangle$,*

$$o_{\langle \Sigma, O\rangle} : \langle \Sigma, O\rangle \to \langle \Sigma, Sorts(\Sigma)\rangle$$

Then, $\langle C, Beh_{SIG}, \mathbf{o}\rangle$ is a reflection.

Proof We shall prove that for each signature of behaviour $\langle \Sigma, O\rangle$ its unit morphism $o_{\langle \Sigma, O\rangle} : \langle \Sigma, O\rangle \to \langle \Sigma, Sorts(\Sigma)\rangle$ is a universal arrow.

Since morphisms in $Beh(SIG)$ are also morphisms in SIG' and each morphism $o_{\langle \Sigma, O\rangle}$ in $Beh(SIG)$ is just the identity $id_{\langle \Sigma, Sorts(\Sigma)\rangle}$ in SIG', it is trivial that for each morphism $h : \langle \Sigma, O\rangle \to \langle \Sigma', O'\rangle$ – which is also a morphism $h : \langle \Sigma, Sorts(\Sigma)\rangle \to \langle \Sigma', Sorts(\Sigma')\rangle$ in SIG' – there exists a unique h' such that

$$id_{\langle \Sigma, Sorts(\Sigma)\rangle}; h' = h$$

because h' must be h.

□

The abstracted institution **Beh(EQ)** is defined as follows:

Theorem 5.3.9 *Let* $\mathbf{EQ}' = \langle SIG', Alg', Eq', \models' \rangle$ *be the institution of equational logic with signatures of the form* $\langle \Sigma, sorts(\Sigma) \rangle$ *for all* $\Sigma \in |SIG|$ *and consider the following:*

1. *The category* $Beh(SIG)$ *and the full inclusion* $Beh_{SIG} : SIG' \rightarrow Beh(SIG)$.

2. *The reflection* $\langle C, Beh_{SIG}, \mathbf{o} \rangle$ *with* $C : Beh(SIG) \rightarrow SIG'$ *and* $\mathbf{o} : Id_{Beh(SIG)} \Rightarrow C$; Beh_{SIG} *as defined in lemma 5.3.8.*

3. *The* $|Beh(SIG)|$*-sorted equivalence relation* $\equiv$ *between algebras defined for all* $\langle \Sigma, O \rangle \in |Beh(SIG)|$ *and* Σ*-algebras (i.e.* $\langle \Sigma, Sorts(\Sigma) \rangle$*-algebras)* A *and* B *as:* $A \equiv_{\langle \Sigma, O \rangle} B$ *iff* A *and* B *have the same behaviour w.r.t.* O.

4. *The* $|Beh(SIG)|$*-sorted set of equations* Obs *defined for all* $\langle \Sigma, O \rangle$ *as:*

$$Obs_{\langle \Sigma, O \rangle} = \{ \forall X.\ t1 = t2 \in Eq(\Sigma) \mid sort(t1) \in O, sorts(X) \subseteq O \}$$

Then, $\mathbf{Beh(EQ)} = \langle Beh(SIG), \alpha Mod, \alpha Sen, \models^{\alpha} \rangle$ *defined as in definition 5.2.3 using the above components is a quotient abstracted institution of* $\mathbf{EQ}'$.

Proof Applying proposition 5.2.4 and theorem 5.2.5, **Beh(EQ)** is an institution unless two algebras with the same behaviour can satisfy different sets of observations.

However, for all $\langle \Sigma, Sorts(\Sigma) \rangle \in |SIG'|$ and Σ-equations (*i.e.* $\langle \Sigma, Sorts(\Sigma) \rangle$-equations) $\forall X.\ t1 = t2 \in Obs_{\langle \Sigma, O \rangle}$, both terms $t1$ and $t2$ correspond to morphisms in St_{Σ} which are also in $St_{\Sigma}(O)$ since they are arrows from a product of

observable sorts — sorts of variables in X — to an observable sort — the sort of $t1$ and $t2$.

All Σ-algebras (*i.e.* $\langle \Sigma, Sorts(\Sigma)\rangle$-algebras) A and B with the same behaviour M map morphisms $t1$ and $t2$ in St_Σ to the same functions $M(t1) = A(t1) = B(t1)$ and $M(t2) = A(t2) = B(t2)$ in *Set*. If functions $M(t1)$ and $M(t2)$ produce the same result for all valuations of the variables in X, equation $\forall X.\ t1 = t2$ is satisfied by A and B; otherwise, neither A or B satisfy the equation. Hence, the equation does not distinguish A from B.

□

By construction, the closure T_oD_o w.r.t. an abstraction morphism $o : \langle \Sigma, O\rangle \rightarrow \langle \Sigma, Sorts(\Sigma)\rangle$ is behavioural abstraction, for a notion of behaviour as in [MG 85]. If T_oD_o is applied to a specification SP with a class of models closed up to isomorphism, then T_oD_oSP is also closed up to isomorphism, as with behavioural abstraction in the sense of [SW 83, NO 88, ST 87].

Unfortunately, **Beh(EQ)** is not observational.

In first place, in order for **Beh(EQ)** to be observational the set of algebras behaviourally equivalent to any given one should be closed up to isomorphism but, in this case, it is not. This can be easily fixed by changing $A \equiv_{\langle \Sigma, O\rangle} B$ to mean that A and B have isomorphic behaviours w.r.t. O, so that T_oD_o directly means behavioural abstraction as is standard. We shall refer to this variant by **Beh'(EQ)**.

However, there are still cases where two algebras satisfying the same observable equations are not behaviourally equivalent (see examples in [ST 87]).

This lack of observationality is caused by non-reachable values in observable sorts. Since equations can only refer to them globally, two algebras may satisfy

the same equations while non-reachable values are more numerous in one algebra than in the other. This is analogous to cases where non-isomorphic algebras satisfy the same equations, *e.g.* all non-empty algebras over a signature without function symbols or constants satisfy the same trivial equations.

The lack of observationality, as discussed at the end of section 5.2.2, prevents **Beh(EQ)** from being useful at specification entailment proofs; in particular, theorem 5.3.3 does not apply.

BehR(EQ)

The lack of observationality of **Beh'(EQ)** can be solved by considering only reachable algebras.

Unlike ordinary algebras, reachable algebras are characterized up to isomorphism by a set of equations, *i.e.* if two reachable algebras satisfy the same ground equations then they are isomorphic.

Similarly, since behaviours are also algebras, reachable behaviours are characterized up to isomorphism by a set of observable equations.

Proposition 5.3.10 *If two Σ-algebras A and B are reachable on observable sorts O and they satisfy exactly the same ground observable equations in $Obs_{(\Sigma,O)}$, then the behaviours of A and B w.r.t. O are isomorphic.*

Proof Let M and N be the behaviours of Σ-algebras A and B w.r.t. O.

Behaviours M and N are algebras over a, generally, infinite signature Σ_O which, by construction, has sorts O and functions names all Σ-terms from observable to observable sorts.

Considering that A and B are reachable on sorts O, their behaviors M and N are reachable Σ_O-algebras. Moreover, since observable equations in $Obs_{(\Sigma,O)}$ are

pairs of Σ-terms from observable to observable sorts, we have that Σ_O-equations are equivalent to equations in $Obs_{\langle\Sigma,O\rangle}$.

If A and B satisfy exactly the same ground observable equations in $Obs_{\langle\Sigma,O\rangle}$, their behaviours M and N satisfy exactly the same ground Σ_O-equations, hence M and N are isomorphic.

□

Let the institution **BehR(EQ)** be as **Beh′(EQ)** but considering only those signature morphisms which are isomorphisms in SIG and those algebras which are reachable, *i.e.* with models defined for all $\langle S,\Omega\rangle \in |SIG|$ as

$$|Mod(\langle S,\Omega\rangle)| = \{A \in |Alg(\langle S,\Omega\rangle)| \mid A \text{ is reachable on } S\}$$

The restriction on the signature morphisms guarantees the σ-reduct of a reachable algebras to be also reachable. Then, taking observations to be all ground equations over observable sorts, the following results hold:

Theorem 5.3.11 BehR(EQ) *is an observational institution.*

Proof For all abstraction morphisms $o : \Pi \to \Sigma$, the set of observable equations Obs_Π includes all ground equations over observable sorts. Therefore, by proposition 5.3.10, any pair of reachable algebras $A, B \in |Alg(\Sigma)|$ such that

$$\forall \varphi \in Obs_\Pi.\ (A \models_\Sigma \varphi \Leftrightarrow B \models_\Sigma \varphi)$$

must have isomorphic behaviours, hence $A \equiv_\Pi B$.

Conversely, if $A \equiv_\Pi B$ then, by theorem 5.2.5, both models must agree on all observations. Hence, **BehR(EQ)** is observational.

□

In order to prove that **BehR(EQ)** is also well-behaved, we must recall that multi-sorted equational logic $\vdash^{\mathbf{EQ}}$ (*c.f.* [GM 81, EM 85]) is a complete inference system w.r.t. satisfaction in **EQ**, that initial algebras satisfy the "no confusion" principle, *i.e.* given a set of equations Φ if an initial algebra among those satisfying Φ satisfies a ground equation φ then $\Phi \models \varphi$, and also the following fact:

Fact 5.3.12 (Hen 88) *Given a equational presentation Φ over Σ and a set of observable sorts O in Σ, all observable ground consequences of Φ in $\vdash^{\mathbf{EQ}}_{\bullet}$ can be derived from the observable ground consequences of each equation in Φ.*

A proof of a more general fact can be found in theorem 6.22 of [Hen 88]. This fact is used in [Hen 88] to prove the soundness of context induction, here, the same fact serves to prove the well-behaviour of **BehR(EQ)**.

Theorem 5.3.13 BehR(EQ) *is a well-behaved institution.*

Proof Let $o : \Pi \to \Sigma$ be an abstraction morphism, Δ an equational theory over Σ, A a Σ-algebra satisfying $\Delta \cap Obs_\Pi$ (the observable theorems of Δ) and $\langle O, \overline{O} \rangle$ the partition of Obs_Π such that

$$A \models_\Sigma O \quad \text{and} \quad A \not\models_\Sigma \overline{O}$$

Then, we take B to be an initial Σ-algebra (unique up to isomorphism) satisfying equations $\Delta \cup O$ and prove that $B \not\models_\Sigma \overline{O}$. Assume $B \models \varphi$, by cases:

1. φ is not a ground equation. In this case, by definition of $\overline{O}$, $\varphi \notin \overline{O}$ trivially.

2. φ is a ground equation. Since initial algebras satisfy the "no confusion" principle ground equations cannot be satisfied by B unless they follow from $\Delta \cup O$, hence $\Delta \cup O \models_\Sigma \varphi$ which, by completeness of $\vdash^{\mathbf{EQ}}$, is equivalent to $\Delta \cup O \vdash^{\mathbf{EQ}} \varphi$.

In this situation we can use fact 5.3.12 to conclude that φ can be derived from the observable ground consequences of each equation in $\Delta \cup O$. By induction on the length of such a derivation we prove that φ is not in $\overline{O}$:

(a) φ is an observable ground consequence of an equation $\varphi' \in \Delta \cup O$. If $\varphi' \in \Delta$, since Δ is closed under entailment, φ must also be in Δ and therefore $\varphi \notin \overline{O}$. On the other hand, if $\varphi' \in O$ it means that A satisfies φ' and therefore A satisfies φ, hence $\varphi \notin \overline{O}$.

(b) φ follows from some observable ground consequences Φ of each equation in $\Delta \cup O$. By the induction hypothesis no equation in Φ belongs to $\overline{O}$, hence $\Phi \subseteq O$. Then, by definition of O, A satisfies Φ and therefore A satisfies φ, hence $\varphi \notin \overline{O}$.

We conclude that no equation satisfied by B is in $\overline{O}$ and therefore $B \not\models_\Sigma \overline{O}$ as expected.

□

Well-behaviour of **BehR(EQ)** follows from the existence of initial models among those satisfying a set of equations, restricting to ground observations and fact 5.3.12. The same properties and, therefore, well-behaviour also hold in **Beh(EQ)** if observations are required to be ground equations.

Other institutions

The reader might protest that using **BehR(EQ)** is too much restrictive, in particular because limiting signature morphisms to be isomorphisms prevents writing any sensible structured specification. In this situation we can search for still another institution.

First, as proposed in [ST 87] we can shift to a stronger logic so that observations can be open formulae. Doing this, we stick to the standard semantics of

behavioural abstraction as in [SW 83, MG 85, NO 88] and obtain an observational institution, however well-behaviour is lost. We shall call this abstracted institution **BehO(EQ)**.

Alternatively, we can preserve well-behaviour by not changing the notion of observation or model, and obtain observationality by changing the relation $\equiv_{\langle\Sigma,O\rangle}$ and thus the semantics of $T_o D_o$ when some observable sorts are not finitely generated. We shall call such an abstracted institution **BehQ(EQ)**.

Proposition 5.3.14 *Let* $\mathbf{EQ}' = \langle SIG', Alg', Eq', \models' \rangle$ *be the institution for equational logic with signatures of the form* $\langle \Sigma, sorts(\Sigma) \rangle$ *for all* $\Sigma \in |SIG|$ *and consider the following:*

1. *The category* $Beh(SIG)$ *and the full inclusion* $Beh_{SIG} : SIG' \to Beh(SIG)$.

2. *The reflection* $\langle C, Beh_{SIG}, \mathbf{o} \rangle$ *with* $C : Beh(SIG) \to SIG'$ *and* $\mathbf{o} : Id_{Beh(SIG)} \Rightarrow C$; Beh_{SIG} *as defined in lemma 5.3.8.*

3. *The* $|Beh(SIG)|$*-sorted equivalence relation* $\equiv$ *between algebras defined for all* $\langle \Sigma, O \rangle \in |Beh(SIG)|$ *and* Σ*-algebras (i.e.* $\langle \Sigma, Sorts(\Sigma) \rangle$*-algebras)* A *and* B *as:*

$$A \equiv_{\langle\Sigma,O\rangle} B \quad \text{iff} \quad \forall \varphi \in Obs_{\langle\Sigma,O\rangle}.\ (A \models_\Sigma \varphi \ \Leftrightarrow\ B \models_\Sigma \varphi)$$

4. *The* $|Beh(SIG)|$*-sorted set of equations* Obs *defined for all* $\langle \Sigma, O \rangle$ *as:*

$$Obs_{\langle\Sigma,O\rangle} = \{\forall X.\ t1 = t2 \in Eq(\Sigma) \mid sort(t1) \in O, sorts(X) \subseteq O\}$$

Then, $\mathbf{BehQ(EQ)} = \langle Beh(SIG), \alpha Mod, \alpha Sen, \models^\alpha \rangle$ *defined as in definition 5.2.3 using the above components is an observational institution* $\alpha_{Obs}\mathbf{EQ}'$.

Proof Applying proposition 5.2.4 and theorem 5.2.5, **BehQ(EQ)** is a quotient abstracted institution. Considering definition 5.2.8, **BehQ(EQ)** is observational trivially.

□

Another institution we may think of is **Beh(ALG)**; *i.e.* we repeat over **ALG** the construction of **Beh(EQ)** over **EQ**. In this case, the notion of abstraction in **Beh(ALG)** is like that in **Beh(EQ)** and the only difference concerns observations, which now can be defined to be closed under negation and conjunction in order to have a well-behaved abstracted institution. This approach, despite dealing with first order sentences over equality, is basically an algebraic approach.

Some comparisons

The above institutions for behaviour are closely related to those explicitly or implicitly defined in [SW 83, MG 85, ST 87, NO 88, Hen 88, Gog 90].

Among them the only one defining explicitly an institution for behaviour is [Gog 90], the so-called *hidden sort equational institution.* However, the models over signatures with non-observable sorts are just ordinary algebras, therefore for a morphism $o : \langle \Sigma, O \rangle \to \langle \Sigma, Sorts(\Sigma) \rangle$, it holds that $Mod[T_o D_o SP] = Mod[SP]$.

The closest of these approaches to **Beh(EQ)** is [GM 82, MG 85]. Behaviours as defined here are equivalent to theirs (from there comes the name) although their definition is not formulated in terms of functors but directly considering infinitary signatures with a new function symbol for each *term* from observable sorts to observable sorts expressible in the original signature.

In [NO 88] the models over a signature with non-observable sorts are as in [Gog 90] but there are more arrows between them. The new *homomorphisms*

relate behaviourally equivalent algebras in such a way that closure under isomorphism produces the same result as in **Beh'(EQ)**, *i.e.* closure up to behavioural equivalence.

Finally, $\alpha_{\equiv}$ in [ST 87] uses an explicit relation of equivalence between models as we do, although equivalence classes of models are not considered models at all, nor is an explicit institution for behaviour defined. The two approaches presented in [ST 87] are observational by definition; the one using ground observations corresponds to **BehQ(EQ)** and the other using open formulae corresponds to **BehO(EQ)**.

In [Hen 88] only finitely generated models are considered and, usually, only a part of the carrier of a sort is defined as observable, however when whole carriers are considered observable we obtain abstraction as in **BehR(EQ)**. Note that in [Hen 88] behavioural equivalence is defined directly as the satisfaction of the same observations, which happens to be consistent with the standard notion of behavioural equivalence due to the observationality and well-behaviour of this institution.

Provided that all specifications define isomorphism-closed classes of models, the semantics of $T_o D_o$ in **Beh'(EQ)** (also [NO 88]), **BehO(EQ)** (also [ST 87]) and **Beh(EQ)** (also [MG 85]) are the same. If we are only concerned with algebras with reachable observable sorts, then **BehR(EQ)** (also [Hen 88]) and **BehQ(EQ)** (also [ST 87]) are also the same.

We can conclude that, among the institutions considered here, in order to obtain an M-complete rule for abstraction and keep behavioural abstraction as meaning what is expected, we should use **BehR(EQ)**. Essentially, we should not hide the generators of observable sorts.

5.4 Treating abstraction as hiding

In the introduction of this chapter we informally classified proving techniques related to behavioural equivalence in essentially two categories: those using *explicit definition of congruences* and those *proving agreement on all observations.*

In our framework, the second technique relates to entailment proofs by *proving satisfaction of all observable theorems* studied in the last section. There, we paid attention to those cases in which abstraction is "better" than hiding, in the sense that it has an exact inference rule.

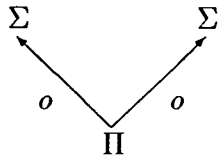
In this section we treat abstraction as ordinary hiding, that is, we apply the proofs techniques developed for hiding in chapter 4 to the particular case where the operation D_σ uses an abstraction morphism o, *i.e.* D_o. This approach does not require an exact rule for abstraction.

In the case of behavioural abstraction in first order logic, this *abstraction as hiding* approach turns out to be related to proofs by *explicit definition of congruences* as used in the literature.

5.4.1 Verification techniques for $T_O D_O$

Chapter 4 presented some proof techniques for the verification of entailments between specifications in *DATU.* Considering abstraction to be the application of a pair $T_o D_o$ using an abstraction morphism o, the same techniques are applicable to the verification of entailments between abstracted specifications. However, some problems arise depending of which abstracted institution we use. For instance, taking **Beh(EQ)** to be our working institution leads to rather poor results.

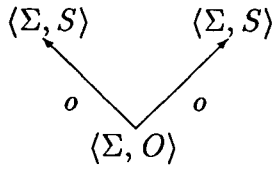
Consider task 1 as presented on page 92 and take $\sigma 1$ and $\sigma 2$ to be an abstraction morphism $o : \Pi \rightarrow \Sigma$:



$$D_o A_{\Phi 2} \epsilon_\Sigma \models D_o A_{\Phi 1} \epsilon_\Sigma$$

All results in chapter 4 related to the inheriting strategy rely on a pushout diagram of signatures which combine in one single signature the visible symbols plus the hidden symbols of the antecedent and the consequent. This basic construct delivers a trivial result in the case of $Beh(SIG)$ and an abstraction morphism o as in the figure.

Proposition 5.4.1 *Given the following diagram in* $Beh(SIG)$

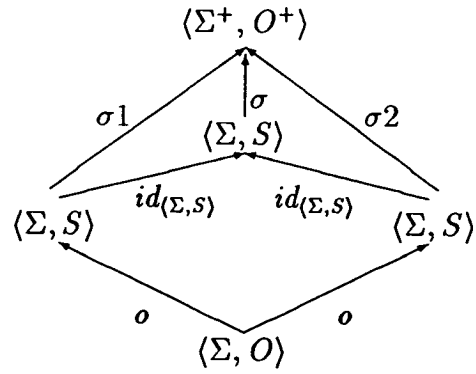


where S *is the set of sorts in* Σ *and* $O \subseteq S$*, then the pushout signature is also* $\langle \Sigma, S \rangle$.

Proof Assume there exists a signature $\langle \Sigma^+, O^+ \rangle$ and a pair of signature morphisms $\sigma 1, \sigma 2 : \langle \Sigma, S \rangle \rightarrow \langle \Sigma^+, O^+ \rangle$ such that $o; \sigma 1 = o; \sigma 2$ (the diagram commutes).

Since signature morphisms in $Beh(SIG)$ are also signature morphisms in SIG, we can consider the original category of signatures, so that the new two morphisms must satisfy $id_\Sigma; \sigma 1 = id_\Sigma; \sigma 2$, hence $\sigma 1 = \sigma 2$. This proves that there exists a

morphism σ from $\langle \Sigma, S \rangle$ to $\langle \Sigma^+, O^+ \rangle$; e.g. $\sigma = \sigma 1 = \sigma 2$



Since the top part of the diagram must also commute, $\sigma 1 = id_{\langle \Sigma, S \rangle}; \sigma$, and identities are cancellable so uniqueness of σ follows immediately.

□

In this situation, techniques based on the pushout of signatures turn out to be irrelevant since proving that the axioms of the consequent follow from the antecedent in the pushout signature is the same as ignoring abstraction, *e.g.* using the rule

$$\frac{SP2 \models SP1}{T_o D_o SP2 \models T_o D_o SP1}$$

The lack of "proper" pushouts compels us to use abstracted institutions with a *richer* category of signatures such as SIG_F in the next section, or to use the $\exists^2$-strategy where a pushout of signatures is not required.

Apart from the lack of a "proper" pushout of signatures, **Beh(EQ)** has still another problem regarding the application of the inheriting strategy.

The inheriting strategy is based on the "inheritance" of some axioms $\Phi 1_h$ specifying some auxiliary hidden symbols from the consequent to the antecedent. In the context of abstraction, the *auxiliary symbols* include a sort which is not hidden but made non-observable; then, $\Phi 1_h$ should include those axioms defining

the auxiliary sort. Unfortunately, in **Beh(EQ)** it does not make sense for an axiom to specify a sort; in other words, it is not possible to divide the equations of a specification in two, those specifying the sorts and those specifying the functions on those sorts.

This situation is improved in the institution of first order logic where equality is defined as an ordinary predicate. In this case, the definition of the equality predicate for non-observable sorts plays the role of $\Phi 1_h$ in the inheriting strategy. Also in a institution with equational logic and reachability constraints (see next chapter), the situation is improved since equations among generators and the reachability constraint itself can play the role of $\Phi 1_h$.

Using the $\exists^2$-strategy

Due to the difficulties discussed above entailment proofs in **Beh(EQ)** as in Task 1 have to rely on the $\exists^2$-strategy:

$$\frac{D_\sigma(A_\Phi T_{\sigma 1;\sigma})D_{\sigma 2}SP2 \models SP1 \qquad D_{\sigma 2}SP2 \models D_{\sigma 1;\sigma}(A_\Phi T_{\sigma 1;\sigma})D_{\sigma 2}SP2}{D_{\sigma 2}SP2 \models D_{\sigma 1}SP1}$$

where $(A_\Phi T_{\sigma 1;\sigma})$ is an enrichment to the antecedent (implementation) which must be guessed. Considering $\sigma 1$ and $\sigma 2$ to be an abstraction morphism $o : \Pi \to \Sigma$ we obtain

$$\frac{D_\sigma(A_\Phi T_\sigma)T_o D_o SP2 \models SP1 \qquad D_o SP2 \models D_{o;\sigma}(A_\Phi T_{o;\sigma})D_o SP2}{D_o SP2 \models D_o SP1}$$

where Φ is a set of Σ-sentences.

The second premise of the rule requires $(A_\Phi T_{o;\sigma})$ to be persistent w.r.t. $D_o SP2$. In the particular case where σ is the identity, it means that Σ-sentences in Φ should not exclude any model (behaviour) in $Mod[D_o SP2]$; hence Φ may include Σ-sentences without observable consequences. Phrasing the same idea the other way round we obtain the following proposition:

Proposition 5.4.2 *Given an observational and well-behaved institution* $\alpha_{Obs}\mathbf{I}$, *an abstraction morphism* $o : \Pi \rightarrow \Sigma$, *specifications* $SP2$ *and* $SP1$ *over* Σ *and a set of* Σ*-sentences* Φ *such that no consequence of* Φ *is observable in* Π, *i.e.* $Cl(\Phi) \cap Obs_\Pi = \emptyset$, *then*

$$\frac{A_\Phi\, T_o D_o SP2 \models SP1}{D_o SP2 \models D_o SP1}$$

is sound.

Proof Take an arbitrary model M in $Mod[D_oSP2]$. According to the semantics of D_o, M is the o-reduct of a Σ-model A in $Mod[SP2]$,

$$M = A|_o \qquad\qquad A \in Mod[SP2]$$

Divide Obs_Π into two sets of observations O and $\overline{O}$ such that

$$A \models_\Sigma O \qquad\qquad A \not\models_\Sigma \overline{O}$$

Since Φ has no observable consequences, A satisfies $(Cl(\Phi) \cap Obs_\Pi) \cup O$ and, by definition of well-behaved institution, there exists a Σ-model B such that

$$B \models_\Sigma \Phi \cup O \qquad\qquad B \not\models_\Sigma \overline{O}$$

Moreover, B satisfies the same observations as A therefore, in an observational institution, we know that

$$M = A|_o = B|_o$$

Considering that the models of $D_o(A_\Phi\, T_o)D_oSP2$ are:

$$\{B|_o \mid B \models_\Sigma \Phi \wedge (\exists\, A \in Mod[SP2].\ A|_o = B|_o)\}$$

we can conclude that $M \in Mod[D_o(A_\Phi\, T_o)D_oSP2]$ and in general

$$D_oSP2 \models D_o(A_\Phi\, T_o)D_oSP2$$

Considering that this entailment holds, the rule proposed is a particular case of the $\exists^2$-strategy, hence soundness follows.

□

In order to understand well the strength of the $\exists^2$-strategy it is important noting that $SP2$ and Φ affect the specification

$$A_\Phi T_o D_o SP2$$

at different levels, contrary to what happens in a specification such as $T_o D_o A_\Phi SP2$. In **Beh(ALG)**, the models of $A_\Phi T_o D_o SP2$ are algebras which satisfy Φ and have a behaviourally equivalent algebra in $Mod[SP2]$ whereas, in the second specification, models are algebras with a behaviourally equivalent algebra which is a model of $SP2$ and satisfies Φ. Therefore, a model of $A_\Phi T_o D_o SP2$ may have a behaviour the algebras of which cannot be a model of $SP2$ and satisfy Φ at the same time, hence

$$A_\Phi T_o D_o SP2 \not\models T_o D_o A_\Phi SP2$$

In relation to the $\exists^2$-strategy, this means that an enrichment $(A_\Phi T_o)$ may be persistent w.r.t. $D_o SP2$ despite the fact that neither $(A_\Phi T_{id_\Sigma})$ nor $(T_o D_o A_\Phi T_{id_\Sigma})$ are persistent w.r.t. $SP2$. This last situation arises in the following example.

Consider the following purported entailment:

$$D_o SP2 \models D_o SP1$$

where $SP2$ and $SP1$ are specifications as in figure 5–1 and o is an abstraction morphism which makes sort s non-observable, *i.e.* $o : \langle\langle\{s\},\{a,b\}\rangle,\emptyset\rangle \rightarrow \langle\langle\{s\},\{a,b\}\rangle,\{s\}\rangle$.

Using the $\exists^2$-strategy and taking σ to be the identity morphism and Φ to be $\{a \neq b\}$, we are required to prove that

$$A_{\{a\neq b\}} T_o D_o SP2 \models SP1$$

$$D_o SP2 \models D_o(A_{\{a\neq b\}} T_o) D_o SP2$$

The first entailment holds trivially since the only axiom in $SP1$ follows immediately from the antecedent using the inference rules shown for A_Φ in chapter 2.

```
SP1 =    sorts s
         operations
           a:  s
           b:  s
         axioms a≠b
         end

SP2 =    sorts s
         operations
           a:  s
           b:  s
         axioms a = b
         end
```

Figure 5–1:

The second entailment requires the persistency of the chosen enrichment. In an observational and well-behaved institution this is also trivial since $a \neq b$ has no observable consequences. Otherwise, we should check that each algebra satisfying $a = b$ is behaviourally equivalent to another algebra satisfying $a \neq b$ w.r.t. the set $\emptyset$ of observable sorts. In this case, this is not difficult to verify since no sort is observable and $\{a \neq b\}$ is a consistent presentation.

On the other hand, it is clear that $A_{\{a \neq b\}}SP2$ is inconsistent and therefore

$$D_o SP2 \not\models D_o A_{\{a \neq b\}} SP2 \qquad D_o SP2 \not\models D_o T_o D_o A_{\{a \neq b\}} SP2$$

When this kind of situation does not arise, we can use the following simplified rule

$$\frac{A_\Phi T_o D_o SP2 \models SP1 \qquad D_o SP2 \models D_o A_\Phi SP2}{D_o SP2 \models D_o SP1}$$

since

$$D_o A_\Phi SP2 \models D_o A_\Phi T_o D_o SP2$$

follows immediately from $SP2 \models T_o D_o SP2$ and the monotonicity of the SBO's.

5.4.2 Behavioural abstraction and first order logic

Work done in the context of the algebraic institution **EQ** can be repeated for **FOL** and **FOLEQ** considering predicates to be functions delivering values of a distinguished observable sort `Bool`. In this section we take a different but rather common approach, suitable to institutions where equality is treated as an ordinary predicate which does not need to be the identity in the models. In this case, pushouts of abstracted signatures can contain two different equality relations, one from the consequent specification and one from the antecedent specification, leading to proof techniques based on the handling of different equalities during the proof of refinement, as in [Wan 82]. This approach generalizes the so-called *ultra-loose* approach where equality is treated as an ordinary predicate [WB 89], in this case visible.

Technically first order signatures (from SIG_F) play the role of the signatures of the abstracted institution among which we distinguish a subcategory of signatures to play the role of "original" signatures, namely, those signatures with a single predicate "$\equiv_s$" for each sort s.

Definition 5.4.3 *Recall that SIG_F is the category of multisorted first order signatures as defined in section 2.5. $SIG_{\equiv}$ is its full subcategory with objects: pairs $\langle \Sigma, Q \rangle$ where $\Sigma = \langle S, \Omega \rangle$ is an algebraic signature, $\Sigma \in |SIG|$, and Q is a S-sorted set where each Q_s is a singleton $\{\equiv_s\}$.*

Then, the inclusion functor $SIG_{\equiv} \hookrightarrow SIG_F$ forms a reflection with a functor C as follows:

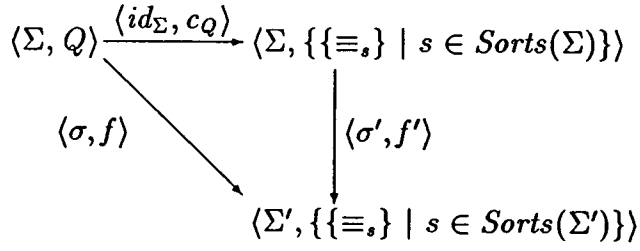
Lemma 5.4.4 *A functor* $C : SIG_F \to SIG_{\equiv}$ *defined over every signature* $\langle \Sigma, Q \rangle$ *and morphism* $\langle \sigma, f \rangle : \langle \Sigma 1, Q1 \rangle \to \langle \Sigma 2, Q2 \rangle$ *in* SIG_F *as*

$$C(\langle \Sigma, Q \rangle) = \langle \Sigma, \{\{\equiv_s\} \mid s \in Sorts(\Sigma)\} \rangle$$

$$C(\langle \sigma, f \rangle) = \langle \sigma, \{P \mapsto \equiv_{\sigma(s)} \mid P \in Q1_s \text{ and } s \in Sorts(\Sigma 1)\} \rangle$$

is a left adjoint of the inclusion $SIG_{=} \hookrightarrow SIG_F$ *with unit* $\mathbf{o}$ *where* $o_{\langle \Sigma, Q \rangle} = \langle id_\Sigma, c_Q \rangle$ *and* c_Q *is the unique* S*-sorted function from each set of relations* Q_s *to the singleton* $\{\equiv_s\}$.

Proof We shall prove that for each signature $\langle \Sigma, Q \rangle$ in SIG_F, its unit morphism $\langle id_\Sigma, c_Q \rangle : \langle \Sigma, Q \rangle \to \langle \Sigma, \{\{\equiv_s\} \mid s \in Sorts(\Sigma)\} \rangle$ is a universal arrow.



Let $\langle \sigma, f \rangle$ be an arbitrary morphism as in the diagram, then there is a unique $\langle \sigma', f' \rangle$ making the diagram commute $\langle id_\Sigma, c_Q \rangle ; \langle \sigma', f' \rangle = \langle \sigma, f \rangle$ since:

1. there is a unique σ' such that $id_\Sigma ; \sigma' = \sigma$, that is $\sigma' = \sigma$,

2. there is a unique function f' sorted by $Sorts(\Sigma)$ with codomain $\{\{\equiv_s\} \mid s \in Sorts(\Sigma')\}$ since each $\{\equiv_s\}$ is a singleton, and singletons are final objects in *Set*.

□

Considering the institution of first order logic **FOL**, we can define an institution $\mathbf{FOL}_{\equiv}$ which restricts **FOL** to the category of signatures $SIG_{\equiv}$.

Taking into account the functor C in the previous lemma, the institution morphism (inclusion) from $\mathbf{FOL}_{\equiv}$ to **FOL** is an abstracting institution morphism. Therefore, hiding of equality can be formalized as an abstraction operator along an abstraction morphism such as $\langle id_\Sigma, c_Q \rangle$.

An abstraction operation $T_{\langle id_\Sigma, c_Q \rangle} D_{\langle id_\Sigma, c_Q \rangle}$ w.r.t. an arbitrary abstraction morphism $\langle id_\Sigma, c_Q \rangle : \langle \Sigma, Q \rangle \rightarrow \langle \Sigma, \{\{\equiv_s\} \mid s \in Sorts(\Sigma)\} \rangle$ in **FOL** relaxes the meaning of equality of those sorts with an empty Q_s in the abstracted signature. If Q_s is empty, $D_{\langle id_\Sigma, c_Q \rangle}$ hides "$\equiv_s$" and $T_{\langle id_\Sigma, c_Q \rangle}$ restores it with an arbitrary meaning, so that after the abstraction the equality symbol can denote an arbitrary predicate. If Q_s is not empty $D_{\langle id_\Sigma, c_Q \rangle}$ makes several copies of the identity and $T_{\langle id_\Sigma, c_Q \rangle}$ collapses them again, so that the resulting predicate is again the same.

Considering the inheriting strategy, abstraction w.r.t. $\mathbf{FOL}_{\equiv} \rightarrow \mathbf{FOL}$ has two advantages over abstraction w.r.t. $\mathbf{EQ} \rightarrow \mathbf{Beh(EQ)}$.

In first place, the category of abstract signatures SIG_F is rich enough to include "proper" pushouts.

Proposition 5.4.5 *The following diagram is a pushout in* SIG_F

$$\begin{array}{ccc} & \langle \Sigma, \{\equiv_s\} \uplus \{\equiv_s\} \rangle & \\ \langle id_\Sigma, i1 \rangle \nearrow & & \nwarrow \langle id_\Sigma, i2 \rangle \\ \langle \Sigma, \{\equiv_s\} \rangle & & \langle \Sigma, \{\equiv_s\} \rangle \\ \langle id_\Sigma, c_Q \rangle \searrow & & \swarrow \langle id_\Sigma, c_Q \rangle \\ & \langle \Sigma, \emptyset \rangle & \end{array}$$

where $i1$ *and* $i2$ *form a pushout in Set w.r.t. the two* c_Q *and* $\uplus$ *denotes disjoint union.*

Proof It must be a pushout since it is the pairing of two pushouts in *SIG* and *Set* respectively. □

Such pushouts are very important when proving entailment between abstracted specifications because they allow the hidden equalities of the antecedent and the consequent to be combined in a single signature, as is common practice in the case of hidden functions.

The second advantage concerns the identification of some axioms as specifying the hidden part. Since equality is treated as an ordinary predicate, some axioms specifying $\equiv$ are included in the specification. Then, equality predicates on non-observable sorts are regarded as auxiliary functions defined in the hidden enrichment and the axioms specifying these predicates are regarded as axioms $\Phi 1_h$ in the inheriting strategy.

This kind of abstraction is weaker than behavioural abstraction. Models of an abstracted signature are reducts forgetting the *equality symbols* but not the *carriers* of the non-observable sorts; consequently $A|_o = B|_o$ implies that $|A|_s = |B|_s$ for all sorts s, therefore $T_o D_o$ does not mean behavioural abstraction. Nevertheless, treating abstraction as the hiding of $\equiv$ results in a sound proof strategy and many practical implementations such as stacks implemented by array-pointer pairs, can be proven correct in this fashion.

There are still a number of issues which make the hiding of equality a very peculiar kind of of hiding.

We know from chapter 4 that the inheriting strategy is sound only for persistent hidden enrichments and complete when the hidden enrichment is independent w.r.t. the visible enrichment and the basic specification.

In the case of abstraction w.r.t. $\mathbf{FOL}_{\equiv} \rightarrow \mathbf{FOL}$, the definition of equality predicates frequently violates persistency and is very rarely independent.

Consider for example the specification of stacks *Stack* in figure 5–2[7]. Axioms like

$$\forall\ \texttt{s:stack; e:elem. pop(push(s, e))} \equiv_{stack} \texttt{s}$$

or even

$$\forall\ \texttt{s, s':stack. s} \equiv_{stack} \texttt{s'} \Rightarrow \texttt{pop(s)} \equiv_{stack} \texttt{pop(s')}$$

violate independence because the interpretation of *pop* (visible enrichment) is dependent on the interpretation of $\equiv_{stack}$ (hidden enrichment).

Usually lack of independence forces the user to *complete* those axioms defining the hidden part $\Phi 1_h$ (see section 4.7). In the case of *Stack* implemented by arrays with pointers, the extra axioms correspond to the definition of $\equiv_{stack}$ in terms of the implementation. This fact prevents, in the context of **FOL**, entailment proofs with abstracted consequents from being fully automatized.

A more serious problem is the lack of persistency. In the context of hiding auxiliary functions, this was regarded as a *bug* in the specification design (see section 4.7).

In the case of abstraction in **FOL**, persistency is violated if the definition of $\equiv_s$ for a non-observable sort s imposes requirements on the structure of the carrier set.

---

[7]Note that `predicates` are added to the concrete syntax defined in chapter 2 in order to represent specifications in *ASL*(**FOL**).

SElem = sorts stack, elem
predicates $\_\equiv_{elem}\_$: elem × elem
operations
axioms
∀ e:elem. $e\equiv_{elem}e$
∀ e1,e2:elem. $e1\equiv_{elem}e2 \Rightarrow e2\equiv_{elem}e1$
∀ e1,e2,e3:elem. $(e1\equiv_{elem}e2 \wedge e2\equiv_{elem}e3) \Rightarrow e1\equiv_{elem}e3$
end

Stack = Enrich *SElem* by
Hidden
sorts
predicates $\_\equiv_{stack}\_$: stack × stack
operations
axioms
∀ s:stack. $s\equiv_{stack}s$
∀ s1,s2:stack. $s1\equiv_{stack}s2 \Rightarrow s2\equiv_{stack}s1$
∀ s1,s2,s3:stack. $(s1\equiv_{stack}s2 \wedge s2\equiv_{stack}s3) \Rightarrow s1\equiv_{stack}s3$
in
sorts
predicates
operations
empty: stack
push: stack, elem -> stack
top: stack -> elem
pop: stack -> stack
axioms
[1] ∀ s:stack; e:elem. $pop(push(s, e))\equiv_{stack}s$
∀ s:stack; e:elem. $top(push(s, e))\equiv_{elem}e$
∀ s, s':stack.
$s\equiv_{stack}s' \Rightarrow (pop(s)\equiv_{stack}pop(s') \wedge top(s)\equiv_{elem}top(s'))$
∀ s, s':stack; e, e':elem.
$(s\equiv_{stack}s' \wedge e\equiv_{elem}e') \Rightarrow push(s, e)\equiv_{stack}push(s', e')$
end

For example, we can write first order axioms specifying $\equiv_s$ to be an equivalence relation such that $\neg(\mathtt{a} \equiv_s \mathtt{b})$ analogously to $SP1$ in figure 5–1. This means that no model can assign the same values to constants a and b, thus the definition of $\equiv_s$ (hidden enrichment) is not persistent w.r.t. an empty specification $\epsilon_{(\{s\},\{a,b\})}$. In consequence, implementations analogous to $SP2$ in figure 5–1 assigning the same value to non-observable constants a and b are incorrect.

In such cases the lack of persistency can be understood as an indication that the hiding of $\equiv_s$ is not a *good* method for treating abstraction.

This is not only true for the inheriting strategy but also for the $\exists^2$-strategy since in such an example no persistent enrichment to

$SP2$ =
```
sorts s
predicates _≡s_:  s × s
operations
  a:  s
  b:  s
axioms a≡sb
end
```

can define a new predicate $\equiv_{old\_s}$ satisfying $\mathtt{a} \not\equiv_{old\_s} \mathtt{b}$ and $\mathtt{a} \equiv_{old\_s} \mathtt{a}$ at the same time.

On the other hand, if only first order sentences of the form $\forall X.t1 \equiv_s t2$ are considered, then all hidden enrichments specifying $\equiv_s$ are persistent. This is in fact what happens in equational specifications such as *Stack* when equations are "read" as first order sentences in which the equality symbol denotes a congruence relation.

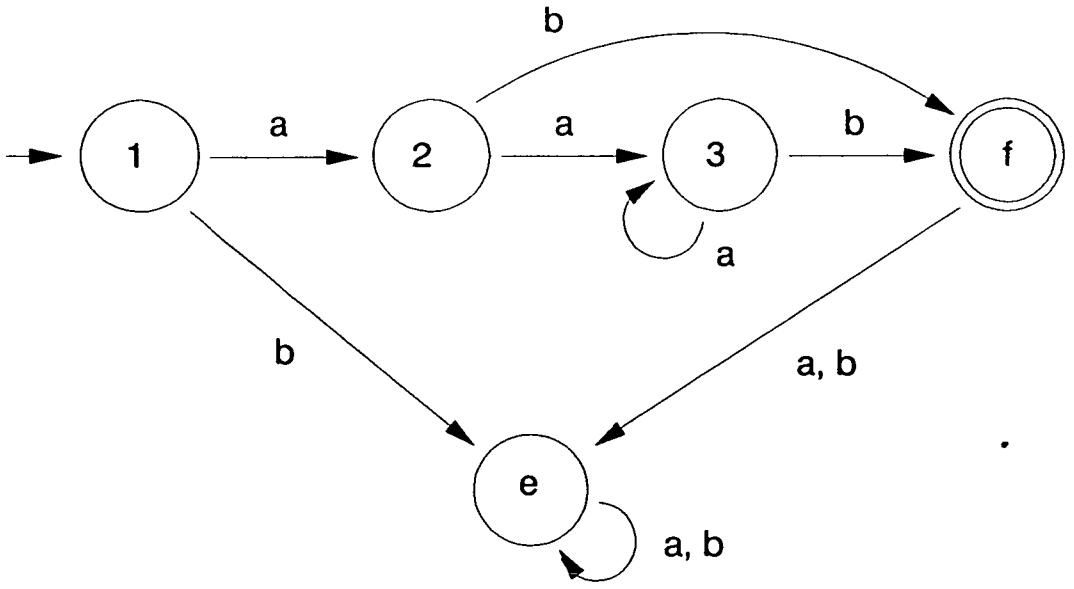


Figure 5–3: Automaton specified in *Automat*

5.5 Example

To close this chapter, we shall dedicate some time to the case of finite automata. This example was used in the introduction of the chapter and it is at the very origin of the whole issue of behavioural abstraction.

We shall sketch three correctness proofs for three similar specification entailments in **Beh(ALG)**. In each case, we prove that an abstract specification of the automaton *Automat* in figure 5–3 is implemented by a specification of the automaton *ImplAut* in figure 5–4, but the notion of abstraction is different in each case. This gives an idea of the trade-off between abstraction and proof complexity, as well as exhibiting the smooth transition from hiding to behavioural abstraction.

Consider the first order specification of a five-state automaton in figure 5–5. Intuitively, *Automat* specifies a five state automaton as in figure 5–3 with a final state `f` and an error state `e`. The transitions are computed by a function `t` and a predicate `v_check` determines the sequences of values accepted by the automaton,

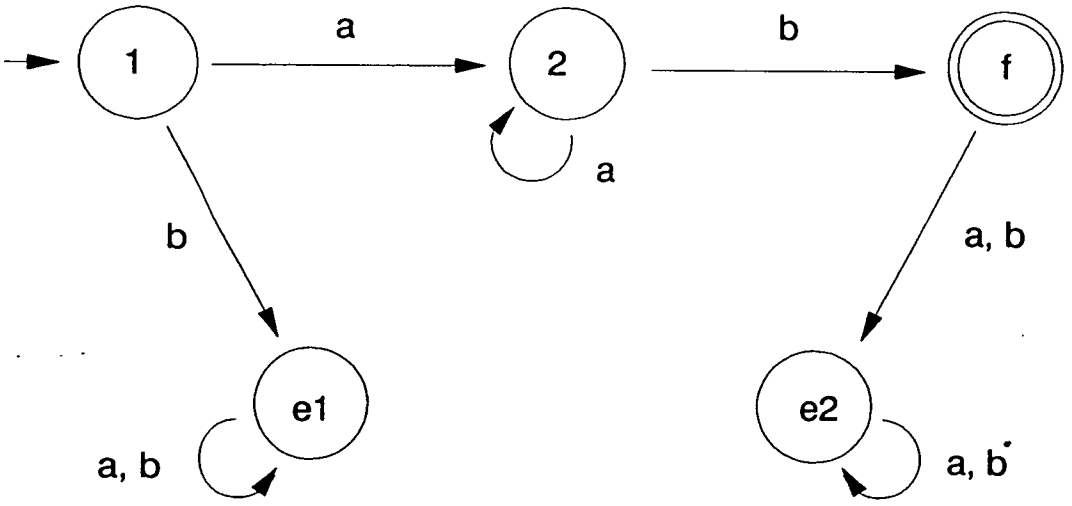


Figure 5–4: Automaton specified in *ImplAut*

in this case the language:

$$\texttt{aa}^*\texttt{b}$$

State 1 accepts the first a, state 2 accepts the second a or an ending b, and state 3 accepts successive a's or an ending b. The same language can be recognized by a four state automaton where 2 and 3 are combined into a single state.

Imagine that we choose an implementation with a single state for 2 and 3, and two error states discriminating the errors produced after the final state from the others, as in figure 5–4. This can be formalized as the specification *ImplAut* in figure 5–6.

Now we shall prove that $D_{\sigma 2}ImplAut \models D_{\sigma 1}Automat$ for appropriate morphisms $\sigma 2$ and $\sigma 1$. Three different cases are considered.

Visible v_check In the first case we choose $\sigma 1$ and $\sigma 2$ to be inclusions with source signature

$$\downarrow\sigma 1 = \downarrow\sigma 2 = \texttt{sorts v, bool, list of v}$$
$$\texttt{operations a, b: v}$$

```
Automat = sorts
          state, v, bool, list of v
          operations
            e,f,1,2,3:  state
            a,b:  v
            true, false:  bool
            []:  list of v
            _::_:  v, list of v -> list of v
            t:  state, v -> state
            check:  state, list of v -> bool
            v_check:  list of v -> bool
          axioms
            a≠b ∧ ( ∀ x:v. (x=a)∨(x=b) )
            true≠false ∧ ( ∀ x:bool. (x=true)∨(x=false) )
            {Some axioms defining list of v, [] and ::}†
            ∀ x:v. t(e,x)=e ∧ t(f,x)=e
            t(1,a)=2 ∧ t(1,b)=e ∧ t(2,a)=3 ∧ t(2,b)=f
            t(3,a)=3 ∧ t(3,b)=f
            ∀ s:state. (s=1)∨(s=2)∨(s=3)∨(s=e)∨(s=f)
            1≠2 ∧ 1≠3 ∧ 1≠e ∧ 1≠f
            2≠3 ∧ 2≠e ∧ 2≠f
            3≠e ∧ 3≠f
            e≠f
            check(f,[])= true
            ∀ s:state. (s≠f) ⇒ check(s,[])= false
            ∀ s:state; h:v; tl:list of v.
              check(s,h::tl)=check(t(s,h),tl)
            ∀ l:list of v. v_check(l)=check(1,l)
          end
```

Figure 5–5: Automaton recognizing aa^*b, version 1

---

†Details on the specification of lists of values are omitted. In the following chapter we shall see how reachability constraints can be used as sentences.

```
ImplAut =  sorts
           state, v, bool, list of v
           operations
             e1,e2,f,1,2:  state
             a,b:  v
             true, false:  bool
             []:  list of v
             _::_:  v, list of v -> list of v
             t:  state, v -> state
             check:  state, list of v -> bool
             v_check:  list of v -> bool
           axioms
             a≠b ∧ (∀ x:v. (x=a)∨(x=b))
             true≠false ∧ ( ∀ x:bool. (x=true)∨(x=false) )
             {Some axioms defining list of v, [] and ::}
             ∀ x:v. t(e1,x)=e1 ∧ t(e2,x)=e2 ∧ t(f,x)=e2
             t(1,a)=2 ∧ t(1,b)=e1 ∧ t(2,a)=2 ∧ t(2,b)=f
             ∀ s:state. (s=1)∨(s=2)∨(s=e1)∨(s=e2)∨(s=f)
             1≠2 ∧ 1≠e1 ∧ 1≠e2 ∧ 1≠f
             2≠e1 ∧ 2≠e2 ∧ 2≠f
             e1≠e2 ∧ e1≠f
             e2≠f
             check(f,[])= true
             ∀ s:state. (s≠f) ⇒ check(s,[])= false
             ∀ s:state; h:v; tl:list of v.
               check(s,h::tl)=check(t(s,h),tl)
             ∀ l:list of v. v_check(l)=check(l,l)
           end
```

Figure 5–6: Automaton recognizing aa^*b, version 2

```
true, false:  bool
[]:  list of v
_::_:  v, list of v -> list of v
v_check:  list of v -> bool
```

In other words, we hide `state, e, f, 1, 2, 3, t` and `check` in *Automat*. This is the case of a specification with a hidden part as defined in the previous chapter and not a proper abstraction as defined in this chapter

In order to apply the inheriting strategy the specification $D_{\sigma 1}\,Automat$ has to be re-written in the form of a specification with a hidden enrichment defining `state, e, f, 1, 2, 3, t` and `check`, and a visible enrichment defining `v_check`. The resulting specification looks like this:

```
Values =   sorts v, bool, list of v
           operations
             a,b:v
             true, false:  bool
             []:  list of v
             _::_:  v, list of v -> list of v
           axioms
             a≠b ∧ (∀ x:v. (x=a) ∨ (x=b) )
             ...
           end

Automat1 = Enrich Values by
           Hidden
             sorts state
             operations
               e,f,1,2,3:  state
               t:  state, v -> state
               check:  state, list of v -> bool
```

```
  axioms ...
in
  sorts
  operations
    v_check:  list of v -> bool
  axioms
    ∀ l:list of v. v_check(l)=check(1,l)
end
```

The hidden enrichment consists of all the axioms in *Automat* except those defining the visible sorts `v`, `list of v` and `bool` and the last axiom specifying `v_check`. Briefly, the non-trivial part of a proof according to the inheriting strategy is to prove that:

$$\left.\begin{array}{l}\text{Enrich } \textit{ImplAut} \text{ by the definition of} \\ \texttt{old\_state, old\_e, old\_1, ...} \\ \texttt{old\_t, old\_check} \\ \text{using axioms in } \textit{Automat}\end{array}\right\} \models \left\{\begin{array}{l}\forall \texttt{l:list of v.} \\ \quad \texttt{v\_check(l)=old\_check(old\_1,l)}\end{array}\right.$$

where `old_state, old_e, old_1, ...` are those symbols in the pushout signature corresponding to the sort `state` and constants `e, 1, ...` in *Automat*1.

In this case, the hidden enrichment is *persistent* w.r.t. the specification of sorts `v` and `bool` and it is also *independent* since the relevant states and functions of the hidden part are totally defined.

A theorem prover proceeding by induction on the list `l` completes the proof.

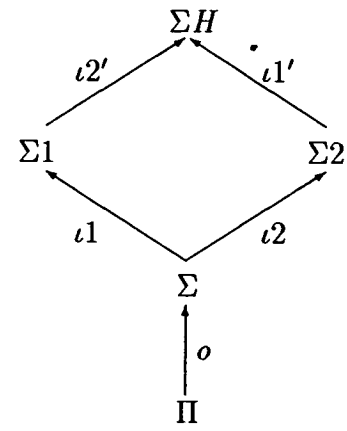
Visible `check` In the second case we choose $\sigma 1$ and $\sigma 2$ to be morphisms in $Beh(SIG)$ with the following abstract signature Π as source:

$$\Pi = {\downarrow}\sigma 1 = {\downarrow}\sigma 2 = \texttt{sorts v, bool, list of v}$$
$$\texttt{non\_obs\_sorts state}$$

```
operations a, b:  v
           true, false:  bool
           []:  list of v
           _::_:  v, list of v -> list of v
           check:  state, list of v -> bool
           v_check:  list of v -> bool
```

In other words, we make the function `check: state, list of v -> bool` visible and the sort `state` visible but non-observable.

In order to distinguish hiding from abstraction we decompose $\sigma 1$ and $\sigma 2$ into o; $\iota 1$ and into o; $\iota 2$ respectively, as in the diagram aside, so that the target signature of o is:




```
Σ = sorts v, bool, list of v, state
    operations a, b:  v
               true, false:  bool
               []:  list of v
               _::_:  v, list of v -> list of v
               check:  state, list of v -> bool
               v_check:  list of v -> bool
```

Then, we prove that

$$D_o D_{\iota 2} ImplAut \models D_o D_{\iota 1} Automat$$

applying the $\exists^2$-strategy simultaneously to the hiding and the abstraction steps. This is performed by adding a persistent enrichment $(A_\Phi T_{o;\,\iota 1;\,\iota 2'})$ to the antecedent, *i.e.* proving that

$$D_{\iota 2'} A_\Phi T_{\iota 1;\,\iota 2'} T_o D_o D_{\iota 2} ImplAut \models Automat$$

In this case the set of ΣH-sentences Φ is chosen to be the union of the following three sets:

1. Sentences specifying the hidden functions in $D_{\iota 1}Automat$:

   ```
   {∀ x:v. old_t(old_e,x)=old_e ∧ old_t(old_f,x)=old_e,
    old_t(old_1,a)=old_2 ∧ old_t(old_1,b)=old_e,
    old_t(old_2,a)=old_3 ∧ old_t(old_2,b)=old_f,
    old_t(old_3,a)=old_3 ∧ old_t(old_3,b)=old_f,
    ∀ s:state. (s=old_1)∨(s=old_2)∨(s=old_3)∨(s=old_e)∨(s=old_f),
    old_1≠old_2 ∧ old_1≠old_3 ∧ old_1≠old_e ∧ old_1≠old_f,
    ... }
   ```

   where `old_t`, `old_e`, `old_1`, ... are those symbols in the pushout signature corresponding to the function `t` and constants `e`, `1`, ... in *Automat*.

2. Equations between states in *Automat* and states in *ImplAut*:

   ```
   {old_1=1 , old_2=2 , old_e=e1 , old_e=e2 , old_f=f }
   ```

3. All sentences specifying the hidden functions in $D_{\iota 2}ImplAut$ except axioms:

   ```
   {∀ s:state. (s=1)∨(s=2)∨(s=e1)∨(s=e2)∨(s=f),
    e1≠e2}
   ```

   Adding the hidden functions in $D_{\iota 2}ImplAut$ amounts to ignore $D_{\iota 2}$ during the proof, as it happens in the inheriting strategy. In this case, however, this cannot be done since these two axioms are inconsistent with the previous sets of sentences already included in Φ. Excluding these axioms solves the problem.

The sentences in Φ are consistent among themselves and with the observations from $D_o D_{\iota 2}ImplAut$. This fact is enough to guarantee the persistency of

$(A_\Phi\, T_{\iota 1;\, \iota 2'}\, T_o)$ w.r.t. $D_o D_{\iota 2} ImplAut$ since the specification $D_o D_{\iota 2} ImplAut$ defines a unique behaviour (up to isomorphism).

Finally, the proof is concluded by proving that:

$$\Phi \models \begin{cases} \texttt{check(old\_f,[])= true} \\ \forall\ \texttt{s:state. (s}\neq\texttt{old\_f)} \Rightarrow \texttt{check(s,[])= false} \\ \forall\ \texttt{s:state; h:v; tl:list of v.} \\ \quad \texttt{check(s,h::tl)=check(old\_t(s,h),tl)} \\ \forall\ \texttt{l:list of v.} \\ \quad \texttt{v\_check(l)=check(old\_1,l)} \end{cases}$$

which follows from the definition of check in Φ (taken from *ImplAut*) considering that old_1=1 and old_f=f in the first, second and fourth sentence, and working case by case through each possible state s and value h in the third sentence.

Visible t Analogously to the previous case, we could also make the transition function t: state,v -> state visible and come to another solution by taking Φ to include the definition of t in *ImplAut* instead of the definition of t in *Automat* and proving that:

$$\Phi \models \begin{cases} \texttt{check(old\_f,[])= true} \\ \forall\ \texttt{s:state. (s}\neq\texttt{old\_f)} \Rightarrow \texttt{check(s,[])= false} \\ \forall\ \texttt{l:list of v.} \\ \quad \texttt{v\_check(l)=check(old\_1,l)} \\ \forall\ \texttt{x:v. t(old\_e,x)=old\_e} \wedge \texttt{t(old\_f,x)=old\_e} \\ \texttt{t(old\_1,a)=old\_2} \wedge \texttt{t(old\_1,b)=old\_e} \wedge \texttt{t(old\_2,a)=old\_3} \\ \texttt{t(old\_2,b)=old\_f} \wedge \texttt{t(old\_3,a)=old\_3} \wedge \texttt{t(old\_3,b)=old\_f} \end{cases}$$

which follows from the definition of t in Φ (taken from *ImplAut*) considering that old_1=1, old_2=2, old_e=e1, old_e=e2 and old_f=f. Note that, unlike in the

previous case, the axiom

```
∀ s:state; h:v; tl:list of v. check(s,h::tl)=check(t(s,h),tl)
```

does not need to be considered since it is directly included in Φ.

There are two conclusions we can extract from these three proofs.

In first place, the situation of having a specification with a hidden part or a behaviourally abstract specification are very similar. The difference is just a technicality depending on which functions and sorts we want to hide; for example if we want to "hide" sort `state` but let `check` or `t` be visible, then `state` has to be considered as a visible but non-observable sort and the whole specification behaviourally abstracted.

The second conclusion regards the complexity of entailment proofs. The more we hide/abstract the consequent specification the harder we have to work at the level of theorem proving.

In the first case of the example we hide more than in the other two cases and a theorem prover has to perform induction over lists of values. In the last case, we hide very little and, although quite a lot of work may be involved in checking the persistency of the chosen enrichment (checking consistency of Φ with the observable consequences of *ImplAut*), the actual amount of theorem proving is reduced to a few simple substitutions.

On the other hand, the use of proper abstraction in the last two cases requires choosing the right set of sentences Φ. This demands a creative task from the prover and a proof of persistency which might be very difficult to obtain.

Interpreting specifications *Automat* and *ImplAut* in first order logic and treating abstraction as the hiding of the equality predicate $\equiv_{state}$ between states does

not lead to a correct proof of the entailment since axioms in *Automat* require states 2 and 3 to be different. This will also happen in the *ultra-loose* approach.

Chapter 6

Specifications with reachability constraints

6.1 Introduction

This chapter is concerned with entailments where specifications use $M_{\langle\sigma,\iota\rangle}$, and reachability constraints in particular. We shall survey problems such as $MTU \models MTU$, $DMTU \models DMTU$ and $ASL \models ASL$. On the whole, we go over the same problems tackled before, but considering constraints $M_{\langle\sigma,\iota\rangle}$ and a mixture of constraints and axioms MA, instead of just axioms A_Φ.

Some ideas from the theory of institutions, in particular, duplex institutions, allow operations such as $M_{\langle\sigma,\iota\rangle}$ to be viewed as a special case of $A_{\{\varphi\}}$. The only problem is that, contrary to more common institutions, duplex institutions $\mathbf{I}$ generally lack an underlying inference system $\vdash^{\mathbf{I}}$.

The reasons why a language with institution independent semantics distinguishes two operations A_Φ and $M_{\langle\sigma,\iota\rangle}$ are mainly historical. Frequently, however, there is a well-known inference system for sentences whereas there is no inference system available for sentences combined with constraints.

The proving techniques related to languages ATU, $DATU$, $DATU^*$ and $\alpha +$ ATU extend immediately to $MATU$, $DMATU$ (*i.e.* ASL), $DMATU^*$ (also called

ASL^*) and $\alpha + MATU$ respectively. However, a inference system dealing with combined sets of sentences and constraints is needed.

Since the institution independent treatment of $M_{\langle\sigma,\iota\rangle}$ with respect to proofs is very poor, we shall devote most of this chapter to working out an inference system for reachability constraints in **EQ**. This can be understood as a typical example of combining two different kinds of sentences with two different inference systems into one institution. Using such a popular institution also allows the whole system to be put into practice for useful specifications.

Section 2 recalls those results in the theory of institutions which allow $M_{\langle\sigma,\iota\rangle}$ to be regarded as a special case of A_Φ. Section 3 studies the particular case of reachability constraints in **EQ**, presenting an inference system $\Vdash$ to cope with $M \models M$ and an inference system $\Vdash^{\mathbf{EQ}}$ in order to cope with $MA \models MA$. Section 4 studies $MATU$, $DMATU$ and $\alpha + DMATU$ as straightforward extensions of ATU, $DATU$ and $\alpha + DATU$ where axioms are a mixture of ordinary sentences and constraints.

6.2 Institutions with constraints

Constraints $M_{\langle\sigma,\iota\rangle}$ can be considered as sentences provided their satisfaction is invariant under change of signature (satisfaction condition).

In order not to confuse a SBO $M_{\langle\sigma,\iota\rangle}$ with the corresponding sentence, we introduce the following notation:

Theorem 6.2.1 *Given an institution* $\mathbf{I} = \langle SIG, Mod, Sen, \models\rangle$, *there exists an* **institution with constraints** *over* $\mathbf{I}$, $\mathbf{I}' = \langle SIG, Mod, Sen', \models'\rangle$ *such that for all* $\Sigma \in |SIG|$

- $Sen'(\Sigma) = Sen(\Sigma) \cup \{\ll \sigma, \iota \gg \mid \sigma, \iota$ *are morphisms in* $SIG, \uparrow\sigma = \downarrow\iota$, and $\uparrow \iota = \Sigma\}$

- $Sen'(\sigma')$ *for a signature morphism* $\sigma' : \Sigma \to \Sigma'$ *extends* $Sen(\sigma')$ *to take account of the new sentences so that for all* Σ*-sentences* $\ll \sigma, \iota \gg$,

$$Sen'(\sigma')(\ll \sigma, \iota \gg) = \ll \sigma, (\iota; \sigma') \gg$$

- $\models'_\Sigma$ *extends* $\models_\Sigma$ *to take account of the new sentences so that for all* $M \in |Mod(\Sigma)|$ *and* Σ*-sentence* $\ll \sigma, \iota \gg$, $M \models'_\Sigma \ll \sigma, \iota \gg$ *iff* $M|_\iota$ *is* σ*-minimal.*

Proof It is not difficult to see that constraints $\ll \sigma, \iota \gg$, like ordinary sentences, satisfy the satisfaction condition. Considering an arbitrary signature Σ, a Σ-sentence $\ll \sigma, \iota \gg$, a signature morphism $\sigma' : \Sigma \to \Sigma'$ and a Σ-model M,

$$M \models'_{\Sigma'} \ll \sigma, (\iota; \sigma') \gg \text{ iff } M|_{\sigma'} \models'_\Sigma \ll \sigma, \iota \gg$$

holds trivially since $(M|_{\sigma'})|_\iota = M|_{(\iota; \sigma')}$, hence both sides are satisfied if and only if $M|_{(\iota; \sigma')}$ is σ-minimal. □

A similar situation is explained and generalized in [GB 90], where an institution **C(I)** is constructed by adding data constraints to an arbitrary institution **I**. Or, more generally, a duplex institution $\mathbf{D}(\alpha)$ can be defined for an arbitrary institution morphism α.

It is also usual practice to prove that a particular kind of constraint satisfies the satisfaction condition and add it directly to the original institution, *e.g.* bounded data constraints in [Mos 89].

In terms of the specification language, $M_{(\sigma,\iota)}$ becomes superfluous since we can choose a institution with constraints $\mathbf{I}'$ which includes constraints as sentences so that

$$M_{(\sigma,\iota)}SP \quad \text{can be written} \quad A_{\{\ll \sigma,\iota \gg\}}SP$$

We are generally concerned with inference systems for specifications over an arbitrary institution equipped with an underlying inference system for its sentences $\vdash^{\mathbf{I}}$. Therefore, if constraints are included in the institution, the treatment of M is transferred to the institution level which should provide an underlying inference system $\vdash^{\mathbf{I}'}$.

We may wonder if this is fair and why other SBO's are not transferred to the institution level. The difference is that the rest of the operations — T_σ, U, D_σ and α — have a proof theoretical treatment at the institution independent level, whereas $M_{\langle\sigma,\iota\rangle}$ only has institution independent semantics in terms of the morphisms in the category of models, and these morphisms are completely independent from the rest of the institution, in particular, independent of the sentences and their satisfaction. Therefore, nothing substantial can be said about proving entailment between specifications with constraints $M_{\langle\sigma,\iota\rangle}$ in an arbitrary institution.

Attempts to improve this situation propose to enrich the definition of institution, which would make it feasible to give $M_{\langle\sigma,\iota\rangle}$ a proof theoretical treatment at the institution independent level. In this direction, some work is in progress [GB 86, Poi 89, Mes 89].

6.3 Reachability constraints

Reachability and data constraints are introduced in specification languages in order to increase their expressive power. In particular, they are useful for the definition of standard models up to isomorphism.

It is known that axiomatizations in **FOL** cannot be categorical unless they describe a finite model, by the Löwenheim-Skolem theorem. Moreover, even if we restrict our models to countable ones, it is not always the case that a $\aleph_0$-categorical axiomatization exists; *e.g.* sound axiomatizations of arithmetic always include

non-standard countable models. Summing up, implicit or explicit reachability constraints are needed if we want to be able to talk about things such as the standard model of arithmetic in a first order language.

In nature, a reachability constraint can be seen as a second order axiom or an infinitary axiom. For example, arithmetic can be defined up to isomorphism by

$$\forall P.\ (P(0) \wedge (\forall n.\ P(n) \Rightarrow P(suc(n))) \Rightarrow (\forall n.\ P(n))$$

or by

$$\forall n.\ (n = 0) \vee (n = suc(0)) \vee (n = suc(suc(0))) \vee \ldots$$

However, there is no complete inference system for second order logic [Bar 77], and inference systems for infinitary logic $L_{\omega_1\omega}$ use infinitary rules in order to be complete [Sco 66].

On the other hand, a reachability constraint provides a sound induction schema which can be added to our inference system as described in in section 3.3.2 and, what is more important, entailments can still be proved correct by relating the reachability constraints used in the antecedent and the consequent. In fact, what is a disadvantage when we try to prove theorems from a constrained specification, becomes an advantage when proving entailment.

For example, in **FOL** an entailment $SP2 \models M_{\{s\}}SP1$ can only be correct if $SP2$ also contains a reachability constraint on sort s or all the models of $SP2$ have a finite carrier for sort s. Otherwise, by the Löwenheim-Skolem theorem, there would exist some infinite models of $SP2$ having a carrier of s of a higher cardinality than in any model of $M_{\{s\}}SP1$.

But before we start intermixing constraints with other sentences, let us focus on a simpler case.

[Bar 77] Ch. A.1 , §5 and [BJ 80] page 203

6.3.1 A logic of reachability constraints

In this section we produce an inference system for specifications $M \models M$ in **EQ**. Or, in other words, we find an adequate underlying inference system for an algebraic institution whose only sentences are reachability constraints. We shall call this system $\vdash\!\!\!\vdash$.

First, an inference system can be given capturing those cases where a constraint is strictly *stronger* than another one; *e.g.* when the generators of every sort are preserved or reduced in the antecedent, while the set of constrained sorts is preserved or enlarged. These cases correspond to trivial cases that can be automatically checked, and indeed they are very much used in practice.

We start by recalling some notation from chapter 2. There, we decided that algebraic specifications $M_{\langle\sigma,\iota\rangle}SP$ for signature inclusions σ and ι as follows:

$$\langle S, \emptyset\rangle \overset{\sigma}{\hookrightarrow} \langle S \cup G, F\rangle \overset{\iota}{\hookrightarrow} \langle S', \Omega'\rangle$$

will be written $M_{\langle G,F,S\rangle}SP$. Moreover, in this situation, an algebra over $\langle S \cup G, F\rangle$ is σ-minimal if it is reachable on sorts G. In other words, given a $\langle S', \Omega'\rangle$-algebra A its ι-reduct is σ-minimal if there is a S-sorted set of variables X (disjoint from Ω) such that for every value $v \in |A|_r$ of a sort $r \in G$ there exists a term $t_v \in |T_{\langle S\cup G,F\rangle}(X)|$ and a valuation $\nu : T_{\langle S\cup G,F\rangle}(X) \to A|_\iota$ such that $\nu(t_v) = v$.

Reachability constraints are sentences in an institution with constraints over **EQ**, therefore a reachability constraint $M_{\langle G,F,S\rangle}$ is a $\langle S', \Omega'\rangle$-sentence $\ll G, F, S \gg$. In general, a triple $\ll G, F, S \gg$ is a $\langle S', \Omega'\rangle$-sentence if G and S are disjoint. $G \cup S \subseteq S'$, functions names in F use sorts in $S \cup G$ and $F \subseteq \Omega'$. Henceforth. Σ-sentences which happen to be constraints are called Σ-constraints.

In this section we shall restrict our attention to reachability constraints of the form $\ll G, F, S \gg$. This can be done without loss of generality if we consider

only injective signature morphisms, so that a reachability constraint $\ll G, F, S \gg$ translated along σ' is equivalent to $\ll \sigma'(G), \sigma'(F), \sigma'(S) \gg$. For non-injective morphisms this is not generally true since two generated sorts could be identified into a single sort having as generators all the generators of the two original sorts.

In the following we shall define which are the *generator operations* of a generated sort and which generated sorts are *one-point.*

Definition 6.3.1 *Given a signature* $\langle S', \Omega' \rangle$, *the* **generators** *of a generated sort* $r \in G$ *w.r.t. a* $\langle S', \Omega' \rangle$*-sentence* $\ll G, F, S \gg$, *are the constants* $k : r$ *and operations* $f : \overline{s} \rightarrow r$ *used in the set of terms* $|T_{\langle S \cup G, F \rangle}(X_S)|_r$ *where* X_S *is a non-empty* S*-indexed set of variables. The generators of* r *w.r.t.* $\ll G, F, S \gg$ *are denoted by* F_r.

A generated sort $r \in G$ *is* **one-point** *w.r.t. a* $\langle S', \Omega' \rangle$*-sentence* $\ll G, F, S \gg$ *if* $|T_{\langle S \cup G, F \rangle}(X_S)|_r$ *is a singleton for* X_S *being a* S*-indexed set of variables with at least two variables of each sort.*

A $\langle S', \Omega' \rangle$*-sentence* $\ll G, F, S \gg$ *is* $\emptyset$**-free** *iff for all sorts* r *in* G,

$$|T_{\langle S \cup G, F \rangle}(X_S)|_r \neq \emptyset$$

where X_S *is a non-empty* S*-indexed set of variables.*

Note that the union of all generators $\bigcup_{r \in G} F_r$ can be smaller than F due to an operation $op : \overline{s} \rightarrow s \in F$ over an unconstrained sort $s \notin G$ or due to a sort $r \in G$ which has no terms $|T_{\langle S \cup G, F \rangle}(X_S)|_r = \emptyset$ despite having an operation $op : \overline{s} \rightarrow r \in F$.

Consider the following example:

$$S = \{s1, s2\} \qquad \Omega = \{f1 : s1 \rightarrow s2$$

$$G = \{s3, s4\} \qquad F = \{f2 : s1, s2 \rightarrow s3$$
$$f3 : s2, s3 \rightarrow s1$$
$$f4 : s4 \rightarrow s3$$
$$f5 : s4 \rightarrow s4\}$$
$$c1 : s2$$
$$c2 : s2\}$$

In the term model $T_{\langle S \cup G, \Omega \cup F \rangle}$, the carriers of $s1$, $s3$ and $s4$ are empty, and $|T_{\langle S \cup G, \Omega \cup F \rangle}|_{s2} = \{c1, c2\}$. In the free generated model $T_{\langle S \cup G, F \rangle}(X_S)$, the carrier of $s4$ is still empty but the carriers of $s1$, $s2$ and $s3$ are infinite:

$$|T_{\langle S \cup G, F \rangle}(X_S)|_{s1} = \{x, z, f3(y, f2(x, y)), f3(y, f2(x, f2(z, y))), ...\}$$
$$|T_{\langle S \cup G, F \rangle}(X_S)|_{s3} = \{f2(x, y), f2(x, f2(z, y)), f2(f3(y, f2(x, y)), y), ...\}$$

for all $x, z \in X_{s1}$ and $y \in X_{s2}$. At the same time $F_{s3} = \{f2\}$ and $F_{s4} = \emptyset$.

Using these definitions it is possible to prove the following results about consistency in a language of reachability constraints.

Lemma 6.3.2 *All models of a Σ-constraint $\ll G, F, S \gg$ are isomorphic to a persistent extension of a quotient of $T_{\langle S \cup G, F \rangle}(X_S)$ for some S-indexed set of variables X_S.*

Proof Given an arbitrary Σ-algebra $A \models_\Sigma \ll G, F, S \gg$, take X_S to be an S-indexed set of variables with as many variables in each sort $s \in S$ as values in $|A|_s$. By definition of finitely generated and construction of $T_{\langle S \cup G, F \rangle}(X_S)$, there is a surjective homomorphism $h : T_{\langle S \cup G, F \rangle}(X_S) \rightarrow A|_\iota$ for $\iota : \langle S \cup G, F \rangle \rightarrow \Sigma$. Considering the kernel induced by h, we conclude that $A|_\iota$ is isomorphic to $T_{\langle S \cup G, F \rangle}(X_S)/Ker(h)$.

□

Proposition 6.3.3 *Some results about consistency are the following:*

1. *Every individual $\langle S \cup G, F\rangle$-sentence $\ll G, F, S \gg$ is consistent*[1].

2. *Every finite set of $\emptyset$-free reachability constraints over a signature is consistent.*

Proof The first fact can be proven considering that a term generated model $T_{\langle S \cup G, F\rangle}(X_S)$ for any S-indexed set of variables X_S satisfies trivially $\ll G, F, S \gg$.

The second fact can be proven considering a final $\langle S', \Omega'\rangle$-algebra $\perp$. In this case all terms $|T_{\langle S \cup G, F\rangle}(X_S)|_s$ for every $\langle S', \Omega'\rangle$-sentence $\ll G, F, S \gg$, S-indexed set of variables X_S and sort $s \in S'$, evaluate to the same value in $\perp$. Hence the existence of a term for each sort in G guarantees reachability, $\perp \models_{\langle S', \Omega'\rangle} \ll G, F, S \gg$.

□

Despite these results, it is not easy to check if an arbitrary set of reachability constraints is consistent or not.

For example consider the following $\langle S', \Omega'\rangle$-constraints:

$$S' = \{s1, s2, s3\} \qquad \Omega' = \{f1 : s1 \rightarrow s2$$
$$f2 : s1 \rightarrow s3$$
$$c2 : s2$$
$$c1 = \ll \{s2, s3\}, \{f1, f2\}, \{s1\} \gg \qquad c2 = \ll \{s3\}, \emptyset, \emptyset \gg$$

---

[1]Note that a $\langle S', \Omega'\rangle$-sentence $\ll G, F, S \gg$ may not be consistent because a sort $r \in G$ has no generators but there exists a function symbol in $\Omega' \setminus F$ with co-arity r.

Constraint $c2$ requires the carrier of sort $s3$ to be empty since no generators or generating sorts are included. Constraint $c1$ requires carriers of $s2$ and $s3$ to be generated from $s1$ using functions $f1$ and $f2$; if $s3$ has an empty carrier $s1$ and $s2$ must also have empty carriers. Finally, since Ω' includes a constant $c2 : s2$, the carrier of sort $s2$ cannot be empty in any $\langle S', \Omega' \rangle$-algebra, hence the union of $c1$ and $c2$ is inconsistent.

Assumption *From now on, we assume our constraints to be $\emptyset$-free. This can be effectively checked and guarantees consistency, which is a rather natural condition for a specification building operation.*

Definition 6.3.4 *A consequence relation $\vdash$ between Σ-constraints is defined as $\ll G2, F2, S2 \gg \vdash \ll G1, F1, S1 \gg$ iff the following two conditions hold:*

1. *All generated sorts are inherited: $G1 \subseteq G2$.*

2. *For all generated sorts $r \in G1$, no new generators are added, $F1_r \supseteq F2_r$, or r is a one-point sort w.r.t. $\ll G2, F2, S2 \gg$.*

Theorem 6.3.5 *Given two $\emptyset$-free Σ-constraints $\ll G1, F1, S1 \gg$ and $\ll G2, F2, S2 \gg$, then*

$$\ll G2, F2, S2 \gg \models \ll G1, F1, S1 \gg \quad \textit{iff} \quad \ll G2, F2, S2 \gg \vdash \ll G1, F1, S1 \gg$$

Soundness proof Let $\iota 1$ and $\iota 2$ be the inclusions $\langle S1 \cup G1, F1 \rangle \hookrightarrow \Sigma$ and $\langle S2 \cup G2, F2 \rangle \hookrightarrow \Sigma$, A a Σ-model of $\ll G2, F2, S2 \gg$ and assume $\ll G2, F2, S2 \gg \vdash \ll G1, F1, S1 \gg$. Then, by definition of satisfaction, every value $v \in |A|_r$ for $r \in G2$ can be reached by some term $t \in |T_{\langle S2 \cup G2, F2 \rangle}(X_{S2})|_r$ for an $S2$-indexed set of variables X_{S2} and a valuation $\nu 2 : T_{\langle S2 \cup G2, F2 \rangle}(X_{S2}) \to A|_{\iota 2}$,

$$\nu 2(t) = v$$

Since $G1 \subseteq G2$, the same holds for all $r \in G1$.

Replace all subterms $t1, ..., tn$ of t of sorts in $S1$ by new variables $x1, ..., xn$ to yield a term t'. By construction of t, the new variables are of sorts in $S2 \cup G2$, hence there exists an $(S2 \cup G2)$-indexed set of variables X_S including those variables in X_{S2} and the new variables $x1, ..., xn$. Extend $\nu 2$ to a valuation $\nu 2' : T_{\langle S2 \cup G2, F2 \rangle}(X_S) \rightarrow A|_{\iota 2}$ such that $\nu 2'(xi) = \nu 2(ti)$ for each new variable xi.

If no new generators are added, $F1_r \supseteq F2_r$, we can show, by induction on the height of t', that $t' \in |T_{\langle S1 \cup G1, F1 \rangle}(X_{S1})|_r$ where X_{S1} is an $S1$-indexed set including variables $x1, ..., xn$.

Suppose that t' is $op(x)$ for a certain function symbol $op : s \rightarrow r$. If neither $s \in S1$ nor $s \in G1$, then $op \notin F1_r$ and therefore op cannot be a generator in $F2_r$ in contradiction with op belonging to a term $t \in |T_{\langle S2 \cup G2, F2 \rangle}(X_{S2})|_r$. If $s \in G1$ then $s \in G2$ as well but, by construction of t', the variable x can only be of a sort in $S1$ or in $S2$ and no sort can be in $G1 \cap G2$ and in $S1 \cup S2$ at the same time, hence we conclude that $s \in S1$. By construction of t', $x \in X_{S1}$ is a new variable and $op \in F1_r$ thus $t' \in |T_{\langle S1 \cup G1, F1 \rangle}(X_{S1})|_r$.

Suppose that t' is $op(t'')$ for a certain function symbol $op : s \rightarrow r$. As before, s must belong to either $S1$ or $G1$. If $s \in S1$ then, by construction t'' must be a variable x and the previous case applies. If $s \in G1$, by the induction hypothesis $t'' \in |T_{\langle S1 \cup G1, F1 \rangle}(X_{S1})|_s$ and since $op \in F1_r$, $t' \in |T_{\langle S1 \cup G1, F1 \rangle}(X_{S1})|_r$ as well.

If op has several parameters the same reasoning applies to each of them.

Summing up, $t' \in |T_{\langle S1 \cup G1, F1 \rangle}(X_{S1})|_r$ for an $S1$-indexed set of variables X_{S1} and there is a valuation $\nu 1 : T_{\langle S1 \cup G1, F1 \rangle}(X_{S1}) \rightarrow A|_{\iota 1}$ such that $\nu 1(xi) = \nu 2'(xi)$ for the new variables $x1, ..., xn$, and t' evaluates to the same value as t, *i.e.* $\nu 1(t') = v$ as expected.

In those cases where r is a one-point sort, the carrier $|A|_r$ must be a singleton and therefore reachable from whatever set of generators we chose for r

provided at least one term of sort r can be generated, and this is guaranteed since $\ll G1, F1, S1 \gg$ is $\emptyset$-free.

Completeness proof Assuming $\ll G2, F2, S2 \gg \not\forall \ll G1, F1, S1 \gg$, we consider those cases in which the inference can fail:

1. If $G1 \not\subseteq G2$ then we can choose a sort $r \in G1$ which is not generated in $\ll G2, F2, S2 \gg$. Now, consider a final algebra A over Σ extended in sort r to a bigger carrier, *e.g.* $|A|_r = \{\bot, \star\}$ while for all $s' \neq r$, $|A|_{s'} = \{\bot\}$ and the functions taking a parameter of sort r are extended in the only possible way, *i.e.* yielding always $\bot$. By construction A still satisfies $\ll G2, F2, S2 \gg$ but it fails to satisfy $\ll G1, F1, S1 \gg$ since $\star$ cannot be generated in A; hence $\ll G2, F2, S2 \gg \not\models_\Sigma \ll G1, F1, S1 \gg$.

2. If for a sort $r \in G1$ it happens that $F1_r \not\supseteq F2_r$ and r is not a one-point sort w.r.t. $\ll G2, F2, S2 \gg$, we can choose a generator $op : \overline{s} \to r \in F2_r$ which is not a generator in $F1_r$.

   Now, we consider the Σ-algebra A with carriers $|A|_s = \{\bot, \star\}$ for all sorts except for one-point sorts w.r.t. $\ll G2, F2, S2 \gg$ which take carriers $|A|_{s'} = \{\bot\}$, and constants and functions defined as follows:

   - Functions (or constants) $f : \overline{s} \to r$ which are not in $F2_r$ always yield $\bot$, *i.e.* if $f \notin F2_r$ then $f(\overline{x}) = \bot$.
   - Functions $f : s1, ..., sn \to r$ which are the only generator of a sort, $F2_r = \{f\}$, are the identity on one of the parameters of the sorts with biggest carrier, *i.e.* $f(x1, ..., xn) = xi$ such that $|A|_{si} \supseteq |A|_{sj}$ for $j = 1, ..., n$.
   - For functions (or constants) which are generators of a sort with several generators, $F2_r = \{f1, f2, ...\}$, we pick one of them to yield always $\star$

while the rest of the generators always yield $\perp$. If $op \in F2_r$ then op is the generator which is chosen to yield $\star$.

By construction of A and since $\ll G2, F2, S2 \gg$ is $\emptyset$-free, $\perp$ and $\star$ are reachable in all sorts which are not one-point sorts w.r.t. $\ll G2, F2, S2 \gg$, hence $A \models \ll G2, F2, S2 \gg$. However, $A \not\models \ll G1, F1, S1 \gg$ because $\star \in |A|_r$ is not reached unless op is used.

□

As argued in [Far 89] many entailments preserve the generators from antecedent to consequent and simple matching of identical constraints is very frequent. Here, this result generalizes previous work on matching constraints to the point that it achieves completeness for single $\emptyset$-free constraints of the form $\ll G, F, S \gg$ in the absence of other kinds of sentences.

The next step towards a true logic of constraints is to extend such an inference system to deal with several simultaneous $\emptyset$-free constraints. But this brings some new problems.

A reachability constraint which constrains several sorts simultaneously can be a stronger requirement than a set of constraints containing a single reachability constraint for each sort. Consider for example a signature $\Sigma = \langle S', \Omega' \rangle$ with

$$S' = \{s1, s2\} \qquad \Omega' = \{f : s1 \rightarrow s2$$
$$g : s2 \rightarrow s1$$
$$c1 : s1$$
$$c2 : s2\}$$

and a Σ-algebra A with $|A|_{s1} = \{0, 1, e\}$ and $|A|_{s2} = \{0', 1', e'\}$ such that $c1 = 0$, $c2 = 0'$, $f(0) = f(1) = 1'$, $g(0') = g(1') = 1$, $f(e) = e'$ and $g(e') = e$. Values e

and e' are mutually generated, hence algebra A satisfies constraints,

$$A \models_\Sigma \ll\{s1\},\{c1,g\},\{s2\}\gg \quad \text{and} \quad A \models_\Sigma \ll\{s2\},\{c2,f\},\{s1\}\gg$$

but A does not satisfy a constraint on both sorts simultaneously

$$A \not\models_\Sigma \ll\{s1,s2\},\{c1,c2,f,g\},\emptyset\gg$$

since neither e nor e' are ever generated using $c1, c2, f$ and g.

This situation arises because sorts $s1$ and $s2$ are *mutually dependent* sorts according to the following definition:

Definition 6.3.6 *Two sorts $s1$ and $s2$ are* **mutually dependent** *in a signature $\langle S,\Omega\rangle$ iff $s1 \rightarrow^* s2$ and $s2 \rightarrow^* s1$, where $\rightarrow^*$ is the transitive closure of the relation $\rightarrow$ defined between sorts in S as follows: $s' \rightarrow s$ iff there exists a function name $f : \omega \rightarrow s$ in Ω and s' is included in the sequence ω.*

On the other hand, joining sets of generators generally leads to weaker constraints. Consider for example,

$$A \models_\Sigma \ll\{s1\},\{c1,f,g\},\{s2\}\gg$$

but A does not satisfy the two constraints,

$$A \not\models_\Sigma \ll\{s1\},\{c1\},\{s2\}\gg \quad \text{and} \quad A \not\models_\Sigma \ll\{s1\},\{f,g\},\{s2\}\gg$$

Taking in consideration the contrary effect of joining sets of sorts and joining sets of generators, we define a correct join constraint in such a way that generators cannot be joined and joining sorts has no effect.

Definition 6.3.7 *The* **join constraint** *of two Σ-constraints $\ll G1,F1,S1\gg$ and $\ll G2,F2,S2\gg$, written $\ll G1,F1,S1\gg\bowtie\ll G2,F2,S2\gg$, is another Σ-constraint $\ll G\bowtie,F\bowtie,S\bowtie\gg$ such that*

$$G\bowtie = G1 \cup G2$$
$$F\bowtie = (\bigcup_{i\in G1} F1_i) \cup (\bigcup_{j\in G2} F2_j)$$
$$S\bowtie = (S1 \cup S2) \setminus (G1 \cup G2)$$

The result is called a **correct join constraint** *if* $G1$ *and* $G2$ *are disjoint and every pair of mutually generated sorts* $r1$ *and* $r2$ *in* $\langle G\bowtie \cup S\bowtie, F\bowtie\rangle$ *is also mutually generated in either* $\langle G1 \cup S1, F1\rangle$ *or in* $\langle G2 \cup S2, F2\rangle$.

Correct join constraints relate to their component constraints according to the following results:

Proposition 6.3.8 *For every correct join constraint* $\ll G1, F1, S1 \gg\bowtie\ll G2, F2, S2 \gg$ *over* Σ,

$$\ll G1, F1, S1 \gg\bowtie\ll G2, F2, S2 \gg \models_\Sigma \{\ll G1, F1, S1 \gg, \ll G2, F2, S2 \gg\}$$

Proof By construction of the join constraint, all generated sorts of $\ll G1, F1, S1 \gg$ are inherited, $G1 \subseteq G\bowtie$. Moreover, since $G1$ and $G2$ are disjoint the set of generators $F\bowtie_r$ for a sort in $r \in G1$ is as $F1_r$ and, in general $F\bowtie_r \subseteq F1_r$ for all $r \in G1$. Hence,

$$\ll G1, F1, S1 \gg\bowtie\ll G2, F2, S2 \gg \vdash \ll G1, F1, S1 \gg$$

By soundness of $\vdash$, the first constraint follows from the join, and similarly for the second constraint.

□

Correct join constraints can only bring together generated sorts which are not mutually generated. In the example shown above, joining constraints

$$\ll\{s1\}, \{c1, g\}, \{s2\}\gg \quad \text{and} \quad \ll\{s2\}, \{c2, f\}, \{s1\}\gg$$

is not correct since f generates $s2$ from $s1$ and g generates $s1$ from $s2$.

Correct join constraints are always weaker than the union of the original constraints.

Proposition 6.3.9 *For all correct join constraints* $\ll G1, F1, S1 \gg \bowtie \ll G2, F2, S2 \gg$ *over* Σ,

$$\{\ll G1, F1, S1 \gg, \ll G2, F2, S2 \gg\} \models_\Sigma \ll G1, F1, S1 \gg \bowtie \ll G2, F2, S2 \gg$$

Proof Let $\iota 1, \iota 2$ and ι be the inclusions $\langle S1 \cup G1, F1\rangle \hookrightarrow \Sigma$, $\langle S2 \cup G2, F2\rangle \hookrightarrow \Sigma$ and $\langle S\bowtie \cup G\bowtie, F\bowtie\rangle \hookrightarrow \Sigma$, A a model of $\{\ll G1, F1, S1 \gg, \ll G2, F2, S2 \gg\}$ and v a value in a generated carrier $|A|_r$ for $r \in G1$ (the following argument is symmetric for $r \in G2$).

By definition of satisfaction, since $A \models_\Sigma \ll G1, F1, S1 \gg$, there exists a term $t \in |T_{\langle S1 \cup G1, F1\rangle}(X_{S1})|_r$ for an $S1$-indexed set of variables $S1$ and a valuation $\nu : T_{\langle S1 \cup G1, F1\rangle}(X_{S1}) \to A|_{\iota 1}$ such that $\nu(t) = v$.

Let $x1, ..., xn \in X_{S1}$ be the free variables of t from which $x1, ..., xk$ belong to sorts in $G2$. Since $A \models_\Sigma \ll G2, F2, S2 \gg$ the values denoted by variables $x1, ..., xk$ can also be generated. Let vi be the value of $\nu(xi)$ for $i = 1, ..., k$; there exists a term $ti \in |T_{\langle S2 \cup G2, F2\rangle}(X_{S2})|$ for each vi for an $S2$-indexed set of variables and a valuation $\nu' : T_{\langle S2 \cup G2, F2\rangle}(X_{S2}) \to A|_{\iota 2}$ such that $\nu'(ti) = vi$.

Combining X_{S1} and X_{S2} into a single $((S1 \cup S2) \setminus G2)$-indexed set of variables X and combining valuations ν and ν' into one valuation $\nu'' : T_{\langle S\bowtie \cup G\bowtie, F\bowtie\rangle}(X) \to A|_\iota$, we have that

$$\nu''(t[t1/x1, ..., tk/xk]) = v$$

Considering the free variables in $t1, ..., tk$ belonging to sorts in $G1$ we can iterate the process again until only variables of sorts in $(S1 \cup S2) \setminus (G1 \cup G2)$ are left.

By definition of correct join constraint, we are guaranteed not to fall into a cycle. No mutual dependency of the sorts generated in one constraint w.r.t. those

sorts generated in the other constraint guarantees a well-founded ordering for induction.

In the base case the free variables of t are assumed to be of sorts in $S\bowtie$, and in the induction step values $v1, ..., vk$ are generated, by induction hypothesis, by terms $t1', ..., tk'$ over an $(S\bowtie)$-indexed set of variables.

Summing up, there exists always a term $t' \in |T_{(S\bowtie \cup G\bowtie, F\bowtie)}(X_{S\bowtie})|_r$ and an $(S\bowtie)$-indexed set of variables $X_{S\bowtie}$, and valuation ν'' such that $\nu''(t') = v$.

Since all generated sorts in the join constraint are also generated in $\ll G1, F1, S1 \gg$ or $\ll G2, F2, S2 \gg$, it follows that $A \models \ll G1, F1, S1 \gg \bowtie \ll G2, F2, S2 \gg$, as expected.

□

In practice, correctness of a join constraint is commonly satisfied. Consider a specification language based on $ASL(\mathbf{EQ})$ where specifications $M_{\langle G1,F1,S1\rangle} ... M_{\langle Gn,Fn,Sn\rangle} SP$ are guaranteed to satisfy that:

1. Reachability constraints do not constrain already generated sorts, *i.e.* Gj in $M_{\langle Gj,Fj,Sj\rangle}$ is disjoint from $\bigcup_{j<i\leq n} Gi$.

2. Generating sorts in one constraint are not constrained in successive constraints, *i.e.* Sj in $M_{\langle Gj,Fj,Sj\rangle}$ is disjoint from $\bigcup_{1\leq i<j} Gi$.

From this and the fact the generated and generating sorts are disjoint in each constraint, it can be concluded that all join constraints $\ll Gj, Fj, Sj \gg \bowtie \ll Gk, Fk, Sk \gg$ for $k < j$ are correct.

It is normal practice in specification design to use constraints arranged in such sequences. More general definitions of reachability constraint, such as the one we use, are chosen on theoretical grounds. In our case, such a general $\ll G, F, S \gg$

satisfies the satisfaction condition and algebraic laws such as commutativity analogously to arbitrary sentences. Although we treat this more general case, we can well believe that correctness of join constraints is very common in practice.

In the following we give an inference system for multiple constraints which is complete when constraints of the form $\ll G, F, S \gg$ are arranged in sequences as above.

Definition 6.3.10 *A consequence relation $\Vdash$ between finite sets of constraints is defined by the following set of rules:*

$$\frac{}{C \Vdash \emptyset} \qquad \frac{}{\emptyset \Vdash \{\ll \emptyset, F, S \gg\}} \qquad \frac{\ll G2, F2, S2 \gg \vdash \ll G1, F1, S1 \gg}{\{\ll G2, F2, S2 \gg\} \Vdash \{\ll G1, F1, S1 \gg\}}$$

$$\frac{\ll G1, F1, S1 \gg \bowtie \ll G2, F2, S2 \gg \text{ is correct}}{\{\ll G1, F1, S1 \gg, \ll G2, F2, S2 \gg\} \Vdash \{\ll G1, F1, S1 \gg \bowtie \ll G2, F2, S2 \gg\}}$$

$$\frac{C21 \Vdash C1}{C21 \cup C22 \Vdash C1} \qquad \frac{C2 \Vdash C11 \quad C2 \Vdash C12}{C2 \Vdash C11 \cup C12} \qquad \frac{C2 \Vdash C3 \quad C3 \Vdash C1}{C2 \Vdash C1}$$

Theorem 6.3.11 (Soundness) *For all sets of $\emptyset$-free reachability constraints $C1$ and $C2$,*

$$C2 \Vdash C1 \implies C2 \models C1$$

Proof We prove soundness of a proof in $\Vdash$ by induction on the rules used.

Case $\frac{}{\emptyset \vdash \{\ll \emptyset, F, S \gg\}}$

By definition of satisfaction all models of the appropriate signature satisfy a constraint with no generated sorts, hence $\emptyset \models \{\ll \emptyset, F, S \gg\}$.

Case $\frac{\ll G2, F2, S2 \gg \vdash \ll G1, F1, S1 \gg}{\{\ll G2, F2, S2 \gg\} \vdash \{\ll G1, F1, S1 \gg\}}$

By soundness of $\vdash$ (Theorem 6.3.5), it must hold that $\ll G2, F2, S2 \gg \models \ll G1, F1, S1 \gg$; hence

$$\{\ll G2, F2, S2 \gg\} \models \{\ll G1, F1, S1 \gg\}$$

Case $\frac{\ll G1,F1,S1\gg \bowtie \ll G2,F2,S2\gg \text{ is correct}}{\{\ll G1,F1,S1\gg,\ll G2,F2,S2\gg\} \vdash \{\ll G1,F1,S1\gg \bowtie \ll G2,F2,S2\gg\}}$

Immediate by proposition 6.3.9.

Cases $\frac{}{C \vdash \emptyset}$ $\frac{C21 \vdash C1}{C21 \cup C22 \vdash C1}$ $\frac{C2 \vdash C11 \quad C2 \vdash C12}{C2 \vdash C11 \cup C12}$ $\frac{C2 \vdash C3 \quad C3 \vdash C1}{C2 \vdash C1}$

These rules are sound by the definition of satisfaction between sets of sentences in an institution.

□

This inference system can be proven complete w.r.t. entailment between sets of constraints having correct joins as discussed before. In order to do that, it is enough to replace each set of constraints by the join of all its constraints. By propositions 6.3.8 and 6.3.9 such a reduction is sound, so that completeness of $\Vdash$ follows directly from completeness of $\vdash$.

Theorem 6.3.12 (Completeness) *For all sets of $\emptyset$-free reachability constraints $C1$ and $C2$ such that all joins among constraints in $C1$ and in $C2$ are correct,*

$$C2 \models C1 \implies C2 \Vdash C1$$

Proof For all $C2 \models C1$ we can prove by structural induction on $C1$ that a formal proof can be derived in $\Vdash$.

Case $C1 = \emptyset$

$C2 \models C1$ holds for all $C2$ since $C2 \Vdash \emptyset$ is immediately derivable by the first inference rule.

Case $C1 = \{\ll G1, F1, S1 \gg\}$

By structural induction on $C2$ we prove that $C2 \models C1$ implies $C2 \Vdash C1$ for any singleton $C1$:

- If $C2 = \emptyset$ and $C2 \models \ll G1, F1, S1 \gg$, it follows that $\ll G1, F1, S1 \gg$ cannot impose any restriction at all, so $G1$ must be empty. In that case, $\emptyset \Vdash \{\ll \emptyset, F1, S1 \gg\}$ is immediately derivable by the second inference rule.

- If $C2 = \{\ll G2, F2, S2 \gg\}$ then by completeness of $\vdash$ for single constraints (Theorem 6.3.5) and the third inference rule, it follows that

$$\{\ll G2, F2, S2 \gg\} \Vdash \{\ll G1, F1, S1 \gg\}$$

  is derivable.

- If $C2 = C21 \cup C22$ we consider the join $\ll G21, F21, S21 \gg$ of all constraints in $C21$, and the join $\ll G22, F22, S22 \gg$ of all constraints in C22. By proposition 6.3.9 $C21 \models \ll G21, F21, S21 \gg$ and, by induction hypothesis, $C21 \Vdash \{\ll G21, F21, S21 \gg\}$. Similarly $C22 \Vdash \{\ll G22, F22, S22 \gg\}$. Combining these derivations and using the fourth rule we can derive that

$$C21 \cup C22 \Vdash \{\ll G21, F21, S21 \gg \bowtie \ll G22, F22, S22 \gg\}$$

  At the same time, by proposition 6.3.8, we have that $\ll G21, F21, S21 \gg \models C21$ and $\ll G22, F22, S22 \gg \models C22$, hence

$$\{\ll G21, F21, S21 \gg, \ll G22, F22, S22 \gg\} \models \ll G1, F1, S1 \gg$$

  Applying proposition 6.3.8 again we conclude that

$$\ll G21, F21, S21 \gg \bowtie \ll G22, F22, S22 \gg \models \ll G1, F1, S1 \gg$$

  Finally, by the previous case,

$$\{\ll G21, F21, S21 \gg \bowtie \ll G22, F22, S22 \gg\} \Vdash \{\ll G1, F1, S1 \gg\}$$

  is derivable. Putting this derivation together with that above using the last rule we conclude that $C12 \cup C22 \Vdash \{\ll G1, F1, S1 \gg\}$.

Case $C1 = C11 \cup C12$

Whenever it is the case that $C2 \models C11 \cup C12$, it is guaranteed that all models of $C2$ are also models of $C11$ and $C12$; *i.e.*

$$C2 \models C11 \qquad C2 \models C12$$

Therefore, by induction hypothesis it holds that $C2 \Vdash C11$ and $C2 \Vdash C11$, so that applying the sixth rule $C2 \Vdash C11 \cup C12$ can be derived.

□

Similarly to the inference system proposed for single constraints, $\Vdash$ only takes account of the simplest implications and it is complete for constraints of the form $\ll G, F, S \gg$. Unfortunately, this requires constraints in $C2$ to constraint disjoint sets of sorts. Consider for example a more difficult case such as:

$$S' = \{s\} \qquad \Omega' = \{a : s$$
$$b : s$$
$$c : s\}$$

and $\langle S', \Omega' \rangle$-constraints

$$c1 = \ll \{s\}, \{a, b\}, \emptyset \gg \qquad c2 = \ll \{s\}, \{a, c\}, \emptyset \gg$$

$$c3 = \ll \{s\}, \{b, c\}, \emptyset \gg \qquad c = \ll \{s\}, \{a\}, \emptyset \gg$$

In this case, it holds that $\{c1, c2, c3\} \models \{c\}$ since among all $\langle S', \Omega' \rangle$-algebras only the trivial (one-point) algebras satisfy the three constraints $c1, c2$ and $c3$, and all these models satisfy c, although $\{c1, c2, c3\} \not\Vdash \{c\}$.

From a more technical standpoint, examples such as this are quite peculiar since they deal with "finite" constraints; *e.g.* $c1$ is equivalent to a sentence $\forall x : S.\ (x = a \vee x = b)$, and by translating all three constraints to this form we can

infer using FOLEQ that $\forall x : S.\ a = x$. If we deal with constraints which are only equivalent to proper infinitary sentences, then a finite consequence such as $\forall x : S.\ a = x$ could never hold because of some fundamental results of model theory (see theorem 6 in [KK 67]).

Nonetheless, as argued before, multiple constraints on one sort are not used in practice; only a generalization of constraints in order to make them comparable to normal sentences has introduced such a possibility. Therefore, we should consider $\Vdash$ as complete for a specification language whose only sentences are reachability constraints of the form $\ll G, F, S \gg$. In terms of specifications in ASL, $\Vdash$ provides a complete inference system in this sense for $M \models M$.

6.3.2 Reachability constraints and sentences

In section 6.3.1, we dealt with an algebraic institution whose only sentences are reachability constraints. However, reachability constraints are never used on their own but, most probably, combined with equations, conditional equations or first order sentences with equality.

From a proof-theoretical standpoint, we have two different kinds of sentences: reachability constraints equipped with a complete inference system $\Vdash$ as defined above and ordinary sentences with another inference system $\vdash^{\mathbf{I}}$, in the case of equational logic a complete inference system $\vdash^{\mathbf{EQ}}$.

By taking the union of these two kinds of sentences, we can easily define a new institution with constraints over $\mathbf{I}$ as in theorem 6.2.1, but there is very little we can directly say about inference for such mixed sets of sentences. Some general work on the combination of different logics can be found in [HST 89a], but not much can be said in general.

In the following we refer to **EQ** with reachability constraints, but also to other institutions with signatures, models, terms and reachability constraints as in **EQ**, *e.g.* **ALG**.

In terms of specifications in ASL, we are addressing the problem of $MA \models MA$.

Independent strategy

The results obtained for a logic of constraints give raise to an immediate strategy for proving entailments such as $A_{C2}A_{\Phi 2}\epsilon_\Sigma \models A_{C1}A_{\Phi 1}\epsilon_\Sigma$ in **EQ**, namely

$$\frac{\Phi 2 \vdash^{\mathbf{EQ}} \Phi 1 \qquad C2 \Vdash C1}{A_{C2}A_{\Phi 2}\epsilon_\Sigma \models_\Sigma A_{C1}A_{\Phi 1}\epsilon_\Sigma}$$

where $\Phi 1$ and $\Phi 2$ are sets of Σ-sentences and $C1$ and $C2$ are sets of Σ-constraints.

This is called the independent strategy because it corresponds to taking the union of the inference rules of the different logics in order to prove entailment between heterogeneous sets of sentences. However, given the two complete inference system $\vdash^{\mathbf{EQ}}$ and $\Vdash$ their union is incomplete w.r.t. the union of their sentences. Cases where the independent strategy fails are easy to find, *e.g.*

$M_{(G1,F1,S1)}A_{\Phi 1}\epsilon_\Sigma$ =

```
Reachable on {nat} by {0, suc}
sorts nat
operations
   0:  nat
   suc:  nat->nat
   _+_ :  nat, nat->nat
axioms
   ∀ n:nat. 0+n=n
   ∀ n,m:nat. (suc m)+n=suc(m+n)
   ∀ n,m:nat. m+n=n+m
end
```

$M_{\langle G2,F2,S2\rangle}A_{\Phi 2\epsilon\Sigma}$= Reachable on {nat} by {0, suc, +}
sorts nat
operations
0: nat
suc: nat->nat
_+_ : nat, nat->nat
axioms
∀ n:nat. 0+n=n
∀ n,m:nat. (suc m)+n=suc(m+n)
end

where it is the case that $\Phi 2 \not\vdash^{\mathbf{EQ}} \Phi 1$ and $\ll G2, F2, S2 \gg \not\vdash \ll G1, F1, S1 \gg$ but the refinement is correct, $M_{\langle G2,F2,S2\rangle}A_{\Phi 2\epsilon\Sigma} \models M_{\langle G1,F1,S1\rangle}A_{\Phi 1\epsilon\Sigma}$.

In order to improve this situation, specific rules must be given to infer sentences from sentences and constraints, $MA \models A$, and to infer constraints from sentences and constraints, $MA \models M$. The first problem has already been tackled using induction principles in theorem proving (section 3.3.2), whereas the second problem remains to be studied. The results are expressed in the form of an inference system for constraints parameterized by a set of sentences, *i.e.* $\ll G2, F2, S2 \gg \vdash_\Phi \ll G1, F1, S1 \gg$ means that $\ll G1, F1, S1 \gg$ can be inferred from Φ and $\ll G2, F2, S2 \gg$.

Characterization of $\vdash_\Phi$

As we saw in section 6.3.1, $\ll G2, F2, S2 \gg \vdash \ll G1, F1, S1 \gg$ checks whether all generated sorts in $\ll G1, F1, S1 \gg$ were already generated in $\ll G2, F2, S2 \gg$ by the same or fewer generators. Now, $\vdash_\Phi$ must take into account that some generators in $\ll G2, F2, S2 \gg$ may be excluded during the check because they are *redundant* w.r.t. the rest of the generators and Φ.

Definition 6.3.13 *Given a Σ-reachability constraint $\ll G, F, S \gg$ and a generated sort $r \in G$, a generator $op \in F_r$ is* **redundant** *in $\ll G, F, S \gg$ w.r.t. a set of Σ-sentences Φ if for all Σ-algebras A such that $A \models_\Sigma \Phi$*

$$A \models_\Sigma \ll G, F, S \gg \quad \text{iff} \quad A \models_\Sigma \ll G, F \setminus \{op\}, S \gg$$

In general, a pair $\langle S'', F'' \rangle$ is **redundant** *in a Σ-constraint $\ll G, F, S \gg$ w.r.t. Φ if $S'' \subseteq S$, $F'' \subseteq F$ and for all algebras such that $A \models_\Sigma \Phi$*

$$A \models_\Sigma \ll G, F, S \gg \quad \text{iff} \quad A \models_\Sigma \ll G, F \setminus F'', S \setminus S'' \gg$$

Considering the definition of satisfaction of a reachability constraint, redundancy of a pair $\langle S'', F'' \rangle$ in a Σ-constraint $\ll G, F, S \gg$ can be rephrased as follows:

Let ι and ι'' be inclusions

$$\langle G \cup (S \setminus S''), F \setminus F'' \rangle \overset{\iota''}{\hookrightarrow} \langle G \cup S, F \rangle \overset{\iota}{\hookrightarrow} \Sigma$$

and A a Σ-algebra such that $A \models_\Sigma \Phi$ and $A \models_\Sigma \ll G, F, S \gg$, if all values of a sort $r \in G$ are denoted by a term $t \in |T_{\langle G \cup S, F \rangle}(X_S)|_r$ for an S-indexed set of variables X_S and under a valuation $\nu : T_{\langle G \cup S, F \rangle}(X_S) \to A|_\iota$, then all values can also be produced by a term $t' \in |T_{\langle G \cup (S \setminus S''), F \setminus F'' \rangle}(X_{S \setminus S''})|_r$ for an $(S \setminus S'')$-indexed set of variables $X_{S \setminus S''}$ and under a valuation

$$\nu' : T_{\langle G \cup (S \setminus S''), F \setminus F'' \rangle}(X_{S \setminus S''}) \to A|_{\iota'';\iota}$$

This definition of redundancy can be considerably simplified if algebras with empty carriers are not taken into account.

Definition 6.3.14 *Given a set of sorts S, the S-indexed set of variables $\mathcal{X}_S$ is defined as follows: for all sorts $s \in S$, $\mathcal{X}_s = \{x_i \mid i \in \mathcal{N}\}$ where $\mathcal{N}$ is the set of the natural numbers.*

Lemma 6.3.15 *Given a signature inclusion* $\iota : \langle G \cup S, F\rangle \hookrightarrow \Sigma$, *a* Σ*-constraint* $\ll G, F, S \gg$ *and a model* $A \models_\Sigma \ll G, F, S \gg$ *without empty carriers, then for every value* $v \in |A|_r$ *for* $r \in G$ *there exists* $t_v \in |T_{\langle G \cup S, F\rangle}(\mathcal{X}_S)|_r$ *and a valuation* $\nu : T_{\langle G \cup S, F\rangle}(\mathcal{X}_S) \to A|_\iota$ *such that* $\nu(t_v) = v$.

Proof By definition of satisfaction of a Σ-constraint $\ll G, F, S \gg$, for every model $A \models_\Sigma \ll G, F, S \gg$ there exists an S-indexed set of variables X_S such that for each value $v \in |A|_r$ for $r \in G$ there exists a valuation $\nu : T_{\langle G \cup S, F\rangle}(X_S) \to A|_\iota$ and a term $t_v \in |T_{\langle G \cup S, F\rangle}(X_S)|$ such that $\nu(t_v) = v$. In other words, each value v is generated from a finite set of values of each sort $s \in S$

$$V_s = \{\nu(x) \mid x \text{ is a variable in } t_v \text{ of sort } s\}$$

We can order the values in V_s so that $V_s = \{v_1, ..., v_n\}$. Then, a valuation $\nu' : T_{\langle G \cup S, F\rangle}(\mathcal{X}_S)$ can be defined for each sort s in S as follows:

$$\nu'(x_i) = v_i \quad \text{for all } i = 1, ..., n$$
$$\nu'(x_i) = \bot \quad \text{for all } i > n$$

where $\bot$ is an arbitrary element in $|A|_s$. Since A has no empty carriers, ν' is well defined and, by definition, there exists a term $t' \in |T_{\langle G \cup S, F\rangle}(\mathcal{X}_S)|_r$ such that $\nu'(t') = v$.

□

Assumption *In the following we shall assume all models to be algebras without empty carriers. This does not produce any problem since reachability constraints are already assumed to be* $\emptyset$*-free.*

In order to take care of redundancy in $\vdash_\Phi$ and to produce, at the same time, a sensible inference system, we have to avoid referring to models and particular valuations. In the following two sufficient criteria for redundancy are presented.

First proof criterion

Proposition 6.3.16 (First proof criterion) *Given a set of Σ-sentences Φ, a Σ-constraint $\ll G, F, S \gg$ and a pair $\langle S'', F'' \rangle$ such that $S'' \subseteq S$, $F'' \subseteq F$ and*

$$\forall\, t \in |T_{\langle G \cup S, F \rangle}(\mathcal{X}_S)|_r.\ \exists\, t' \in |T_{\langle G \cup (S \setminus S''), F \setminus F'' \rangle}(\mathcal{X}_{S \setminus S''})|_r.\ \Phi \models_\Sigma \forall \mathcal{X}_S.\ t = t'$$

for every sort $r \in G$ then $\langle S'', F'' \rangle$ is redundant in $\ll G, F, S \gg$ w.r.t. Φ.

Proof Let ι and ι'' be the inclusions

$$\langle G \cup (S \setminus S''), F \setminus F'' \rangle \overset{\iota''}{\hookrightarrow} \langle G \cup S, F \rangle \overset{\iota}{\hookrightarrow} \Sigma$$

and A a Σ-algebra without empty carriers satisfying Φ and $\ll G, F, S \gg$. According to lemma 6.3.15, every value $v \in |A|_r$ for $r \in G$ can be produced by a term $t_v \in |T_{\langle G \cup S, F \rangle}(\mathcal{X}_S)|_r$ using a valuation $\nu : T_{\langle G \cup S, F \rangle}(\mathcal{X}_S) \to A|_\iota$ such that $\nu(t) = v$. By assumption, there exists also a term

$$t'_v \in |T_{\langle G \cup (S \setminus S''), F \setminus F'' \rangle}(\mathcal{X}_{S \setminus S''})|_r \subseteq |T_{\langle G \cup S, F \rangle}(\mathcal{X}_S)|_r$$

such that $\Phi \models_\Sigma \forall X.\ t_v = t'_v$, hence $\nu(t'_v) = v$.

Taking the ι''-reduct of the valuation ν, we obtain a homomorphism over $\langle G \cup (S \setminus S''), F \setminus F'' \rangle$

$$\nu|_{\iota''} : T_{\langle G \cup S, F \rangle}(\mathcal{X}_S)|_{\iota''} \to A|_{\iota''; \iota}$$

which by definition can be restricted to terms in $|T_{\langle G \cup (S \setminus S''), F \setminus F'' \rangle}(\mathcal{X}_{S \setminus S''})|$

$$\nu' : T_{\langle G \cup (S \setminus S''), F \setminus F'' \rangle}(\mathcal{X}_{S \setminus S''}) \to A|_{\iota''; \iota}$$

so that ν' is a valuation with $\nu'(t'_v) = v$ thus $A \models_\Sigma \ll G, F \setminus F'', S \setminus S'' \gg$.

Entailment in the other direction follows immediately from

$$\ll G, F \setminus F'', S \setminus S'' \gg \vdash \ll G, F, S \gg$$

and soundness of $\vdash$ w.r.t. constraint entailment (Theorem 6.3.5).

□

This fact together with an inference system $\vdash^{\mathbf{ALG}}$ provide a sufficient criterion for checking redundancy of some sorts and functions in a reachability constraint w.r.t. a presentation. For example, consider

```
Σ = sorts nat
      operations
         0:  nat
         suc:  nat -> nat
         _+_ :  nat, nat -> nat

Φ = { ∀ n:nat. 0+n=n,
      ∀ n,m:nat. (suc m)+n=suc(m+n)}
```

It is not difficult to check that "+" is redundant in $\ll\{\mathtt{nat}\},\{\mathtt{0},\mathtt{suc},+\},\emptyset\gg$ w.r.t. Φ using the above criterion. In this case, it is enough to prove by structural induction on terms of sort nat that every term containing "+" is equal to a term without "+".

In fact, the above criterion characterizes redundancy in **EQ** when the set of generating carriers S in a constraint $\ll G, F, S\gg$ is empty.

Proposition 6.3.17 *Given a signature $\langle G, F\rangle$, a set of $\langle G, F\rangle$-equations Φ and a constraint $\ll G, F, \emptyset\gg$, if a pair $\langle\emptyset, F''\rangle$ is redundant in $\ll G, F, \emptyset\gg$ w.r.t. Φ, then*

$$\forall\, t \in |T_{\langle G,F\rangle}|_r.\ \exists\, t' \in |T_{\langle G,F\setminus F''\rangle}|_r.\ \Phi \models_{\langle G,F\rangle} \forall\emptyset.\ t = t'$$

for all sorts $r \in G$.

Proof Let ι be the inclusion

$$\langle G, F \setminus F''\rangle \overset{\iota}{\hookrightarrow} \langle G, F\rangle$$

and I the quotient of $T_{\langle G,F\rangle}$ modulo the least congruence generated by the equations in Φ. By definition of least congruence, we have that

$$\nu_I(t) = \nu_I(t') \quad \textit{iff} \quad \Phi \models_{\langle G,F\rangle} \forall \emptyset.\ t = t'$$

where $\nu_I : T_{\langle G,F\rangle} \to I$ maps each term to its quotient class of terms in the algebra I. In other words, the algebra I under valuation ν_I characterizes those equations satisfied by all the models of Φ.

By definition of I, it is clear that $I \models_{\langle G,F\rangle} \Phi$ and also that $I \models_{\langle G,F\rangle} \ll \dot{G}, F, \emptyset \gg$, and since $\langle \emptyset, F''\rangle$ is redundant in $\ll G, F, \emptyset \gg$ w.r.t. Φ, $I \models_{\langle G,F\rangle} \ll G, F \setminus F'', \emptyset \gg$ as well.

Then, every term $t \in |T_{\langle G,F\rangle}|_r$ for $r \in G$ denotes a value $\nu_I(t) \in |I|_r$ which can be generated by a term $t' \in |T_{\langle G,F\setminus F''\rangle}|_r$ under a valuation which can only be ν_I, *i.e.* $\nu_I(t') = \nu_I(t)$, hence $\Phi \models_{\langle G,F\rangle} \forall \emptyset.\ t = t'$.

□

Redundancy and sufficient completeness

Sufficient completeness of a free extension $F^{eq}_{\sigma}SP$, as defined on page 77, means that *no new values are added to old carriers.* In particular, having a generated sort $r \in G$ with a redundant generator $op \in F_r$ in $\ll G, F, \emptyset \gg$ w.r.t. a set of equations eq is equivalent to sufficient completeness of $F^{eq}_{\iota}(F_{\sigma}\epsilon_{\langle\emptyset,\emptyset\rangle})$ with

$$\langle \emptyset, \emptyset\rangle \overset{\sigma}{\hookrightarrow} \langle G, F \setminus \{op\}\rangle \overset{\iota}{\hookrightarrow} \langle G, F\rangle$$

In fact sufficient completeness as originally defined [GH 78] looks very similar to our criterion for checking redundancy (when all sorts are generated), whereas our definition in chapter 3 is analogous to the model theoretic definition in [Gau 78, Gan 83, EM 85].

In the literature, both definitions refer to initial semantics, and while the model theoretic definition requires that: *no new values are added to old carriers*, the proof theoretic definition requires that: *every new term must be provably equal to an old one (using equational logic)*. For equational specifications with initial semantics both definitions are equivalent but for loose semantics their trivial generalizations are not equivalent any more [Ber 87].

Second proof criterion

Unfortunately, if sentences are not equations, there are redundancies that we cannot prove using the above criterion, even if the set of generating carriers S is empty. For example:

Σ' = sorts nat
operations
0: nat
suc: nat -> nat
rand: nat -> nat

Φ' = { $\forall$ n:nat. 0+n=n,
$\forall$ n,m:nat. (suc m)+n=suc(m+n),
$\forall$ n:nat. rand(n)=n $\vee$ rand(n)=suc(n)}

In this case the function rand is clearly redundant in $\ll\{\mathtt{nat}\}, \{\mathtt{0}, \mathtt{suc}, \mathtt{rand}\}, \emptyset \gg$ w.r.t. Φ' since rand cannot produce other natural numbers than those generated by 0 and suc. However, the proof criterion fails since given a term rand(0) $\in |T_{(\{\mathtt{nat}\},\{\mathtt{0},\mathtt{suc},\mathtt{rand}\})}|$ there is no term $t' \in |T_{(\{\mathtt{nat}\},\{\mathtt{0},\mathtt{suc}\})}|$ such that for all A satisfying Φ', $A \models \mathtt{rand(0)} = \mathtt{t'}$.

A difference between satisfying the above criterion and actual redundancy is that, for redundancy, given a term t, the term t' may be different for each model of Φ, while in the above criterion, t' must be same in all models of Φ. For this reason, the proof criterion is not a necessary condition for redundancy in **ALG**.

~~In tight specification languages where only one model of Φ is considered (up to isomorphism), for example the initial model, this problem disappears.~~

Cases such as the redundant function `rand` make us think that our proof criterion should be improved.

Proving redundancy as it stands in the definition involves a quantification over models which is not acceptable from a proof-theoretic point of view. We cannot expect to give a different term equivalent to `rand(0)` for each possible model of Φ'. However, if we want to prove that `rand` is redundant, we can show that all *possible results* of `rand(t)` are natural numbers generated by `0` and `suc`, assuming that term `t` is generated by `0` and `suc`.

~~Taking the institution **FOLEQ** and including set theory in all our specifications[2], we can express our second proof criterion as follows:~~

Proposition 6.3.18 (Second proof criterion) *Given a set of Σ-sentences Φ including set theory, a Σ-constraint $\ll G, F, S \gg$ and a pair $\langle S'', F'' \rangle$ such that $S'' \subseteq S$, $F'' \subseteq F$ and*

$$\forall\, t \in |T_{\langle G \cup S, F \rangle}(\mathcal{X}_S)|.\ \exists\, T' \subseteq |T_{\langle G \cup (S \setminus S''), F \setminus F'' \rangle}(\mathcal{X}_{S \setminus S''})|.\ \Phi \models_\Sigma \forall \mathcal{X}_S.\ t \in T'$$

then $\langle S'', F'' \rangle$ is redundant in $\ll G, F, S \gg$ w.r.t. Φ.

~~[2] Set theory is a first order theory *SetTh* which can be assumed to be included in all our specifications over *ASL*(**FOLEQ**).~~

notation "$t \in T'$" must be understood as an infinitary disjunction $= t' \mid t' \in T'\}$. This implies extending EQ to include disjunction een equations and, in particular, infinitary disjunctions.

Proof The proof proceeds analogously to the proof for the first criterion (proposition 6.3.16) taking into consideration that if a Σ-algebra without empty sorts A satisfies

$$A \models_\Sigma \forall \mathcal{X}_S.\ t \in T'$$

there exists a term

$$t'_v \in T' \subseteq |T_{\langle G \cup (S \setminus S''), F \setminus F'' \rangle}(\mathcal{X}_{S \setminus S''})|_r \subseteq |T_{\langle G \cup S, F \rangle}(\mathcal{X}_S)|_r$$

and a valuation $\nu : T_{\langle G \cup S, F \rangle}(\mathcal{X}_S) \to A|_\iota$ such that $\nu(t_v) = \nu(t'_v)$.

□

The second criterion is almost equivalent to redundancy when T' includes all terms t' equivalent to t in some model of Φ. At the same time, the second criterion is equivalent to the first criterion when T' is a singleton.

Redundancy of the function `rand` in the example presented above can be proven using the second proof criterion by taking $T' = \{t, \mathbf{suc}(t)\}$ for every term $t \in |T_{\langle \{nat\}, \{0, suc\} \rangle}|$.

EQ including infinitary disjunctions

This second criterion is defined for ~~**FOLEQ including set theory among their sentences**~~, unfortunately this prevents their inference systems from being complete using finite rules ~~due to incompleteness of set theory as proved by Gödel.~~ However, we consider this approach satisfactory just as we consider an induction schema satisfactory for proving inductive theorems.

According to this second criteria for redundancy, we can give an inference system $\vdash_\Phi$ between constraints parameterized by a set of sentences Φ which extends $\vdash$ as follows:

$$\frac{\ll G1, F1, S1 \gg \vdash \ll G2, F2, S2 \gg}{\ll G1, F1, S1 \gg \vdash_\Phi \ll G2, F2, S2 \gg}$$

$$\frac{\langle S'', F'' \rangle \text{ is redundant in } \ll G, F, S \gg \text{ w.r.t. } \Phi}{\ll G, F, S \gg \vdash_\Phi \ll G, F \setminus F'', S \setminus S'' \gg}$$

An inference system $\Vdash_\Phi$ between sets of constraints is defined to be the same as $\Vdash$, but using $\vdash_\Phi$ instead of $\vdash$ for single constraints.

Completeness of $\Vdash_\Phi$ does not hold. This is so because of the same considerations as in $\Vdash$ about using several constraints on one sort. Another reason for this is because, as has just been said, we have no complete system for redundancy, which leads to incompleteness of $\vdash_\Phi$.

Solving $MA \models MA$

Taking into consideration the proof criteria for redundancy, we can also give an overall inference system for mixed sets of constraints and sentences in **ALG**. Such a system must include: the inference system for sentences $\vdash^{\mathbf{ALG}}$, the inference system for sets of constraints $\Vdash$, an induction schema such as the one proposed for $MA \models A$ in section 3.3.2 and a rule to take care of redundant generators.

An inference system $\Vdash^{\mathbf{ALG}}$ between pairs consisting of a set of equations and a set of reachability constraints is defined by the following rule:

$$\frac{\Phi 2 \vdash^{\mathbf{ALG}+IND(C2)} \Phi 1 \qquad C2 \Vdash_{\Phi 2} C1}{\langle C2, \Phi 2\rangle \Vdash^{\mathbf{ALG}} \langle C1, \Phi 1\rangle}$$

where $\vdash^{\mathbf{ALG}+IND(C2)}$ is the inference system $\vdash^{\mathbf{ALG}}$ plus the induction rules which follow from reachability constraints in $C2$.

In the general case, the system is incomplete for several main reasons:

1. Incompleteness of the induction rule w.r.t. the inductive consequences of a theory.

2. Incompleteness of ~~set theory which results in incompleteness of~~ the second criterion for redundancy.

3. Incompleteness caused by the simultaneous use of constraints over the same sort.

4. Incompleteness caused by the existence of sets of sentences which can entail reachability constraints on their own.

The first two reasons have been sufficiently explained. The third refers to the incompleteness of $\Vdash$ unless the sets of constraints can be correctly joined as discussed in section 6.3.1 and in the example with three constants on page 210.

The fourth reason refers to those cases where reachability can be imposed by the sole use of sentences. In institutions such as **EQ** or **FOLEQ**, this is the case for presentations with only finite models. For example:

$$\forall x : bool.\ (x = true) \vee (x = false) \models \ll\{bool\}, \{true, false\}, \emptyset \gg$$

where a reachability constraint is equivalent to a first order sentence. In this case, we should explicitly add a rule inferring a reachability constraints from disjunctions of the form $\forall x : r.\ (x = v1) \vee (x = v2) \vee ... \vee (x = vn)$.

In other institutions the problem can become harder. For example, if we consider **FOL** with only enumerable models, rational numbers are axiomatizable using only sentences (see page 76 of [Men 71]). Therefore a reachability constraint over rational numbers follows from a set of first order sentences.

Summing up, incompleteness arises in very specific situations about which we can do very little in general. In practice, almost all entailments we find in algebraic specifications, and in particular the example on page 212 where the independent strategy fails, are derivable in the system presented above. For this reason, we shall consider $\Vdash^{\mathbf{ALG}}$ *satisfactory* w.r.t. $MA \models MA$ in **ALG**.

6.4 Structured specifications with constraints

Our purpose in this section is proving entailments $ASL \models ASL$ and $ASL \models M$ in particular, using the satisfactory inference systems already developed for $DATU \models DATU$ and $MA \models MA$.

Since reachability constraints can be seen as a particular kind of sentences and the inference system for $DATU \models DATU$ is parameterized by the underlying logic of the sentences $\vdash^{\mathbf{I}}$ but is institution independent otherwise, we obtain an inference system for $ASL \models ASL$ by using our satisfactory inference system for $DATU \models DATU$ and an underlying inference system dealing with ordinary sentences and constraints. In the case of **ALG** and restricting signature morphisms to be injective, $\Vdash^{\mathbf{ALG}}$ can be used.

In the following, we shall briefly revisit the inference rules for theorem proving and the inheriting strategy for D_σ but keeping in mind that sentences are reachability constraints or a mixture of constraints and another kind of sentences such as equations.

6.4.1 Proving constraints from specifications

In chapter 3 theorem proving was studied, in particular $DATU \models A$, and it was shown to be adequate for $DATU \models ATU$ due to the M-completeness of the inference rules for T_σ and U. Results in theorem 3.3.12 are independent of the nature of the sentences, and therefore they hold for reachability constraints. For example, we can say that

$$\frac{SP \vdash \ll \sigma, \iota \gg}{T_{\sigma'} SP \vdash \sigma'(\ll \sigma, \iota \gg)}$$ is sound and M-complete (exact)

and similarly for sentences being pairs consisting of a set of constraints and a set of ordinary sentences, *e.g.*

$$\frac{SP \vdash \langle \sigma(C), \sigma(\Phi) \rangle}{D_\sigma SP \vdash \langle C, \Phi \rangle} \qquad \text{is sound and complete}$$

As explained in chapter 3, SBO's with an exact inference rule can be eliminated to yield a *normalized* specification. That is why inference systems for $ATU \models ATU$, $MTU \models MTU$ and $MATU \models MATU$ are immediate from the underlying inference systems $\vdash^{\mathbf{I}}$, $\Vdash$ and $\Vdash^{\mathbf{I}}$ for $A \models A$, $M \models M$ and $MA \models MA$ respectively.

In the case of specifications with constructors $k{+}ASL \models A$, little has been said in general. For generic constructors, an explicitly given relation between sentences is all that can be inferred whereas for FQR-constructors the results given are only valid for equations.

If sentences happen to be constraints, generic constructors $\langle P, \Lambda \rangle$ should include an explicit relation between constraints in Λ. In the case of FRQ in the algebraic institution **EQ** with reachability constraints, some new inference rules are given below.

Proposition 6.4.1 $\dfrac{SP \vdash \ll G, F, S \gg}{Q_{eq} SP \vdash \ll G, F, S \gg}$ *is a sound inference rule.*

Proof Given a signature inclusion $\iota : \langle G \cup S, F \rangle \hookrightarrow Sig[SP]$ and an algebra $A \in Mod[SP]$, for $r \in G$ and for every $v \in |A|_r$ there exists a term $t_v \in |T_{\langle G \cup S, F \rangle}(X_S)|_r$ and a valuation $\nu : T_{\langle G \cup S, F \rangle}(X_S) \to A$ such that $\nu(t_v) = v$. By definition of the quotient algebra $A/\sim_A^{eq}$ there exists a surjective valuation $\nu_Q : T_{\langle G \cup S, F \rangle}(X_S) \to A/\sim_A^{eq} |_\iota$ defined as $\nu_Q(t) = [\nu(t)]$. Then $\nu_Q(t_v) = [v]$ and, since ν_Q is surjective, all values in $|A/\sim_A^{eq}|_r$ (quotient classes of values in $|A|_r$) are some valuation of some term in $T_{\langle G \cup S, F \rangle}(X_S)$.

Proposition 6.4.2 $\dfrac{}{R_G SP \vdash \ll G, \Omega, S \setminus G \gg}$ *where* $Sig[SP] = \langle S, \Omega \rangle$ *is a sound inference rule.*

Proof Given a $\langle S, \Omega \rangle$-algebra $A \in Mod[SP]$, by definition of $Reach_G$, $Reach_G(A)$ is reachable on sorts in G; that is to say, carriers $|Reach_G(A)|_r$ for $r \in G$ are finitely generated from sorts not in G using all the functions in Ω, hence $Reach_G(A) \models_{\langle S,\Omega \rangle} \ll G, \Omega, S \setminus G \gg$.

Proposition 6.4.3 $\dfrac{SP \vdash \ll G, F, S \gg}{R_{G'} SP \vdash \ll G, F, S \gg}$ *is a sound inference rule provided* $G' \cap S = \emptyset$ *and* $\ll G', \Omega', S' \setminus G' \gg$ *for* $\langle S', \Omega' \rangle = Sig[SP]$ *is* $\emptyset$*-free.*

Proof Given a $\langle S', \Omega' \rangle$-algebra $A \in Mod[SP]$, the operation $R_{G'}$ removes non-reachable values from carriers $|A|_r$ for sorts $r \in G'$. By cases:

- If $r \in G$, $R_{G'}$ may remove some values from $|A|_r$ but not all of them since $\ll G', \Omega', S' \setminus G' \gg$ is $\emptyset$-free. Hence, because $G' \cap S = \emptyset$ all values in $|Reach_{G'}(A)|_r$ continue to be the evaluation of a term in $T_{\langle G \cup S, F \rangle}(X_S)$ for a certain S-indexed set of variables X_S as they are in $|A|_r$.

- If $r \notin G$, $R_{G'}$ may remove some but not all values from $|A|_r$. Since r is neither a generated nor a generating sort in $\ll G, F, S \gg$, reachability of A on sorts G is not affected.

□

Proposition 6.4.4 $\dfrac{SP \vdash \ll G, F, S \gg}{F_\sigma^{eq} SP \vdash \ll \sigma(G), \Omega', S' \gg}$ *is a sound inference rule provided* $\restriction \sigma = \langle S', \Omega' \rangle$.

Proof A free extension can quotient the elements of the old carriers and add some new reachable values to them. Since quotients cannot prevent reachability from holding, all generated sorts continue to be generated.

Proposition 6.4.5 $\dfrac{}{F_\sigma^{eq} SP \vdash \ll G, \Omega', \sigma(S) \gg}$ *is a sound inference rule provided* $\sigma : \langle S, \Omega \rangle \to \langle \sigma(S) \uplus G, \Omega' \rangle$.

Proof By definition of F_σ^{eq}, carriers in the new sorts G are generated from the carriers in old sorts $\sigma(S)$.

6.4.2 Specifications with hidden generators

In chapter 4 some strategies to deal with hiding, $DATU \models DATU$, have been presented. Strategies and results are independent of the nature of the sentences and therefore immediately translate into strategies for $ASL \models ASL$. Nevertheless, checking for persistency and independence have a very different intuition when sentences are reachability constraints.

In the examples seen in chapter 4, persistency and independence looked like very reasonable requirements: Persistency as used there *guarantees that the specification of some auxiliary functions does not affect the visible part* and independence *makes the choice of a model of the whole specification independent of the choice of model for the auxiliary functions when they are not completely defined.* Some difficult cases where these requirements are not met, studied at the end of that chapter, stem from mistakes in the design of the specification or an incomplete specification of the auxiliary functions respectively, according to this view.

In chapter 5, the same proof strategy is applied to abstracted specifications. In that context, independence rarely holds and persistency is easily violated.

```
SP =    Enrich Bool by
          sorts elem, set
          operations
          axioms
        end
```

Figure 6–1: Booleans plus sorts for elements and sets

Now, if we consider that the hidden part of the consequent includes some of the generators and a constraint for a given sort, independence is generally lost and also persistency is lost unless the generated sort is hidden.

Example

Consider for example a specification of sets generated by $\emptyset$, singleton $\{\_\}$ and union $\_ \cup \_$ where the only visible functions are `insert`, `choose` and `remove`, such as *Set* in figure 6–2, built on top of *SP* in figure 6–1. A user may implement such a specification using a representation of sets by lists as described in the specification *Impl* in figure 6–3.

It is our goal to prove that $Impl \models Set$. Since *Set* has the form of a specification *SP* with a hidden and a visible enrichment we may apply the inheriting strategy.

First of all, it makes little sense to discuss the persistency of the hidden generators, $\{\emptyset, \_\cup\_, \{\_\}\}$, w.r.t. the visible sort `set`. In general, an enrichment including the hidden generators of a visible sort is never persistent w.r.t. the specification of the sort they generate, unless the generators are redundant. Moreover, independence fails since other operations on sets such as `insert` and `remove` are intimately dependent on the generators.

The most dramatic side of this problem is that the example corresponds to a common refinement style:

```
Set =    Enrich SP by
         Hidden
           sorts
           operations
             ∅:  set
             _∪_:  set, set -> set
             {_}:  elem -> set
           axioms
             Reachable on {set} using {∅, _∪_, {_}} from {elem}
             ∀ S:set. S ∪ ∅ = S
             ∀ S:set. S ∪ S = S
             ∀ S1, S2:set. S1 ∪ S2 = S2 ∪ S1
             ∀ S1, S2, S3:set. (S1 ∪ S2) ∪ S3 = S1 ∪ (S2 ∪ S3)
         in
           sorts
           operations
             insert:  set, elem -> set
             choose:  set -> elem
             remove:  set, elem -> set
             _∈_:  elem, set -> bool
           axioms
             ∀ e:elem; S:set. insert(S, e) = {e} ∪ S
             ∀ e1,e2:elem. e1 = e2 ⇔ e1∈{e2} = true
             ∀ e:elem. e∈ ∅ = false
             ∀ e:elem; S1,S2:set. e∈(S1∪S2) = (e∈S1 ∨ e∈S2)
             ∀ e:elem; S:set. e∈remove(S, e) = false
             ∀ e1,e2:elem; S:set. (e1≠e2) ⇒ (e2∈remove(S, e1) = e2∈S)
             ∀ S:set. (S≠ ∅) ⇒ (choose(S)∈S = true)
         end
```

Figure 6–2: Sets with hidden generators

```
Impl =    Enrich SP by
          Hidden
            sorts list
            operations
              α:  list -> set
              []:  list
              _::_:  elem, list -> list
            axioms
              Reachable on {list} using {[], ::} from {elem}
              Reachable on {set} using {α} from {list}
          in
            sorts
            operations
              insert:  set, elem -> set
              choose:  set -> elem
              remove:  set, elem -> set
              _∈_:  elem, set -> bool
            axioms
              ∀ e:elem; L:list. insert(α(L), e) = α(e::L)
              ...
              {Defining choose, remove and ∈ in terms of list operations}
              ...
          end
```

Figure 6–3: Sets represented by lists

First a data type is specified in terms of its own constructors using a reachability constraint and some axioms (e.g. sets in terms of $\emptyset$, $\{\_\}$ *and* $\cup$, *natural numbers in terms of 0 and* suc, *stacks in terms of* empty *and* push, *etc.), and then the implementation is carried out in terms of some primitive data types (e.g. lists, arrays, pointers, etc.) substituting the set of generators by a unique abstraction function,* α. *Moreover, if the original generators are too low level, they are hidden from the user.*

This may not be the most common approach when lists implement sets but it is certainly the approach used when arrays implement lists or stacks.

At first sight, our inference system for $DATU \models DATU$ seems to fail badly in all the interesting cases where the hidden functions are generators. However, we recall from chapter 4 that persistency of the hidden enrichment w.r.t. the specification on which it is built is *not necessary*; as explained in proposition 4.4.2, the hidden enrichment only needs to preserve those models which are actually used in the antecedent (implementation). This is also the case in the $\exists^2$-strategy, where $SP2 \models D_{\sigma 1;\sigma}(A_\Phi T_{\sigma 1;\sigma})SP2$ is required (see section 4.2).

Although both the $\exists^2$-strategy and the inheriting strategy as formulated in proposition 4.4.2 can be used, only the $\exists^2$-strategy will deliver some good results. The reason for this stems from the close relationship between the interpretation of a sort and the interpretation of its generators. This "strongly interdependent" relationship ruins independence of the hidden enrichment defining the generators and, in almost all cases, prevents the success of the inheriting strategy.

Using the $\exists^2$-strategy in our example, we are required to prove the persistency of an enrichment over *Impl* such as

```
EImpl =   Enrich Impl by
            sorts
            operations
```

```
    ∅:  set
    _∪_:  set, set -> set
    {_}:  elem -> set
  axioms
    Reachable on {set} using {∅, _∪_, {_}} from {elem}
    ∅ = α(□)
    ∀ e:elem. {e} = α([e])
    ∀ L1, L2:list. α(L1) ∪ α(L2) = α(append(L1, L2))
end
```

This holds since all possible sets generated from a list by α can also be generated by $\emptyset$, $\_\cup\_$ and $\{\_\}$ using their definitions in *EImpl*.

At this point we may recognize this as common practice in correctness proofs, particularly in abstract model specification. Nevertheless, this criterion is commonly used in an *ad hoc* fashion whereas here it belongs to a general strategy for D_σ in the particular case that sentences include reachability constraints and some generators are hidden.

Similarly to the last chapter for behavioural abstraction, the basic concepts of persistency and independence arise again as key factors in the proof of specification entailments. The multiple re-use of the same ideas shows the flexibility of our institution-independent results in chapter 4.

6.4.3 Behaviourally abstract specifications with hidden generators

Very often, sorts with hidden generators happen to be non-observable sorts. In this case, as discussed in [ST 88b], the reachability constraint does not constrain the re-

sulting specification since a non-reachable algebra is, most of time[3], behaviourally equivalent to its reachable subalgebra. However, many proofs of specification entailments extend the antecedent, *e.g.* using a persistent enrichment, to the point that the hiding in the consequent specification can be ignored; in this situation, also reachability constraints on non-observable sorts must be proven to hold in the extended antecedent.

In fact, when the hiding of constructors together with behavioural abstraction take place in a consequent specification, we run into sophisticated specification entailments known as data reification in abstract model specification languages (see for example chapter 8 in [Jon 86]).

Proofs can be carried out in our system using the $\exists^2$-strategy for the hidden generators and we can either prove all observable consequences or treat behavioural abstraction as hiding. Representation invariants like those used in [Jon 86] usually help to prove that the generators of the antecedent are redundant w.r.t. the generators of the same sort in the consequent and therefore the constraint in the consequent is persistent w.r.t. the antecedent.

Example

Consider an example of data reification where a table specified in terms of a list of $\langle index, value \rangle$ pairs, as shown in *Tabl*1 of figure 6–5, is implemented by a B-tree, as specified in *Tabl*2 of figure 6–6.

In this case we have to prove that $Tabl2 \models Tabl1$ where *Tabl*1 contains simultaneously hidden generators and non-observable sorts.

---

[3]This is always the case when observations are *ground* equations.

```
Pairs =    sorts index, value, pair, table
           operations
             (_,_): index, value -> pair
           axioms
           end
```

Figure 6–4: Index and value pairs plus a sort for tables

Behavioural abstraction cannot be ignored since theorems such as:

```
∀ i:index; v:value. remove(insert(T_empty, i, v), i) = T_empty
```

hold in *Tabl*1 but not in *Tabl*2. But, contrary to the case of stacks implemented by arrays and pointers, several different tables in *Tabl*1 can have a unique representation in *Tabl*2. In abstract model specifications this phenomenon is called *implementation bias* and makes proofs more complicated (cf. section 9.1 in [Jon 86]).

The entailment proof can be carried out as follows:

1. *The hidden enrichment of Tabl*1 *is added to Tabl*2 *and proven persistent.* This enrichment includes the specification of lists and the function $\alpha 1$ with the reachability constraint over `table` by $\alpha 1$, so that we obtain an enriched antecedent *ETabl*2 as in figure 6–7.

   Due to the lack of independence, $\alpha 1$ must be further specified in *ETabl*2 in terms of $\alpha 2$ and B-trees. This can be done by a single axiom φ as follows:

   $[\varphi]$ `∀ T:tree. α2(T) = α1(tree2list(T))`

   with `tree2list: tree -> list` delivering the preorder traversal of a tree in the form of a list.

   From this definition, all those tables generated by $\alpha 2$ from trees can also be generated using $\alpha 1$ from lists. This can be formally proven by showing

```
Tabl1 =   Non_observable on {table}
            Enrich Pairs by
              Hidden
                sorts list
                operations
                  α1:  list -> table
                  ...
                  {Standard operations on lists of ⟨index, value⟩ pairs}
                  ...
                axioms
                  Reachable on {table} using {α1} from {list}
                  Reachable on {list} using {[], ::} from {pair}
                  ...
                  {Standard axioms on lists of ⟨index, value⟩ pairs}
                  ...
              in
                sorts
                operations
                  T_empty:  table
                  insert:  table, index, value -> table
                  remove:  table, index -> table
                  look_up:  table, index -> value
                axioms
                  T_empty = α1([])
                  ∀ i:index; v:value; L:list. insert(α1(L), i, v) = α1((i,v)::L)
                  ∀ i:index; v:value; L:list. look_up(α1((i,v)::L), i) = v
                  ∀ i1,i2:index; v:value; L:list. i1≠i2 ⇒
                   look_up(α1((i2,v)::L), i1) = look_up(α1(L), i1)
                  ∀ i:index. remove(α1([]), i) = α1([])
                  ∀ i:index; v:value; L:list.
                   remove(α1((i,v)::L), i) = remove(α1(L), i)
                  ∀ i1,i2:index; v:value; L:list. i1≠i2 ⇒
                   remove(α1((i2,v)::L), i1) = insert(remove(α1(L), i1), i2, v)
              end
```

Tabl2 = Enrich *Pairs* by

Hidden

sorts tree

operations

mark: value

$\alpha 2$: tree -> table

...

{Standard operations on B-trees of pairs $\langle index, value \rangle$ ordered by index}

...

axioms

Reachable on {table} using {$\alpha 2$} from {tree}

Reachable on {tree} using {tree_empty, node} from {pair}

...

{Standard axioms for B-trees of pairs $\langle index, value \rangle$ ordered by index}

...

in

sorts

operations

T_empty: table

insert: table, index, value -> table

remove: table, index -> table

look_up: table, index -> value

axioms

T_empty = $\alpha 2$(tree_empty)

$\forall$ i:index; v:value; T:tree.

insert($\alpha 2$(T), i, v) = $\alpha 2$(tree_add(T, i, v))

$\forall$ i:index; v:value. look_up($\alpha 2$(leaf(i, v)), i) = v

$\forall$ i1,i2:index; v:value; T1,T2:tree.

(i1<i2 $\Rightarrow$ look_up($\alpha 2$(node(T1,T2,i2)), i1) = look_up($\alpha 2$(T1), i1))

$\wedge$ (i1$\geq$i2 $\Rightarrow$ look_up($\alpha 2$(node(T1,T2,i2)), i1) = look_up($\alpha 2$(T2), i1)

$\forall$ i:index; T:tree. remove($\alpha 2$(T), i) = $\alpha 2$(tree_add(T, i, mark))

end

```
ETabl2 =   Enrich Tabl2 by
             sorts list
             operations
               α1:  list -> table
               ...
               {Standard operations on lists of ⟨index, value⟩ pairs}
               ...
             axioms
               Reachable on {table} using {α1} from {list}
               Reachable on {list} using {[], ::} from {pair}
               ...
               {Standard axioms on lists of ⟨index, value⟩ pairs}
               ...
           end
```

Figure 6–7: Enriched antecedent specification

the redundancy of $\langle\{\texttt{tree}\},\{\alpha 2\}\rangle$ in $\ll\{\texttt{table}\},\{\alpha 2,\alpha 1\},\{\texttt{tree},\texttt{list}\}\gg\}$ w.r.t. a set of axioms Φ defining lists and trees plus the corresponding induction schemata, and using $\Vdash$ to prove that:

$$\{\ll\{\texttt{table}\},\{\alpha 2\},\{\texttt{tree}\}\gg\}\Vdash\{\ll\{\texttt{table}\},\{\alpha 2,\alpha 1\},\{\texttt{tree},\texttt{list}\}\gg\}$$
$$\Vdash_{\Phi}\{\ll\{\texttt{table}\},\{\alpha 1\},\{\texttt{list}\}\gg\}$$

Summing up, all models satisfying $\ll\{\texttt{table}\},\{\alpha 2\},\{\texttt{tree}\}\gg$ also satisfy $\ll\{\texttt{table}\},\{\alpha 1\},\{\texttt{list}\}\gg$; *i.e.* adding $\ll\{\texttt{table}\}.\{\alpha 1\},\{\texttt{list}\}\gg$ to *Tabl2* is a persistent enrichment.

We can assume the specification of lists to be persistent w.r.t. the definition of $\langle index, value\rangle$ pairs, hence the specification of lists is also persistent w.r.t. *Tabl2*.

2. *Prove that all visible consequences of Tabl*1 *hold in* $A_{\{\varphi\}}ETabl2$, *i.e.* all sentences such as

```
∀ i:index;v:value. look_up(insert(T_empty,i,v),i) = v
∀ i:index;v1,v2:value. look_up(insert(insert(T_empty,i,v1),i,v2),i) = v2
...
```

This amounts to restricting to those tables which can be generated by `T_empty`, `insert` and `remove`, under a context which can only be `look_up` since it is the only operation yielding an observable value from a non-observable table. We shall call the whole collection of visible sentences Δ_{inf}.

Due to the persistency of the enrichment over *Tabl*2 and since Δ_{inf} only refers to symbols defined in *Tabl*2, it suffices to prove that

$$Tabl2 \models \Delta_{\text{inf}}$$

but there are at least three ways to do this proof.

(a) The most straightforward solution seems to be structural induction on the tables used in Δ_{inf}, that is, those generated by `T_empty`, `insert` and `remove`.

In *Tabl*2, `T_empty`, `insert` and `remove` are specified in terms of $\alpha 2$ and trees. Hence, in order to inductively prove a theorem on `table` we need to do a simultaneous induction on sorts `table` and `tree`, where the predicate on sort `tree` stands for something like: *a tree generated by the definition of* `T_empty`, `insert` *and* `remove` *in Tabl*2. Such a predicate is usually known in abstract model specifications as a representation invariant. In this case, due to the complexity of `insert` and `remove`, it is very hard to find such an invariant. By the definition of B-tree we know that insertions and removals must preserve the balance of a tree but it is difficult to know, a priori, if some other subtle properties also hold.

Another problem is that there is no obvious way to inductively present Δ_{inf} without referring to lists. Unlike *Stack* and other specifications without hiding studied in [Hen 88], we cannot identify the operations used in the observable terms and the generators of tables.

(b) In order to overcome these difficulties, we can try to prove something stronger than Δ_{inf}, namely a property holding for all tables generated from lists instead of those generated by `T_empty`, `insert` and `remove`. Consider the following set of sentences from *Tabl*1:

$$[\Delta] \quad \begin{cases} \forall\ \texttt{i:index; v:value; L:list.}\ \texttt{look\_up}(\alpha 1((\texttt{i},\texttt{v})::\texttt{L}),\ \texttt{i}) = \texttt{v} \\ \forall\ \texttt{i1,i2:index; v:value; L:list.}\ \texttt{i1} \neq \texttt{i2} \Rightarrow \\ \quad \texttt{look\_up}(\alpha 1((\texttt{i2},\texttt{v})::\texttt{L}),\ \texttt{i1}) = \texttt{look\_up}(\alpha 1(\texttt{L}),\ \texttt{i1}) \end{cases}$$

Since all tables are reachable by $\alpha 1$ from a list and by definition of `insert` and `remove` in *Tabl*1, it is enough to prove that Δ is satisfied by $A_{\{\varphi\}}ETabl2$.

At this point we need to complete φ so that all tables generated from a list can also be generated from a tree.

$[\varphi']$ $\forall$ `L:list.` $\alpha 1$`(L)` = $\alpha 2$`(list2tree(L))`

with `list2tree: list -> tree` inserting all the elements of a list into an empty tree.

We could have chosen φ to be $\{\varphi,\varphi'\}$ in the first instance, but at that point there was no obvious need. Note that the pair of functions `list2tree` and `tree2list` plays the role of a many-to-many relation between lists representing tables and trees representing tables like the retrieve relation in reification proofs [Jon 86], but they are at the specification level instead of being model constructions.

Now, Δ can be proven to hold in $A_{\{\varphi,\varphi'\}}ETabl2$. First, $\alpha 1$ is replaced by its definition in terms of $\alpha 2$ and `list2tree` and then we can proceed by induction.

As before, we can devise a simultaneous induction on `tree` and `table` so that a predicate states an invariant over trees generated by `list2tree`, *e.g.* distinguishing trees with at most one value for each index. But, in contrast to the previous attempt, we expect to prove the two theorems in Δ rather than an infinite set of theorems Δ_{inf}.

(c) Alternatively, we can treat behavioural abstraction as the hiding of a predicate $\equiv_{Table}$ in *Tabl1* in the context of first order logic. The predicate $\equiv_{Table}$ should be explicitly defined by some axioms such as reflexivity, transitivity and congruence omitted in *Tabl1* and used in the following axioms:

```
T_empty ≡Table α1([])
∀ i:index; v:value; L:list. insert(α1(L), i, v) ≡Table α1((i,v)::L)
∀ i:index. remove(α1([]), i) ≡Table α1([])
∀ i:index; v:value; L:list.
  remove(α1((i,v)::L), i) ≡Table remove(α1(L), i)
∀ i1,i2:index; v:value; L:list. i1≠i2 ⇒
  remove(α1((i2,v)::L), i1) ≡Table insert(remove(α1(L), i1), i2, v)
```

Similarly to the example of stacks in the last chapter (see page 177), this hidden enrichment specifying the predicate $\equiv_{Table}$ is persistent w.r.t. to the specification of the sort `table` because two tables are never required not to be equivalent (see discussion at the end of section 5.4.2).

Applying the $\exists^2$-strategy we can define $\equiv_{Table}$ from *Tabl1* as a predicate $\equiv_{Tabl1}$ in *ETabl2*. When defining $\equiv_{Tabl1}$, it is difficult to identify exactly those trees which correspond to the same list for a certain sequence of operations on tables. It is easier to choose $\equiv_{Tabl1}$ big enough to include the desired cases and still respect different observations. In this particular example, we can define $\equiv_{Tabl1}$ to identify all those trees with the same most recent values for each index.

In the end, this approach might not be very different from the previous one. For example $\equiv_{Tabl1}$ can be defined in *ETabl2* as follows:

$\forall$ `T1,T2:tree.` `T1`$\equiv_{Tabl1}$`T2` $\Leftrightarrow$

`list2tree(tree2list(T1)) = list2tree(tree2list(T2))`

Proving that $\equiv_{Tabl1}$ is a congruence on sort `table` w.r.t. the rest of the operations in *ETabl2* requires us to prove that `look_up` does not distinguish between equivalent tables, and that is similar to proving $A_{\{\varphi,\varphi'\}}ETabl2 \models \Delta$ as above.

3. The rest of the proof is standard theorem proving.

Chapter 7

Structure and Proofs

7.1 Introduction

In previous chapters we were especially concerned with specification entailments where the consequent is flat or has a special structure. In the course of this analysis, three main problems in *ASL* proofs have arisen, namely: proving entailment w.r.t. specifications with hidden parts, proving entailment w.r.t. abstracted specifications and proving entailment involving specifications with reachability constraints. Now, we are acquainted with these problems and have some proof techniques to deal with them, however no attempt has been made to accommodate correctness proofs to the structure of specifications.

According to what has been said in previous chapters, confronting the problem of proving $SP2 \models SP1$ for arbitrary structured specifications $SP2$ and $SP1$ there is little we can directly do. In general, $SP1$ over *ASL* can be *flattened* by eliminating T_σ and U, putting together all axioms and all constraints and grouping all hiding into a single D_σ, so that $SP1$ becomes a specification such as $D_\sigma A_{(C,\Phi)\in\Sigma}$ for a set of ordinary Σ-sentences Φ and a set of Σ-constraints C. This is done by the systematic application of the reduction rules for T and U w.r.t. theorems and constraints – see sections 3.2 and 6.4.1 – plus the permutation rules for D_σ – see section 4.2.

Then, if D_σ is trivial, $SP2 \models D_\sigma A_{\langle C,\Phi\rangle \epsilon_\Sigma}$ can be solved by mixed theorem proving of ordinary sentences and constraints, as described in section 6.3.2. Otherwise, we try to reduce $SP1$ to a specification with a hidden and a visible enrichment, so that the inheriting strategy or the $\exists^2$-strategy can be used.

Summing up, we are using a two-step procedure: first the consequent is reduced (*normalized*) to a poorly structured specification and then a simple strategy based on mixed theorem proving completes the proof.

As it was already pointed out in section 3.2, this two-step procedure may make proofs harder or even impossible. Specifications $SP2$ and $SP1$ are, commonly, steps in a sequence of entailments from specification to implementation; therefore, we can expect many bits and pieces of $SP1$ to be shared by $SP2$, and both have a similar structure since only a few aspects of a specification are refined at a time. If a proof of $SP2 \models SP1$ ignores these structural similarities by flattening $SP1$, the proof becomes harder. Moreover, consequents with constructors cannot be flattened because the inference rules for the constructors are not M-complete, hence entailment proofs cannot use a two-step procedure as sketched above.

The main goal of this chapter is to show that matching the structure of the antecedent to the structure of the consequent makes correctness proofs easier.

Section 2 formalizes the relation between inference rules and specification transformations so that a homogeneous presentation of the two-step strategy can be given. Section 3 studies how whole subspecifications of the consequent can be viewed as single requirements and dealt with as a unit. Section 4 is concerned with modular implementations and, in particular, with the case of consequents with constructors, $SP1$ over $k{+}ASL$. In section 5 our attention is drawn to the structure of the antecedent and how it can ease theorem proving.

7.2 Revisiting *MATU*

Although in the previous chapters we seemed to encourage the use of reduction rules and inference systems for sentences (theorem proving) and constraints, we promised to give a more homogeneous presentation which allows taking advantage of the structure of specifications during entailment proofs. In this section, the so-called two-step strategy *flattening + theorem proving* is formalized as a particular sequence of applications of some specification transformations.

Firstly, it is shown how sound and M-complete inference rules give rise to specification entailments.

According to the basic proof rule presented in theorem 3.5.1, sound inference rules are to be used for pulling theorems out of the antecedent, therefore the following theorem is as expected.

Theorem 7.2.1 *For any sound inference rule* $\dfrac{SP \vdash f(\varphi)}{\xi SP \vdash g(\varphi)}$ *and any specification SP,*

$$\xi A_{\{f(\varphi)\}} SP \models A_{\{g(\varphi)\}} \xi A_{\{f(\varphi)\}} SP$$

Proof By definition of A_Φ we know that $A_{\{f(\varphi)\}} SP \models f(\varphi)$ and by definition of soundness, we conclude that $\xi A_{\{f(\varphi)\}} SP \models g(\varphi)$. By definition of A again,

$$Mod[A_{\{g(\varphi)\}} \xi A_{\{f(\varphi)\}} SP] = Mod[\xi A_{\{f(\varphi)\}} SP]$$

□

According to the basic proof rule presented in theorem 3.5.1, M-complete inference rules are to be used for pulling axioms out of the consequent, therefore the following theorem is as expected.

Theorem 7.2.2 *For any M-complete inference rule* $\dfrac{SP \vdash f(\varphi)}{\xi SP \vdash g(\varphi)}$ *and any specification* SP*, provided* ξ *distributes over* U*,*

$$A_{\{g(\varphi)\}}\xi SP \models \xi A_{\{f(\varphi)\}}SP$$

Proof By the definition of A and U, and the distributivity of ξ we know that:

$$Mod[\xi A_{\{f(\varphi)\}}SP] = Mod[\xi(SP\ U\ A_{\{f(\varphi)\}}\epsilon_{Sig[SP]})] = Mod[(\xi SP)U(\xi A_{\{f(\varphi)\}}\epsilon_{Sig[SP]})]$$

The theory $Cl(\{f(\varphi)\})$ is M-complete (in fact exact) w.r.t. $A_{\{f(\varphi)\}}\epsilon_{Sig[SP]}$, then by definition of an M-complete inference rule, $Cl(\{g(\varphi)\})$ is also an M-complete theory for $\xi A_{\{f(\varphi)\}}\epsilon_{Sig[SP]}$, *i.e.* $Mod[A_{\{g(\varphi)\}}\epsilon_{Sig[\xi SP]}] \subseteq Mod[\xi A_{\{f(\varphi)\}}\epsilon_{Sig[SP]}]$. Hence, by monotonicity of U,

$$A_{\{g(\varphi)\}}\xi SP = (\xi SP)U(A_{\{g(\varphi)\}}\epsilon_{Sig[\xi SP]}) \models (\xi SP)U(\xi A_{\{f(\varphi)\}}\epsilon_{Sig[SP]}) = \xi A_{\{f(\varphi)\}}SP$$

□

We recall from their definitions that most SBO's in *ASL* distribute over U (see proposition 2.2.6).

These results are valid for all sound and M-complete rules regardless of the nature of the axioms. In particular, axioms can be reachability constraints so that from sound and M-complete inference rules as above w.r.t. constraints, we obtain the corresponding entailments for pulling constraints:

$$A_{\{g(\ll\sigma,\iota\gg)\}}\xi SP \models \xi A_{\{f(\ll\sigma,\iota\gg)\}}SP$$

$$\xi A_{\{f(\ll\sigma,\iota\gg)\}}SP \models A_{\{g(\ll\sigma,\iota\gg)\}}\xi A_{\{f(\ll\sigma,\iota\gg)\}}SP$$

for all specifications SP.

A two-step strategy for proving correctness, as sketched at the beginning of this chapter, uses some reduction rules plus the inference rules for theorems and constraints. This strategy can be formalized as follows:

Definition 7.2.3 *Given an institution* **I**, *the inference system* $\vdash_F$ *for specification entailment in* k+*ASL*(**I**) *is composed of the following rules:*

1. *Basic proof rule:*

$$\frac{\Phi 2 \vdash^{\mathbf{I}} \Phi 1}{A_{\Phi 2} SP \vdash_F A_{\Phi 1} \epsilon_{Sig[SP]}}$$

2. *Pulling sentences (axioms) out of the consequent:*

$A_\Phi(SP1\ U\ SP2) \vdash_F (A_\Phi SP1)\ U\ SP2$ $\quad$ $A_{\sigma(\Phi)} T_\sigma SP \vdash_F T_\sigma A_\Phi SP$

$A_\Phi(SP1\ U\ SP2) \vdash_F SP1\ U\ (A_\Phi SP2)$ $\quad$ $A_{\Phi 1} A_{\Phi 2} SP \vdash_F A_{\Phi 2} A_{\Phi 1} SP$

3. *Pulling sentences (theorems) out of the antecedent:*

$SP1\ U\ (A_\Phi SP2) \vdash_F A_\Phi(SP1\ U\ (A_\Phi SP2))$ $\quad$ $T_\sigma A_\Phi SP \vdash_F A_{\sigma(\Phi)} T_\sigma A_\Phi SP$

$(A_\Phi SP1)\ U\ SP2 \vdash_F A_\Phi((A_\Phi SP1)\ U\ SP2)$ $\quad$ $D_\sigma A_\Phi SP \vdash_F A_{\sigma^{-1}(\Phi)} D_\sigma A_\Phi SP$

$\langle P, \Lambda \rangle A_\Phi SP \vdash_F A_{\{\varphi\}} \langle P, \Lambda \rangle SP$ $\quad$ *provided* $(\Phi, \varphi) \in \Lambda$

4. *Permuting D (equality* $=_F$ *means logical derivability* $\vdash_F$ *in both directions):*

$A_\Phi D_\sigma SP =_F D_\sigma A_{\sigma(\Phi)} SP$

$SP1\ U\ D_\sigma SP2 =_F D_\sigma(T_\sigma SP1\ U\ SP2)$

$D_\sigma SP1\ U\ SP2 =_F D_\sigma(SP1\ U\ T_\sigma SP2)$

$T_\sigma D_{\sigma 1} SP =_F D_{\sigma 1'} T_{\sigma'} SP$ *provided* $\sigma, \sigma 1, \sigma'$ *and* $\sigma 1'$ *form a pushout diagram as in proposition 4.2.1*

5. *Absorbing laws:*

$T_\sigma \epsilon_{\flat\sigma} =_F \epsilon_{\flat}$ $\quad$ $\epsilon_\Sigma\ U\ \epsilon_\Sigma =_F \epsilon_\Sigma$ $\quad$ $A_\emptyset \epsilon_\Sigma =_F \epsilon_\Sigma$

$A_{\Phi 1} A_{\Phi 2} SP =_F A_{\Phi 1 \cup \Phi 2} SP$ $\quad$ $D_\sigma D_{\sigma'} SP =_F D_{\sigma;\sigma'} SP$

6. Cut rule:

$$\frac{SP3 \vdash_F SP2 \quad SP2 \vdash_F SP1}{SP3 \vdash_F SP1}$$

Since all the rules of the system correspond to trivial equivalences or already proven theorems, soundness is immediate.

Fact 7.2.4 $\vdash_F$ *is sound w.r.t. specification entailment.*

Note that the SBO $M_{\langle\sigma,\iota\rangle}$ has been omitted since, without loss of generality, we can consider constraints to be a particular kind of sentences in Φ.

In the case of an institution **ALG** with algebras as models, injective morphisms and reachability constraints as a form of sentences, the basic rule becomes

$$\frac{\langle C2, \Phi 2\rangle \Vdash^{\mathbf{ALG}} \langle C1, \Phi 1\rangle}{A_{C2} A_{\Phi 2} SP \vdash_F A_{C1} A_{\Phi 1} \epsilon_{Sig[SP]}}$$

as in section 6.3.2. If **EQ** is considered we can add some rules for the *FRQ* constructors corresponding to the sound inference rules presented in sections 3.4 (for equations) and 6.4.1 (for reachability constraints).

By the nature of $\vdash_F$ typical correctness proofs of $SP2 \models SP1$ rely on the systematic application of the rules for pulling axioms out of the consequent and permutation laws for D_σ on $SP1$ until it is reduced — once simplified by the absorbing laws — to a flat specification of the form $D_\sigma A_{\Phi 1} \epsilon_\Sigma$,

$$D_\sigma A_{\Phi 1} \epsilon_\Sigma \vdash_F ... \vdash_F SP1$$

If D_σ is trivial the consequent is just $A_{\Phi 1} \epsilon_\Sigma$. On the side of the antecedent, we can pull out theorems

$$SP2 \vdash_F ... \vdash_F A_{\Phi 2} SP2$$

as required until correctness is immediately proven by, at most, one application of the basic rule. That is, we prove that

$$A_{\Phi 2} SP2 \vdash_F A_{\Phi 1} \epsilon_\Sigma$$

by proving that

$$\Phi 2 \vdash^{\mathbf{I}} \Phi 1$$

in the corresponding institution.

This inference system is the simplest we are going to consider and it produces correctness proofs according to a two-step procedure: *flattening + theorem proving* — from there comes the name, F for *flattening.*

Assuming a complete inference system $\vdash^{\mathbf{I}}$ for the underlying institution, the $\vdash_F$ system, although quite modest, it is complete for $DATU \models ATU$.

In the case of institutions where constraints are not considered as sentences, solving entailments in $DATU \models ATU$ is not enough. In order to solve entailments in $ASL \models MATU$, the institution needs to be extended to an institution with constraints and the corresponding inference system needs to be extended appropriately to handle constraints; *e.g.* $\Vdash^{\mathbf{ALG}}$ extends $\vdash^{\mathbf{ALG}}$. This results in an inference system $\vdash_F$ which is satisfactory for $ASL \models MATU$, already stronger that other strategies using an independent approach as suggested in [Bre 89] and [Far 89].

In order to handle D_σ in the consequent, the inference system $\vdash_F$ can be immediately improved by adding the following rule:

$$\text{[Bias]} \qquad \frac{SP2 \vdash_F SP1}{D_\sigma SP2 \vdash_F D_\sigma SP1}$$

This rule together with the cut rule allows the application of the $\exists^2$-strategy. We shall call such an enhanced system $\vdash_{VH}$ and, as shown in chapter 4, it is complete for $DATU \models DATU^*$ and satisfactory for $DATU \models DATU$ and for $ASL \models ASL$ if we use an institution with constraints as discussed above.

Nevertheless, none of these systems makes use of the possible common structure between antecedent and consequent. In the following section $\vdash_F$ is enriched to handle shared subspecifications.

7.3 Shared subspecifications

Specifications in *ASL* can be seen as structured collections of requirements. We can use requirements of two kinds, sentences and constraints, *i.e.* A_Φ and $M_{\langle\sigma,\iota\rangle}$, whereas the rest of the operations group these requirements using T_σ and U and sometimes relax them by means of abstraction/hiding D_σ.

According to this division of requirements, we are already provided with an inference system for sentences $\vdash^{\mathbf{I}}$ and an inference system for constraints, $\Vdash$ in the case of reachability constraints of the form $\ll G, F, S \gg$, and an inference system for their combination – see chapter 6.

In this section we treat whole subspecifications as single constraints, and in this sense U is considered as a third operation for introducing requirements. For example we can say that the following specification

$$M_{\langle\sigma,\iota\rangle} A_\Phi \epsilon_\Sigma \; U \; (T_\iota SSP)$$

consists of three requirements: the axioms Φ, the constraint $\ll\sigma,\iota\gg$ and the subspecification *SSP*.

Note that considering U as an operation which structures requirements or an operation which introduces a new requirement is a choice left to the prover and this choice may lead to very different proofs. Moreover, any argument of the U or both can be treated as single requirements.

Consider for instance an implementation of sets of natural numbers by lists of natural numbers. Sets of natural numbers are specified by:

```
SetNat =  Reachable on {set} using {∅, insert} from {nat}
            Enrich NatBoolby
              sorts set
              operations
                ∅:  set
                insert:  nat, set -> set
                ∈:  nat, set -> bool
              axioms
                ∀ n:nat. n∈ ∅ = false
                ∀ n:nat; S:set. n∈insert(n,S) = true
                ∀ n1,n2:nat; S:set. n1≠n2 ⇒
                  n1∈insert(n2,S) = n1∈S
                ∀ n1,n2:nat; S:set.
                  insert(n1, insert(n2, S)) = insert(n2, insert(n1, S))
                ∀ n:nat; S:set. insert(n, insert(n, S)) = insert(n, S)
            end
```

where *NatBool* is a specification of standard natural numbers as shown below:

```
NatBool =  Reachable on {nat} using {0, suc} from ∅
             Reachable on {bool} using {true, false} from ∅
               sorts nat, bool
               operations
                 0:  nat
                 suc:  nat -> nat
                 _+_:  nat, nat -> nat
                 true, false:  bool
                 _or_:  bool, bool -> bool
                 _>_:  nat, nat -> bool
               axioms
                 ∀ x:nat. x+0 = x
                 ∀ x,y:nat. x+suc(y) = suc(x+y)
```

```
        0>0 = false
        ∀ n:nat. suc(n)>0 = true
        ∀ x,y:nat. suc(x)>suc(y) = x>y
    end
```

Consider now, a implementation of *SetNat* by lists of natural numbers

```
ListNat =  Reachable on {list} using {[],_::_} from {nat}
             Enrich NatBoolby
               sorts list
               operations
                 []:  list
                 _::_:  nat, list -> list
                 count:  list, nat -> nat
               axioms
                 ∀ n:nat. count([],n) = 0
                 ∀ n:nat; L:list. count(n::L, n) = suc(count(L, n))
                 ∀ n1,n2:nat; L:list. n1≠n2 ⇒
                   count(n1::L, n2) = count(L, n2)
             end
```

and a constructor k such that $k(ListNat) \models SetNat$.

An entailment proof may consist of a proof that the axioms over set in *SetNat* hold in $k(ListNat)$, that the sort set is constrained in $k(ListNat)$ and that *NatBool* is also a subspecification of $k(ListNat)$. For the first two proofs we use theorem and reachability constraint proving rules respectively whereas for the third some new rules for inferring subspecifications must be developed.

If no inference system for subspecifications is provided the entailment proof could not consider *NatBool* as a single requirement. Then, the axioms in *NatBool* and its reachability constraint would be mixed up with those of *Set*, and only when the whole consequent is reduced to a single set of axioms and constraints could the real proof start, that is, as in the two-step strategy.

If we have an inference system for subspecifications, we can consider *NatBool* to be a single requirement and prove that it holds in *ListNat*. But we can also not consider it as a single requirement, or even try to consider *Set* as a single requirement. It is up to the prover to discover which is the good choice.

7.3.1 Inferring subspecifications

In this section we develop an inference system for specifications analogous to that for sentences and for reachability constraints.

In these systems there are two components, an underlying inference system such as $\vdash^{\mathbf{I}}$ or $\Vdash$, for sentences and reachability constraints respectively, plus a collection of M-complete and sound inference rules for the different SBO's.

With respect to the underlying inference system we choose a trivial one: the identity. In other words, a subspecification can only be inferred from itself. That is why we can only deal, at this stage, with subspecifications shared by the antecedent and the consequent. We can take account of this by changing the basic proof rule in $\vdash_F$ to

$$\frac{\Phi 2 \vdash^{\mathbf{I}} \Phi 1}{A_{\Phi 2} SP \ U \ SSP \vdash_F A_{\Phi 1 \in Sig[SP]} \ U \ SSP}$$

so that our "reduced" consequent contains some axioms $\Phi 1$ and a shared specification *SSP*. If we do not want to change the basic proof rule in $\vdash_F$ we can deal with shared subspecifications by adding the following rule to $\vdash_F$:

$$[Shared] \quad \frac{SP2 \vdash_F SP1}{SP2 \ U \ SSP \vdash_F SP1 \ U \ SSP}$$

Using this rule we can deal first with shared subspecification and then use $\vdash_F$ to perform an entailment proof as before.

This rule opens the topic of the next section: *modular proofs for modular implementations*. For the moment, we shall stick to the problem of shared subspecifications.

With respect to M-complete and sound rules for subspecifications, we bypass these concepts and directly give *pulling rules* analogous to those given in the last section. This simplifies previous considerations showing how sound and M-complete rules can be used to pull requirements out of the consequent and the antecedent.

Pulling subspecifications out of the consequent

We expect to reduce the consequent to a collection of requirements by pulling all of them out to the top level. Therefore we are after axioms of the following form:

$$\xi SP \ U \ SSP \vdash \xi(SP \ U \ SSP')$$

for specifications SP, SSP and SSP', and SBO ξ.

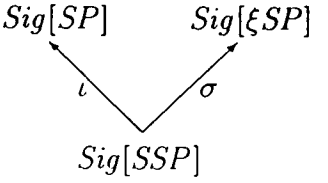
Proving a pulling-subspecification axiom for each SBO can be shortened by giving the following technical lemma. Recall from chapter 2 the definition of invariance (see page 32) and that $\{\_\}$ is an SBO such that for all Σ-models A the specification $\{A\}$ has signature Σ and models $\{A\}$.

Lemma 7.3.1 *If ξ is invariant w.r.t. SP and $\langle \iota, \sigma \rangle$, then*

$$\xi SP \ U \ T_\sigma SSP \models \xi(SP \ U \ T_\iota SSP)$$

provided ξ distributes over the union of model classes.

Proof Let B be a model of $\xi SP \ U \ T_\sigma SSP$. Since $B \in Mod[\xi SP]$, by distributivity of ξ w.r.t. the union of model classes, there exists a model $A \in Mod[SP]$ such that $B \in Mod[\xi\{A\}]$ and, by invariance of ξ, $A|_\iota = B|_\sigma$.



Since B is also a model of $T_\sigma SSP$, $B|_\sigma \in Mod[SSP]$. Therefore, by definition of T, A must be a model of $T_\iota SSP$ as well. In conclusion, $A \in Mod[SP \ U \ T_\iota SSP]$ and $B \in Mod[\xi(SP \ U \ T_\iota SSP)]$.

□

Now the actual rules are given and proven correct using the lemma.

Theorem 7.3.2 *The following axioms are sound w.r.t. specification entailment:*

$$A_\Phi SP\ U\ SSP \vdash A_\Phi(SP\ U\ SSP)$$

$$M_{(\sigma,\iota)}SP\ U\ SSP \vdash M_{(\sigma,\iota)}(SP\ U\ SSP)$$

$$(SP1\ U\ SP2)\ U\ SSP \vdash SP1\ U\ (SP2\ U\ SSP)$$

$$(SP1\ U\ SP2)\ U\ SSP \vdash (SP1\ U\ SSP)\ U\ SP2$$

$$T_\sigma SP\ U\ T_\sigma SSP \vdash T_\sigma(SP\ U\ SSP)$$

$$D_\iota SP\ U\ SSP \vdash D_\iota(SP\ U\ T_\iota SSP)$$

for all specifications $SP, SP1, SP2$ *and* SSP.

Proof All selectors L are invariant w.r.t. any specification SP and $\langle id_{Sig[SP]}, \dot{id}_{Sig[SP]}\rangle$, since by definition,

$$Mod[LSP] \subseteq Mod[SP]$$

for all specifications SP. In other words, $L\{A\}$ can only be $\{A\}$ or empty. Hence, the first four axioms referring to selectors A_Φ, $M_{(\sigma,\iota)}$ and U are sound.

T_σ is invariant w.r.t. any specification SP and $\langle id_{Sig[SP]}, \sigma\rangle$ since by definition for any model $A \in Mod[SP]$ all models $B \in Mod[T_\sigma\{A\}]$ have reduct $B|_\sigma = A$.

In the last case, D_ι is invariant w.r.t. a specification SP and $\langle \iota, id_{Sig[SP]}\rangle$ since for any model $A \in Mod[SP]$ and all $B \in Mod[D_\iota\{A\}]$ it holds that $B = A|_\iota$.

Finally, we recall from their definitions that all SBO's in ASL distribute over the union of model classes (see proposition 2.2.5).

□

Pulling subspecifications out of the antecedent

We expect to prove that a subspecification *SSP* of the consequent holds in the antecedent by inheriting *SSP* to the top of the antecedent, *i.e.* we are after rules of the following form:

$$\xi(SP\ U\ SSP') \vdash \xi(SP\ U\ SSP')\ U\ SSP$$

Similarly to the previous case, pulling subspecifications out of the antecedent is immediate for invariant SBO's.

Lemma 7.3.3 *If ξ is invariant w.r.t. $T_\iota SSP$ and $\langle \iota, \sigma \rangle$, then*

$$\xi(SP\ U\ T_\iota SSP) \models \xi(SP\ U\ T_\iota SSP)\ U\ T_\sigma SSP$$

provided ξ distributes over the union of model classes.

Proof Let B be a model of $\xi T_\iota SSP$. By distributivity of ξ over the union of model classes, there exists $A \in Mod[T_\iota SSP]$ such that $B \in Mod[\xi\{A\}]$ and, by invariance of ξ, $B|_\sigma = A|_\iota$. Since $A|_\iota \in Mod[SSP]$, $B \in Mod[T_\sigma SSP]$ and so, in general,

$$\xi T_\iota SSP \models T_\sigma SSP$$

By monotonicity of ξ, it follows that $\xi(SP\ U\ T_\iota SSP) \models T_\sigma SSP$ and therefore

$$\xi(SP\ U\ T_\iota SSP) \models \xi(SP\ U\ T_\iota SSP)\ U\ T_\sigma SSP$$

□

As a consequence of this lemma, pulling subspecification rules for A_Φ, $M_{\langle \sigma, \iota \rangle}$, U, T_σ and D_σ in the antecedent are sound since these SBO's are invariant and distribute over the union of model classes — see proof of theorem 7.3.2.

Corollary 7.3.4 *The following axioms are sound w.r.t. specification entailment:*

$$A_\Phi(SP\ U\ SSP) \vdash A_\Phi(SP\ U\ SSP)\ U\ SSP$$

$$M_{(\sigma,\iota)}(SP\ U\ SSP) \vdash M_{(\sigma,\iota)}(SP\ U\ SSP)\ U\ SSP$$

$$SP1\ U\ (SP2\ U\ SSP) \vdash (SP1\ U\ (SP2\ U\ SSP))\ U\ SSP$$

$$(SP1\ U\ SSP)\ U\ SP2 \vdash ((SP1\ U\ SSP)\ U\ SP2)\ U\ SSP)$$

$$T_\sigma(SP\ U\ SSP) \vdash T_\sigma(SP\ U\ SSP)\ U\ T_\sigma SSP$$

$$D_\iota(SP\ U\ T_\iota SSP) \vdash D_\iota(SP\ U\ T_\iota SSP)\ U\ SSP$$

for all specifications $SP, SP1, SP2$ *and* SSP.

There are other pulling subspecification rules for the antecedent corresponding to the constructors. In the following, we consider FRQ-constructors in **EQ**. Later, some comments are added for the case of generic constructors.

Proposition 7.3.5 *Given a signature morphism* $\iota : \langle S, \Omega\rangle \to \langle S', \Omega'\rangle$,

$$R_{\{s\}}(SP\ U\ T_\iota SSP) \vdash R_{\{s\}}(SP\ U\ T_\iota SSP)\ U\ T_\iota SSP$$

is sound w.r.t. specification entailment, provided $s \in (\Omega' \setminus \iota(\Omega))$ *or* $Mod[SSP]$ *is closed under subalgebras.*

Proof As in the previous lemma, it is enough to prove that $R_{\{s\}}\, T_\iota SSP \models T_\iota SSP$.

In the case where $s \in \Omega' \setminus \iota(\Omega)$, since $R_{\{s\}}$ only changes the carrier of s, we have that for all SP, $D_\iota R_{\{s\}} SP = D_\iota SP$. In the particular case of SP being $T_\iota SSP$ we obtain

$$D_\iota R_{\{s\}}\, T_\iota SSP = D_\iota\, T_\iota SSP$$

and applying T_ι to both sides and simplifying, we conclude that

$$R_{\{s\}}\, T_\iota SSP \models T_\iota D_\iota R_{\{s\}}\, T_\iota SSP = T_\iota D_\iota\, T_\iota SSP = T_\iota SSP$$

In the case where $Mod[SSP]$ is closed under subalgebras, we recall that for every algebra $A \in Mod[T_\iota SSP]$, the application of $R_{\{s\}}$ delivers a subalgebra $Reach_{\{s\}}(A)$ of A. Since reduct functors preserve subalgebras, $Reach_{\{s\}}(A)|_\iota$ is a subalgebra of $A|_\iota$. Since $A|_\iota \in Mod[SSP]$ and $Mod[SSP]$ is closed under subalgebras, it holds that $Reach_{\{s\}}(A)|_\iota \in Mod[SSP]$. Hence, $Reach_{\{s\}}(A) \in Mod[T_\iota SSP]$ and, in general, we conclude that

$$R_{\{s\}} T_\iota SSP \models T_\iota SSP$$

. □

Define a signature morphism σ as **forgetting a set of equations** eq iff $D_\sigma Q_{eq} SP = D_\sigma SP$ for all specifications SP [1, 2].

Proposition 7.3.6

$$Q_{eq}(SP \ U \ T_\iota SSP) \vdash Q_{eq}(SP \ U \ T_\iota SSP) \ U \ T_\iota SSP$$

is sound w.r.t. specification entailment, provided ι forgets eq or $Mod[SSP]$ is closed under quotients.

Proof By definition, if ι *forgets* eq, $D_\iota(Q_{eq} T_\iota SSP) = D_\iota T_\iota SSP$. Applying T_ι to both sides and simplifying,

$$Q_{eq} T_\iota SSP \models T_\iota D_\iota Q_{eq} T_\iota SSP = T_\iota D_\iota T_\iota SSP = T_\iota SSP$$

---

[1] An alternative definition can be given in terms of the congruence $\sim_{eq}$ induced by the set of equations eq: σ *forgets* eq *iff every sort with a pair of different elements* $(e1, e2) \in \sim_{eq}$ *is not the image of any sort in* $\downarrow \sigma$.

[2] In order to prove this property we can check that all the sorts mentioned in the equations eq are forgotten in σ plus those sorts which although not directly mentioned depend on the ones mentioned (see [KM 87] for a definition of dependency).

hence the entailment is correct as in the proof of lemma 7.3.3.

In the case where $Mod[SSP]$ is closed under quotients we can prove the same result. For all algebras $A \in Mod[T_\iota SSP]$, there is a homomorphism $k : A \to (A/\sim_{eq})$. Applying the reduct functor $-|_\iota$, we obtain $k|_\iota : A|_\iota \to (A/\sim_{eq})|_\iota$ and $(A/\sim_{eq})|_\iota$ is the quotient of $A|_\iota$ w.r.t. the kernel of $k|_\iota$, $(A|_\iota)/Ker(k|_\iota)$.

$$\begin{array}{ccc} A & & A|_\iota \\ \downarrow k & \stackrel{-|_\iota}{\Longrightarrow} & \downarrow k|_\iota \\ A/\sim_{eq} & & (A/\sim_{eq})|_\iota \end{array}$$

Finally, since $A|_\iota \in Mod[SSP]$ its quotient will also be in $Mod[SSP]$, $A|_\iota/Ker(k|_\iota) \in Mod[SSP]$, hence $(A/\sim_{eq})|_\iota \in Mod[SSP]$ and in general

$$D_\iota(Q_{eq}\, T_\iota SSP) \models SSP$$

Applying T_ι to both sides and simplifying we again obtain that $Q_{eq}\, T_\iota SSP \models T_\iota SSP$. □

Proposition 7.3.7

$$F_\sigma^{eq}(SP\ U\ SSP) \vdash F_\sigma^{eq}(SP\ U\ SSP)\ U\ T_\sigma SSP$$

is sound w.r.t. specification entailment, provided $(Free_\sigma^{eq} A)|_\sigma = A$ *for all* $A \in Mod[SSP]$[3].

Proof Since all constructors distribute over the union of model classes, it is enough to show that F_σ^{eq} is invariant w.r.t. SSP and $\langle id_{Sig[SSP]}, \sigma \rangle$ under the condition given.

---

[3]In this case sufficient completeness and consistency of $F_\sigma^{eq} SSP$ is not enough. This new requirement is usually called *strong persistency*, see [EM 85].

For every algebra $A \in Mod[SSP]$, $F_\sigma^{eq}\{A\}$ has a unique model $Free_\sigma^{eq} A$ such that

$$(Free_\sigma^{eq} A)|_\sigma = A$$

Hence, F_σ^{eq} is invariant w.r.t. SSP and $\langle id_{Sig[SSP]}, \sigma \rangle$. □

The implementation of sets of natural numbers by lists of natural numbers $k(ListNat) \models SetNat$ as described at the beginning of the section can be proven correct using the above rules.

Let us consider the whole subspecification *NatBool* to be a single requirement. Then, it must be proven that *NatBool* can be pulled out of *SetNat* using the rules for consequents, and out of $k(ListNat)$ using the rules for antecedents.

In first place we shall rewrite specifications *SetNat* and *ListNat* in a more concise form with *NatBool* as a distinguished subspecification, *i.e.*

$$SetNat = M_{Cset} A_{AxSet} T_{\iota_s}(\epsilon_\Sigma \ U \ NatBool)$$

$$ListNat = M_{Clist} A_{AxList} T_{\iota_l}(\epsilon_\Sigma \ U \ NatBool)$$

where M_{Cset} stands for the reachability constraint on `set` and $A_{AxSet} T_{\iota_s}$ stands for the enrichment adding sets, M_{Clist} stands for the reachability constraint on `list`, $A_{AxList} T_{\iota_l}$ stands for the enrichment adding lists and Σ is the signature of *NatBool*.

Since *SetNat* only contains T_σ and selectors, and all these have pulling subspecifications rules for the consequent, we can easily apply the rules in theorem 7.3.2 and conclude that,

$$\begin{aligned}(M_{Cset} A_{AxSet} T_{\iota_s} \epsilon_\Sigma) \ U \ (T_{\iota_s} NatBool) &\vdash M_{Cset}((A_{AxSet} T_{\iota_s} \epsilon_\Sigma) \ U \ (T_{\iota_s} NatBool)) \\ &\vdash M_{Cset} A_{AxSet}((T_{\iota_s} \epsilon_\Sigma) \ U \ (T_{\iota_s} NatBool)) \\ &\vdash M_{Cset} A_{AxSet} T_{\iota_s}(\epsilon_\Sigma \ U \ NatBool) = SetNat\end{aligned}$$

Similarly in the antecedent we can apply the rules in corollary 7.3.4 and conclude that

$$
\begin{aligned}
&k(ListNat) = kM_{Clist}A_{AxList}T_{\iota_l}(\epsilon_\Sigma\ U\ NatBool)\\
&\quad \vdash kM_{Clist}A_{AxList}((T_{\iota_l}(\epsilon_\Sigma\ U\ NatBool))\ U\ (T_{\iota_l}NatBool))\\
&\quad \vdash kM_{Clist}((A_{AxList}((T_{\iota_l}(\epsilon_\Sigma\ U\ NatBool))\ U\ (T_{\iota_l}NatBool)))\ U\ (T_{\iota_l}NatBool))\\
&\quad \vdash k((M_{Clist}((A_{AxList}((T_{\iota_l}(\epsilon_\Sigma\ U\ NatBool))\ U\ (T_{\iota_l}NatBool)))\ U\ (T_{\iota_l}NatBool)))\ U\ (T_{\iota_l}NatBool))
\end{aligned}
$$

Imagine the constructor is composed of the following SBO's:

1. A free extension $F^{eq2}_{\sigma 2}$ defining the membership operation $\in$ for lists, *e.g.* $\sigma 2$ is a inclusion morphism which adds the membership operation $\in$, and

   $$eq2 =_{def} \{\forall\ \mathtt{n : nat},\ \mathtt{L : list.}\ \mathtt{count(L, n)} > 0 = \mathtt{n} \in \mathtt{L}\}$$

   defines $\in$ on lists.

2. A derive $D_{\sigma 1}$ forgetting `count` and renaming `list` to `set`, `[]` to $\emptyset$ and `_ :: _` to `insert`.

3. A quotient Q_{eq1} identifying those lists which have different numbers of occurrences of the same elements or different ordering, *e.g.*

   $$
   \begin{aligned}
   eq1 =_{def}\ \{\ &\forall\ \mathtt{n:nat;\ S:set.\ insert(n,\ insert(n,\ S))\ =\ insert(n,\ S)},\\
   &\forall\ \mathtt{n1,n2:nat;\ S:set.}\\
   &\quad \mathtt{insert(n1,\ insert(n2,\ S))=insert(n2,\ insert(n1,\ S))}\}
   \end{aligned}
   $$

Summing up, the constructor is defined as

$$k =_{def} Q_{eq1}D_{\sigma 1}F^{eq2}_{\sigma 2}$$

The subspecification *NatBool* can be pulled along this constructor using the rules presented above as follows:

1. Since $Free^{eq2}_{\sigma 2}$ is strongly persistent we can pull the subspecification and obtain

$$Q_{eq1} D_{\sigma 1}((F^{eq2}_{\sigma 2}((M_{Clist}...))) \; U \; (T_{\iota_l;\, \sigma 2} NatBool))$$

2. Morphism $\sigma 1$ adds count and renames other symbols in the signature $\uparrow \iota_s$ so that we can write $\iota_l;\, \sigma 2$ as $\iota_s;\, \sigma 1$, therefore the rule for $D_{\sigma 1}$ in corollary 7.3.4 can pull the subspecification $T_{\sigma 1} T_{\iota_s} NatBool$ and obtain

$$Q_{eq1}((D_{\sigma 1}((F^{eq2}_{\sigma 2}...))) \; U \; (T_{\iota_s} NatBool))$$

3. Finally, equations $eq1$ do not affect `nat`, *i.e.* ι_s forgets $eq1$, hence

$$(Q_{eq1}(D_{\sigma 1}...)) \; U \; (T_{\iota_s} NatBool)$$

At this point the [*Shared*] rule can be applied to *match* the subspecifications $T_{\iota_s} NatBool$ of the consequent and the antecedent, reducing the entailment to

$$Q_{eq1}((D_{\sigma 1}((F^{eq2}_{\sigma 2}...))) \; U \; (T_{\iota_s} NatBool)) \models M_{Cset} A_{AxSet} T_{\iota_s} \epsilon_{\Sigma}$$

which can be proven if $\vdash_F$ is extended with the rules corresponding to the sound inference rules for *FRQ* given in sections 3.4 and 6.4.1, as discussed above.

Note that rules pulling sentences and subspecification out of the antecedent enlarge the antecedent specifications, most of the time unnecessarily. This behaviour can be thought like that of a theorem prover which adds all inferred theorems from a presentation as axioms to the original presentation. In all the cases which have been considered, the rules for pulling sentences and subspecifications can be turned into equalities (bi-directional rules) in order to solve that problem.

Analogously to the inference rules for constructors in chapter 3, pulling subspecifications along constructors is difficult. Most of the axioms given above have

strong side conditions restricting their use to the simplest cases and some of the side conditions are model-theoretical. Sufficient syntactic conditions can be given in most of the cases, but like theorem proving for FQR-constructors in chapter 3, the model-theoretical nature of the FQR-constructors prevents better results.

Generic constructors $\langle P, \Lambda \rangle$ are useless for anything except theorem proving or, at most, mixed theorem proving depending on the nature of Λ. If we would like to treat subspecifications as single constraints, generic constructors must be enriched with some information about the symbols which are modified and those which are not. This can be achieved by changing the notation for constructors to record explicitly over which subspecifications they are invariant.

Definition 7.3.8 *A well-formed* **S-constructor** *is a triple* $\langle P, \Lambda, \langle \iota, \sigma \rangle \rangle$, *where* $\langle P, \Lambda \rangle$ *is a generic constructor with*

$$[\![P]\!] = (\uparrow \iota, \uparrow \sigma, f_P)$$

for some function $f_P : |Mod(\uparrow \iota)| \rightarrow |Mod(\uparrow \sigma)|$, *and for every specification* SP *over* $\uparrow \iota$ *the* $\langle P, \Lambda \rangle$ *is invariant w.r.t.* SP *and* $\langle \iota, \sigma \rangle$.

Trivially generic constructors are a particular case of S-constructors where $\downarrow \iota = \downarrow \sigma$ is the empty signature.

Since constructors distribute over the union of model classes, we can use lemma 7.3.3 to conclude that all subspecifications SSP over $\downarrow \sigma = \downarrow \iota$ can be pulled out through $\langle P, \Lambda, \langle \iota, \sigma \rangle \rangle$, *i.e.*

$$\langle P, \Lambda, \langle \iota, \sigma \rangle \rangle (SP \ U \ T_\iota SSP) \vdash (\langle P, \Lambda, \langle \iota, \sigma \rangle \rangle (SP \ U \ T_\iota SSP)) \ U \ T_\sigma SSP$$

Adding the rules for pulling subspecifications and the [*Shared*] rule to $\vdash_F$ we obtain $\vdash_{AS}$, that is, an inference system combining a basic rule for axioms (the basic rule in $\vdash_F$) and a basic rule for subspecifications (the [*Shared*] rule).

Taking into consideration that sentences in A_Φ can also be constraints, we have, in practice, a two step inference system *flattening + requirement proving* with three kinds of requirements: ordinary sentences, constraints and subspecifications.

Formal systems which include a treatment of subspecifications are a great deal better that previous ones; however, there is not a clear characterization of them unless we restrict attention to modular entailments, as discussed below.

7.4 Modular entailments

In the introduction to this chapter we stressed that proofs of $SP2 \models SP1$ should be able to take advantage of the similarities between the antecedent and the consequent, in particular, their structural similarities.

The work on shared subspecifications should be complemented with a structure-matching mechanism, so that we do not need to pull out subspecifications in cases where antecedent and consequent specifications differ only in a small bit, deep in the structure. The obvious solution is a simplification rule

$$[Simpl] \qquad \frac{SP2 \vdash SP1}{\xi SP2 \vdash \xi SP1}$$

whose soundness follows from the monotonicity of all SBO's (see proposition 2.2.4).

Although [*Simpl*] seems rather simple, it is an extremely powerful rule. For instance, adding [*Simpl*] to $\vdash_F$ permits to derive rules such as [*Bias*] (page 248) and [*Shared*] (page 252). In fact, the combination of $\vdash_F$ with [*Simpl*] plus the rules for pulling subspecification form $\vdash_M$, the most powerful inference system we are going to consider.

In order to characterize $\vdash_M$, we restrict to the study of *modular entailments*, that is, those specification entailments where the antecedent respects the structure of the consequent. For example, let *ListNat* and *NatBool* be specifications

like those considered above, and *SetNat'*, *ListNat'* and *NatBool'* be specifications analogous to *SetNat*, *ListNat* and *NatBool* but without any reachability constraint on `nat`, then $k(ListNat) \models SetNat'$ can be proven correct by proving, roughly, that

$$NatBool \models NatBool'$$

$$k(ListNat') \models SetNat'$$

since applying the simplification rule to the first entailment and combining the two entailments (cut rule) we obtain:

$$\frac{\dfrac{NatBool \vdash_M NatBool'}{k(ListNat) \vdash_M k(ListNat')} \qquad k(ListNat') \vdash_M SetNat'}{k(ListNat) \vdash_M SetNat'}$$

Then, the entailment $NatBool \models NatBool'$ is proven using $\vdash_F$ and $k(ListNat') \models SetNat'$ is proven using the rules for pulling subspecifications and sentences in $\vdash_M$.

In the case of parameterized specifications in [SW 83, ST 88b], a modular entailment $P2(SP2) \models P1(SP1)$ can be proven correct by proving that

$$P2 \models P1 \qquad\qquad SP2 \models SP1$$

where $P2 \models P1$ means that $P2(SP) \models P1(SP)$ for any specification SP of the right signature.

There is a clear analogy between the proofs of $P2(SP2) \models P1(SP1)$ and of $k(ListNat) \models SetNat'$. Taking specifications $SP2$ and $SP1$ to be the subspecifications *NatBool* and *NatBool'*, and $P2$ and $P1$ to be

$$P2 = \lambda\, X : \Sigma.\ k M_{Clist} A_{AxList} T_{\iota_l}(\epsilon_\Sigma\ U\ X)$$

$$P1 = \lambda\, X : \Sigma.\ M_{Cset} A_{AxSet} T_{\iota_s}(\epsilon_\Sigma\ U\ X)$$

our proof in $\vdash_M$ becomes

$$P2(SP2) \vdash_M P1(SP2) \qquad\qquad SP2 \vdash_M SP1$$

Although $P2 \models P1$ is a stronger requirement than $P2(SP2) \models P1(SP2)$, often the derivation $P2(SP2) \vdash_M P1(SP2)$ treats $SP2$ as a shared subspecification and uses rules for pulling subspecifications which do not rely on the models of $SP2$. In this case, the same proof schema can be used for any Σ-specification SP taking the place of $SP2$; in other words, we have a *proof* of $P2 \models P1$.

In some specification languages requiring parameterized specifications to denote persistent free extensions F^{eq}_{σ} (*e.g.* [EM 85]), parameterized specifications are always invariant w.r.t. any specification of the right signature and $\langle id_{|\sigma}, \sigma \rangle$. Therefore, the part of the proof handling shared parameters (subspecifications) is trivial. Other languages with a more general parameterization mechanism, such as λ-abstraction in $\lambda + ASL$, cannot use these shortcuts in general and rules for pulling subspecifications are needed.

In this section we are mainly concerned with the use of the simplification rule and the cases where its application is safe, *i.e.* its application in goal-oriented (backwards) proofs cannot take us from a correct entailment to an entailment which cannot be proven correct. In this sense, we shall see how modular implementations lead to modular proofs of correctness.

7.4.1 Specifications as arrows

A clean notation for dealing with structure is to consider the category of signatures and their products as objects, and ASL specifications, either ground or parameterized, as arrows, analogously to the concrete free theories used by Lawvere (see section 2.3).

For practical reasons we shall not consider arrows to be single specifications but sets of equivalent specifications, *i.e.* specifications with the same signature

and class of models and, in the parameterized case, parameterized specifications with the same signature pair and function between classes of models.

Fact 7.4.1 *Given an institution with category of signatures SIG (closed under finite products) and model functor Mod then an ordered category Spec(SIG) is defined as follows:*

- **Objects**: *Objects in SIG.*
- **Morphisms**: *Triples* $\langle \Sigma 1, F, \Sigma 2 \rangle : \Sigma 1 \rightarrow \Sigma 2$ *where* $\Sigma 1$ *and* $\Sigma 2$ *are signatures in SIG and F is a function from model classes over* $\Sigma 1$ *to model classes over* $\Sigma 2$, *i.e.* $F : \mathcal{P}(|Mod(\Sigma 1)|) \rightarrow \mathcal{P}(|Mod(\Sigma 2)|)$.
- *Composition is as expected:* $\langle \Sigma 1, F1, \Sigma 2 \rangle ; \langle \Sigma 2, F2, \Sigma 3 \rangle = \langle \Sigma 1, F1; F2, \Sigma 3 \rangle$.
- *Homsets are ordered as follows:* $\langle \Sigma 11, F1, \Sigma 12 \rangle \leq \langle \Sigma 21, F2, \Sigma 22 \rangle$ *iff* $\Sigma 11 = \Sigma 21$, $\Sigma 12 = \Sigma 22$ *and* $\forall M \subseteq |Mod(\Sigma 11)|$. $F1(M) \subseteq F2(M)$.

Ordered categories can generally be viewed as 2-categories. Then, composition as it has been defined above is called horizontal composition and the ordering induces another kind of composition known as vertical composition (following the terminology in [GB 80]).

Moreover, order preservation by composition in $Spec(SIG)$ is equivalent to the compatibility of horizontal and vertical composition for refinement, discussed in general in [GB 80] and proved for refinement relations in [ST 88b].

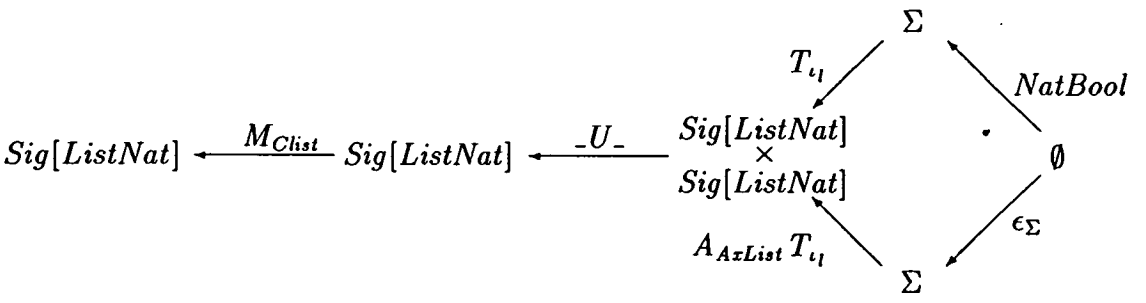
Note that all SBO's such as T_σ, A_Φ, $(\lambda X : \Sigma . SP1)$ and $\langle P, \Lambda \rangle$ correspond to arrows in $Spec(SIG)$. In particular, specifications SP in $\lambda + k + ASL$ correspond to arrows from the empty signature to $Sig[SP]$. For example, the specification $NatBool$ with $\Sigma = Sig[NatBool]$ corresponds to an arrow

$$\Sigma \xleftarrow{\text{NatBool}} \emptyset$$

and $ListNat = PList(NatBool)$ with $PList = \lambda X : \Sigma.\ M_{Clist}((A_{AxList}\, T_{\iota_1} \epsilon_\Sigma)\, U(T_{\iota_1} X))$ can be represented by a compound arrow

$$Sig[ListNat] \xleftarrow{\quad PList \quad} \Sigma \xleftarrow{\quad \text{NatBool} \quad} \emptyset$$

or, if the finer structure is considered, we obtain



Note that the existence of finite products of signatures is needed to represent SBO's or parameterized specifications with several parameters, *e.g.* $\_U\_$.

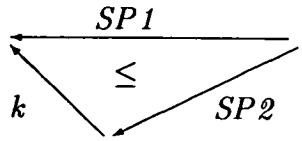
Following [ST 88b], refinements can be replaced by constructor implementations in order to capture the fact that some constructor (piece of code) can reduce our original specification (task) to a simpler one. The composition of constructor implementations and its compatibility with horizontal composition is rephrased as follows according to our new terminology.

Given a generic constructor $\langle P, \Lambda \rangle$ with $[\![P]\!] = (\Sigma 1, \Sigma 2, f_P)$, there is a morphism $\langle \Sigma 1, F, \Sigma 2 \rangle$ in $Spec(SIG)$ associated to that constructor where F is defined as $F(M) = \{f_P(A) \mid A \in M\}$ for all $M \subseteq |Mod(\Sigma 1)|$. Morphisms in $Spec(SIG)$ whose function F is defined by *lifting* a function between single models form a subcategory $Func(SIG)$ of $Spec(SIG)$, *i.e.* such morphisms include the identities in $Spec(SIG)$ and are closed under composition.

Generic constructors are defined by programs P in a language L, therefore F should be a recursive function and P a procedure to compute it. Nevertheless, we shall proceed ignoring this aspect of the nature of constructors. Constructors can

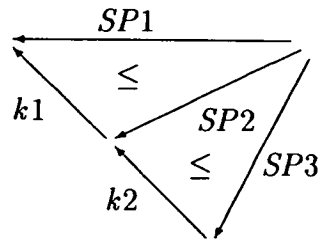
be either *FQRD*-constructors, generic constructors, S-constructors or anything else with semantics in terms of functions between models but, for the time being, they are just treated as morphisms in a subcategory of *Spec*(*SIG*).

Definition 7.4.2 *A specification SP1 is implemented by an specification SP2 via a constructor k iff the following diagram commutes in Spec*(*SIG*).

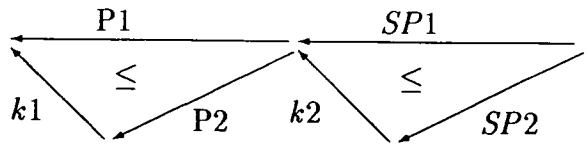


Note: In a ordered category, such a diagram commutes if $SP2;\ k \leq SP1$, *i.e.* if $k(SP2) \models SP1$.

Since constructors are closed under composition, constructor implementations compose as expected. For instance in a situation such as the following diagram, it holds that $SP3;\ k2;\ k1 \leq SP1$.



Compatibility of horizontal and vertical composition also follows immediately from the composability of constructors and the fact of working in an ordered category. For instance in a situation such as the following diagram, it holds that $SP2;\ k2;\ P2;\ k1 \leq SP1;\ P1$.

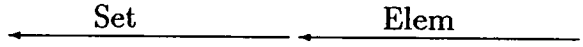


7.4.2 Modular programming

Modular programming amounts to the independent implementation of the different *pieces* of the specification. This simple idea is the clue for simplifying the verification and providing reusable implementations.

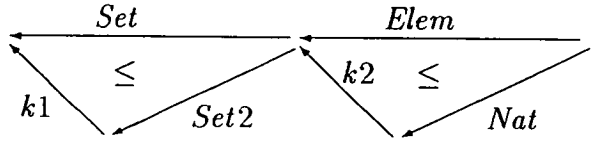
Some approaches distinguish between structured and modular specifications, where only the second kind of structure needs to be respected in the implementation (*e.g.* ACT ONE). Others prefer to see modular decomposition as an implementation decision which therefore is not reflected in the specification language (*e.g.* CLEAR and ASL). Many specification languages, despite using just one structuring mechanism, provide key words such as *protected* or *use* which allow modular implementation (programming) to be required by the specifier.

For us, all these differences are just different diagrams relating the antecedent to the consequent of a specification entailment. Imagine a consequent $Set(Elem)$ specifying sets of elements



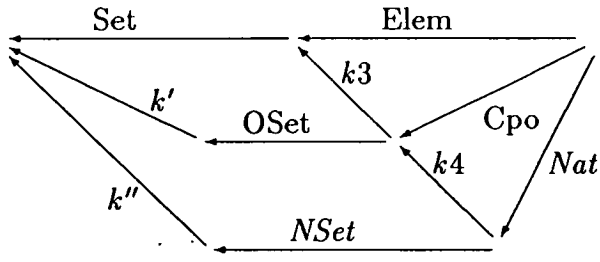
where the source signature of *Elem* is the empty signature or, more generally, the signature of the basic data types of our favorite programming language.

Consider an entailment $k1(Set2(k2(Nat))) \models Set(Elem)$ such that the following diagram commutes (*i.e.* $Set2$; $k1 \leq Set$ and Nat; $k2 \leq Elem$):



In this case, we say that the entailment is *modular* since $k2(Nat) \models Elem$ and $k1(Set2(SP)) \models Set(SP)$ for all specification SP of the right signature.

Other antecedents relying on the fact that the elements are natural numbers, or simply that they are ordered, give rise to the following diagrams:



where *Nat*; *NSet*; $k'' \leq Elem$; *Set* and *Cpo*; *OSet*; $k' \leq Elem$; *Set* and therefore

$$k''(NSet(Nat)) \models Set(Elem) \qquad k'(OSet(Cpo)) \models Set(Elem)$$

However, *OSet* is only a correct implementation of *Set* provided there is an order relation among the elements, and *NSet* is only a correct implementation of *Set* if the elements are natural numbers. Then, we say that these entailments are not *modular.*

Definition 7.4.3 *An entailment* $P2(SP2) \models P1(SP1)$ *is* **modular** *iff* $P2 \leq P1$ *and* $SP2 \leq SP1$.

In the example above, $k1(Set2(k2(Nat))) \models Set(Elem)$ is modular whereas

$$k'(OSet(Cpo)) \models Set(Elem) \text{ and } k''(NSet(Nat)) \models Set(Elem)$$

are not modular.

In terms of entailment between parameterized specifications, as in [ST 88b], we can say that $k1(Set2) \models Set$ since for all specification *SP* of the right signature $k1(Set2(SP)) \models Set(SP)$. On the other hand, $k'(Oset(SP)) \models Set(k3(SP))$ does not necessarily hold unless $SP \models Cpo$ and $k3(Cpo) \models Elem$.

In programming languages we can find the same idea of *module* referring to the compilation of programs, instead of referring to specification entailment. For example, if a program $k2$; $k1$ is compiled to an object $o2$; $o1$ using separate compilation, the linker can still reuse object $o1$ when the code in $o2$ is changed.

In specification languages, modular or separate entailment (opposed to separate compilation for programming languages) is very common but explicit semantics are rarely given. Remarkable exceptions are the stratified semantics of PLUSS [Bid 88] and ASL with Π-abstraction as in [SST 90].

Π-abstraction, contrary to standard λ-abstraction in $\lambda + ASL$, forces correct entailments to be modular. This is a very reasonable restriction to a general syntactic mechanism based on substitution such as λ-parameterization. Similarly, classic parameterization mechanisms in algebraic specifications, as in [EM 85], consider also a syntactic construction, in this case the pushout of specifications (equational presentations). In turn, this pushout construction is required to agree with a functor interpretation of parameterized specifications. The existence of a functor interpretation is less demanding and, probably, more elegant than imposing modular entailment or stratified semantics; however, the latter caters to the urgent need of easing verification while the former does not.

The fact of a entailment being modular has immediate consequences in terms of verification techniques:

Fact 7.4.4 *For arbitrary specifications $SP2$ and $SP1$, and an arbitrary SBO ξ,*

$$\frac{\xi(SP2) \vdash \xi(SP1)}{SP2 \vdash SP1} \quad \textit{provided } \xi(SP2) \models \xi(SP1) \text{ is modular}$$

is sound w.r.t. specification entailment.

Proof Immediate from the definition of modular entailment.

This fact guarantees that the tactic associated with the simplification rule (backwards application) is not just sound but safe for modular entailment, in

the sense that its application in proof search cannot reduce a valid purported entailment to a wrong one.

This is of great significance since it means that we do not need to carry the whole antecedent around in our proofs, but just the module of the antecedent corresponding to the module of the consequent we want to verify. In the large, a modular entailment can be proven correct in a module-by-module proof, *i.e.* a modular proof.

7.4.3 Modular programming and constructors

Hitherto constructors have been viewed as arrows in $Spec(SIG)$ and therefore as particular specification building operations. However there is a fundamental difference between a constructor and a specification from the software development point of view: *Once we have got a constructor we do not expect to implement it any further.* At most, constructors may be refined by other constructors (*program transformation*) but here we are not concerned with this problem.

This principle of software development is reinforced by a proof-theoretic fact: *Constructors lack M-complete inference rules.* For this reason, neither generic constructors nor $FQRD$-constructors in **EQ** can be *flattened* to a set of axioms and constraints. In an inference system such as $\vdash_M$, this means that entailments with constructors in the consequent can only be proven correct if the same constructors are also used in the antecedent and they can eventually be *simplified* using [*Simpl*].

This is particularly obvious for S-constructors, since we explicitly ignore in $\vdash_M$ what an S-constructor $\langle P, \Lambda, \langle \iota, \sigma \rangle\rangle$ does except for what Λ, ι and σ can tell us. Hence, $SP2 \models \langle P, \Lambda, \langle \iota, \sigma \rangle\rangle SP1$ cannot be proven correct unless $\langle P, \Lambda, \langle \iota, \sigma \rangle\rangle$ is also present in $SP2$. Moreover, no requirement in the consequent — neither axiom, reachability constraint nor subspecification — can be pulled through an S-constructor (generic constructor), hence entailment proofs must treat the re-

quirements in a consequent with a S-constructor as two separated blocks, those requirements preceding the S-constructor and those requirements following it.

If *FQRD*-constructors are considered the result is similar. Inference rules for F_σ^{eq}, Q_{eq} and $R_{\{s\}}$ are sound but not M-complete, hence no rules for pulling sentences (or constraints) out of the consequent appear in $\vdash_M$.

We can conclude that for derivable specification entailments in $\vdash_M$, intermixing constructors with specifications encourages modular programming. In fact, we can expect common entailments to be *K-modular.*

Definition 7.4.5 *Given a subcategory of constructors K in Spec(SIG), a entailment $SP2 \models SP1$ is* **K-modular** *if for all constructors $k \in |K|$ and decomposition $P1(k(SP1'))$ of $SP1$, there exists a decomposition $P2(k(SP2'))$ of $SP2$ such that*

$$P2 \leq P1 \qquad\qquad SP2' \leq SP1'$$

Immediately from this definition and the simplification rule, it follows that:

Fact 7.4.6 *Given a specification language Lang, a language of constructors K and a complete inference system $\vdash_L$ for entailments in $K + Lang \models Lang$, then $\vdash_L$ with the simplification rule is complete w.r.t. K-modular entailments in $K + Lang \models K + Lang$.*

Restricting to K-modular entailments allows constructors, which do not have M-complete inference rules, to be used in the consequent. However, for S-constructors (generic constructors) and *FQRD*-constructors, not even complete inference rules are available. The lack of complete rules prevents having a complete inference system for $K + ASL \models ASL$.

In order to obtain completeness, we should restrict to constructors with complete inference rules like D_σ. Nevertheless, if the inference rules for the constructors

are considered satisfactory, as is the case for S-constructors, we have a satisfactory inference system for $K + ASL \models ASL$ and, therefore, we also have a satisfactory inference systems for K-modular entailments in $K + ASL \models K + ASL$.

Our specification language lacks, however, an explicit way of forcing modular entailments during the development of software. Restricting to K-modular entailments, we can intermix some constructors with our specification in order to implicitly impose modular development and programming. Nevertheless, explicit use of Π-abstraction as proposed in [SST 90] would make specifications clearer.

7.5 Structure and theorem proving

Theorem proving has already been treated in chapter 3; however, some aspects related to structured specifications and, in particular, to specifications with a modular implementation must be added.

In chapter 3, proving theorems from structured specifications was considered a more complicated task than proving theorems from presentations or flat specifications. On the other hand, as it has been shown in [SB 83], structure far from complicating can ease theorem proving over specifications.

Taking advantage of the locality of the symbols used in a purported theorem, a smart theorem prover can find a small subspecification where the theorem can actually be proven. Of course, in the worst case no advantage is gained from the structure but in many cases a lot of effort can be saved, particularly if the underlying inference system is hard to use.

Similar considerations apply to proving reachability constraints and whole subspecifications. In these two cases, however, locality is guaranteed.

But perhaps the most striking result relating structure to theorem proving is that of O.Schoett about proving theorems from abstracted specifications which are implemented in a modular fashion.

7.5.1 Modular programming and data abstraction

In chapter 5 we studied in detail how to prove entailment w.r.t. behaviourally abstracted specifications, or in general, abstracted specifications.

$$SP2 \models T_o D_o SP1$$

Difficulties arise because the proof rule for D_σ is not M-complete and also because the number of observations is usually infinite. The first problem, although more fundamental, disappears when considering well-behaved observational abstractions, while the second one results harder to tackle.

Sound rules for the different SBO's in *ASL* never posed a problem. In particular for D_σ and T_σ we have sound and complete rules which imply we also have a sound and complete rule for abstraction

$$\frac{SP \models \varphi}{T_o D_o SP \models \varphi} \quad \varphi \in Obs_{\downarrow_o}$$

and therefore we feel confident when proving theorems such as $\xi(T_o D_o SP) \models \varphi$ or proving correctness of entailments such as $\xi(T_o D_o SP) \models SP1$. Nevertheless, the fact that only observations $Obs_{\downarrow_o}$ can be inferred by the rule and that there are infinitely many such observations, make such proofs very tedious.

Consider for example an abstracted specification of stacks within a bigger specification, *e.g.* the specification of a compiler $Comp(\alpha_{\equiv_\Pi} Stack)$. Every time we want to prove a simple property of the compiler involving the use of stacks, we have to write the whole proof using directly the observable properties of stacks. In some cases a single proof can require the use of all the observations and therefore drive us to the use of induction.

On the other hand, in a modular implementation of $Comp(\alpha_{\equiv_\Pi} Stack)$ a program (constructor) implementing *Comp* will *use* stacks but in the course of its execution it may never discriminate between two stacks which are behaviourally indistinguishable. Considering an execution of a program to be a theorem relating the input to the output, executions of the compiler can be seen as observable theorems φ over the signature of $Comp(\alpha_{\equiv_\Pi} Stack)$, hence we may expect

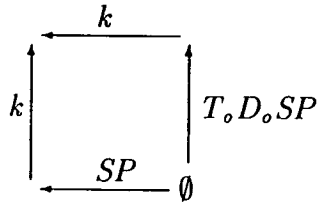
$$Comp(\alpha_{\equiv_\Pi} Stack) \models \varphi \text{ iff } Comp(Stack) \models \varphi$$

In consequence *the abstraction on stacks can be ignored when proving theorems about the compiler.*

In [Sch 87] an elegant formulation of this phenomenon is presented and a notion of *stability* is defined over programming languages characterizing when behavioural abstraction can be ignored in the course of proving properties of modular specifications. Now, the problem and its solution are revisited in our language of arrows.

Let us first define a stable constructor w.r.t. an abstracted specification as a constructor which cancels the effect of the abstraction.

Definition 7.5.1 *A constructor k is* **stable** *w.r.t. a specification SP and an abstraction operation T_oD_o iff the following diagram commutes, i.e. $k(SP) = k(T_oD_oSP)$.*



If k is stable w.r.t. SP and T_oD_o we guarantee that for all sentences φ

$$k(SP) \models \varphi \text{ iff } k(T_oD_oSP) \models \varphi$$

actually this notion of stability is stronger than that in [Sch 87]. Theorem 7.5.3 below recovers Schoett's notion of stability and shows their relationship.

Moreover, for any modular entailment $k(k2) \models P(T_o D_o SP)$ it follows that:

$k(k2) \models k(T_o D_o SP)$	by definition of modular entailment
$\models k(SP)$	by definition of stable constructor
$\models P(SP)$	by definition of modular entailment

hence, for all sentences φ

$$P(SP) \models \varphi \implies k(k2) \models \varphi$$

As in the case of a compiler using stacks, stability of k allows us to replace $T_o D_o SP$ by SP when proving theorems from $k(T_o D_o SP)$. In other words, we can assume that the implementation of stacks satisfies its specification when, in fact, it is only behavioural equivalent to an algebra satisfying it.

The main problem with this definition of stability is to determine when a constructor is stable.

In general, constructors are not stable. Consider for instance the case where k is the identity; requiring $k(SP) = k(T_o D_o SP)$ means that abstraction does not abstract at all, $SP = T_o D_o SP$. Although a characterization of stable constructor does not seem easy to obtain, a sufficient condition is available.

Fact 7.5.2 *Constructors of the form D_o; k are stable w.r.t. any specification SP and abstraction operation $T_o D_o$.*

Proof By definition of D_σ and T_σ, it is clear that for all morphisms σ,

$$D_\sigma T_\sigma D_\sigma SP = D_\sigma SP$$

hence $k(D_o T_o D_o SP) = k(D_o SP)$. □

Constructors of this form require k to "work" on the abstracted signature, but this change may be too demanding for the language of constructors. So, the main question is: Do constructors know how to work on abstracted signatures?

In a case such as **Beh(EQ)** and *FQRD*-constructors, the answer is *no*. Equations have only been defined for standard algebraic signatures, and sentences over an abstracted signature are only the observable sentences. Therefore, if k defines a `2-pop` operation on $D_o Stack$, an axiom like

```
∀ s:stack. 2-pop(s)=pop(pop(s))
```

is not allowed.

Changing the institution **Beh(EQ)**, by extending the set of sentences over an abstracted signature to include equalities between non-visible terms and a notion of satisfaction for them, might solve this problem. However, in a more realistic example where the constructors are programs, we are not concerned with equalities. What we expect is to be able to use the original functions such as `pop` and to ensure, by an encapsulation mechanism, that the constructor cannot gain access to any information about the nature of the non-observable values. For example, stable constructors should be able to use `pop`, store non-observable values and make copies of them, but it should not be able to check equality between non-observable values.

Technically this can be interpreted as follows: *The encapsulation mechanism is a constructor implementing D_o, for an abstraction morphism $o : \Pi \rightarrow \Sigma$.* Instructions in a constructor k on top of D_o when executed amount to sentences in $Sen(\Pi)$, that is, observations in $Sen(\Sigma)$.

Work in [Sch 87] looks at the problem and results from a different point of view since abstraction is, there, an atomic SBO.

Stable constructor do not need to have the form $D_o; k$. Intuitively, the hiding step D_o can be delayed in the implementation, *e.g.* a stable constructor can have the form $k1; D_{o'}; k2$ where $D_{o'}$ is intended to hide the same things as D_o. The hiding step can be shifted along the whole constructor to give a stable constructor $k3; D_{o''}$.

The condition for such *delaying* to be sound is that $k3$ (or $k1$ in the first case) preserves behavioural equivalence; *i.e.* for every two models A and B such that $A|_o = B|_o$, it holds that $k3(A)|_{o''} = k3(B)|_{o''}$, which is essentially the definition of stability in [Sch 87].

Theorem 7.5.3 *Consider a constructor k' defined by a function $f_{k'} : |Mod(\Sigma)| \to |Mod(\Sigma')|$ and abstraction morphisms $o : \Pi \to \Sigma$ and $o' : \Pi' \to \Sigma'$.*

$$\begin{array}{ccc} \Pi' & \xleftarrow{D_{o'}} & \Sigma' \\ & & \uparrow{\scriptstyle k'} \\ \Pi & \xleftarrow{D_o} & \Sigma \end{array}$$

If for all Σ-models A and B, $A|_o = B|_o \Rightarrow f_{k'}(A)|_{o'} = f_{k'}(B)|_{o'}$ then $k'; D_{o'}$ is stable w.r.t. any Σ-specification and $T_o D_o$.

Proof Let k be a constructor defined by a function $f_k : |Mod(\Pi)| \to |Mod(\Pi')|$ such that for all $A \in Mod(\Sigma)$

$$f_k(A|_o) = f_{k'}(A)|_{o'}$$

Since every model over Π is the o-reduct of some model over Σ, f_k is totally defined over $Mod(\Pi)$. And since $f_{k'}; \_|_{o'}$ delivers the same result for equivalent models, f_k is well-defined.

By definition of k the diagram commutes.

$$\begin{array}{ccc} \Pi' & \xleftarrow{D_{o'}} & \Sigma' \\ {\scriptstyle k}\uparrow & & \uparrow{\scriptstyle k'} \\ \Pi & \xleftarrow{D_o} & \Sigma \end{array}$$

By fact 7.5.2, $D_o; k$ is stable w.r.t. any specification SP and abstraction operation $T_o D_o$, hence

$$D_{o'}(k'(SP)) = k(D_o(SP)) = k(D_o(T_o D_o SP) = D_{o'}(k'(T_o D_o SP)$$

□

Using constructors of the form $k'; D_{o'}$ has two benefits. First, constructors and specifications only need to talk about sentences and signatures in the original institution, and that may be necessary if we do not have a syntax for sentences and signatures in the abstracted institution, *e.g.* infinitary signatures for behaviours in [MG 85]. In second place, $D_{o'}$ should be implemented by an encapsulation mechanism, but since it is at the "top" of the implementation, we can just ignore it.

Moreover, if abstraction is an atomic operation, as in [Sch 87], only the second presentation makes sense. But, if we have notions such as abstracted signatures, the first presentation considering D_o to be an encapsulation mechanism in the language of constructors is more concise.

Chapter 8

Conclusions

8.1 Main conclusions

> *"This thesis seeks a formal system for proving implementation steps to be correct in ASL and related languages ..."* (page 11 of the introduction)

The main conclusion is the feasibility of proving specification entailments in $ASL(\mathbf{I})$ for a given institution $\mathbf{I}$.

The results in this thesis can be summarized by a classification of specification entailments depending on whether they enjoy a complete inference system, a *satisfactory* inference system or just a sound inference system.

Assuming that we work in an institution whose consequence relation comes with a sound and complete inference system $\vdash^{\mathbf{I}}$, entailment between specifications has a complete inference system in the case of

$$DATU \models ATU$$

and also

$$DATU \models DATU^*$$

as discussed in chapter 4, where specifications in $DATU^*$ are those in $DATU$ using D_σ only in the context of persistent and independent hidden enrichments.

Since abstraction α is introduced in chapter 5 as a particular case of $T_\sigma D_\sigma$, we can also say that $\alpha + DATU \models DATU^*$ enjoys a complete inference system. Moreover, if we restrict the use of abstraction to an observational and well-behaved institution, $\alpha + DATU \models \alpha + DATU^*$ also has a complete inference system.

Finally, if our institution **I** treats constraints as ordinary sentences and the inference system $\vdash^{\mathbf{I}}$ is still sound and complete, we conclude that $\alpha + DMATU \models \alpha + DMATU^*$ (written $\alpha + ASL \models \alpha + ASL^*$) has a complete inference system.

A satisfactory inference system is an informal notion intended to capture incomplete inference systems which appear to be *good enough* in the great majority of the cases. In practice, such inference systems can be considered satisfactory for formal software development.

In an institution with reachability constraints such as **EQ** we have that:

- Considering structural induction as a satisfactory solution for inferring the inductive consequences of a presentation, $DMATU \models DATU^*$ has a satisfactory inference system.

- Considering set theory to be satisfactory for reasoning about sets and some additional assumptions, $ASL \models ASL^*$ also has a satisfactory inference system.

  Therefore, restricting the use of abstraction to an observational and well-behaved abstracted institution w.r.t. **EQ**, $\alpha + ASL \models \alpha + ASL^*$ has a satisfactory inference system as well.

- Considering we have satisfactory means for checking consistency, the $\exists^2$-strategy provides a satisfactory inference system for $DATU \models DATU$. And, in conjunction with the previous considerations, there are satisfactory inference systems for $ASL \models DATU$, $ASL \models ASL$ and $\alpha + ASL \models \alpha + ASL$.

If S-constructors $\langle P, \Lambda, \langle \iota, \sigma \rangle \rangle$ are considered, we can consider Λ and the pair $\langle \iota, \sigma \rangle$ to give a satisfactory account of the properties of the program P. Then, we have a satisfactory inference system for $k{+}ASL \models ASL$. Moreover, if specification entailments are restricted to K-modular entailments then

$$k{+}ASL \models k{+}ASL$$

also has a satisfactory inference system.

These results can be summarized in the following global conclusion: *In order to make verification acceptable we must choose an institution with a good inference system (including inference of constraints if they are to be used) specify with care the hidden parts of specification so that they are persistent and independent, choose an abstraction notion related to an observational and well-behaved institution, produce S-constructors with an appropriate set of properties and use modular entailments. Moreover, the prover must be prepared to complete some of the specifications during the proof of entailment so that problems caused by a lack of independence, or even a lack of persistency, can be solved.*

8.2 Problems and solutions

> *"In the process, we use ASL as a tool to investigate the verification of implementations in general ..."*

Although the main goal of the thesis has been solving specification entailments in *ASL*, the structure of the thesis and most of the work is dedicated to specific verification problems which arise in *ASL* but also in most other specification languages. In the following a short summary is given.

1. *The main verification problems arise in consequent specifications with hidden parts, behaviourally abstracted specifications, specifications with reachability*

constraints and partially implemented specifications (specifications with constructors).

This is the conclusion of section 3.5 stemming from the characterization, in theorem 3.3.12, of the inference rules available for each SBO. From a technical point of view, all these verification problems arise because the inference rule for D_σ is not M-complete (see counterexample in figure 3–2) and the rules for $M_{\langle\sigma,\iota\rangle}$ and $\langle P, \Lambda\rangle$ are not complete (see sections 3.3.2 and 3.4).

2. *There is a complete inference system for reachability constraints of the form* $\ll G, F, S \gg$.

   Although reachability and other kinds of constraints have been used in algebraic specification for a long time, no attempt has been made to infer reachability constraints from other constraints or from specifications. In chapter 6 inference systems for single constraints (Definition 6.3.4), sets of constraints (Definition 6.3.10) and mixed sets of equations and constraints (in section 6.3.2) are presented; the first (Theorem 6.3.5) and the second under some restrictions (Theorem 6.3.12) are complete.

3. *"Proving all observable theorems to hold" is a sound strategy for proving correctness w.r.t. abstracted specifications, if we use an observational and well-behaved notion of abstraction* (Theorem 5.3.3).

   This conclusion is a solid argument for preferring certain notions of abstraction rather than others. Previous work on behavioural abstraction led to a multiplicity of "slightly different" definitions which can now be classified.

   More striking is the fact that context induction [Hen 89] and many *ad hoc* proofs happen to be sound with respect to the standard notion of behaviour because they restrict the models of a specification to be reachable algebras,

hence restricting to the domain of an observational and well-behaved institution **BehR(EQ)** (Theorems 5.3.11 amd 5.3.13).

4. *If the hidden parts of a specification are specified as persistent and independent enrichments, correctness proofs become very much simpler.*

   This conclusion follows from theorems 4.4.1, 4.4.3, 4.5.1 and 4.5.3 in chapter 4 where persistency and independence are proven sufficient and necessary for the soundness and completeness of the inheriting strategy.

   In the case of hiding auxiliary functions, persistency and independence of the hidden enrichments are indeed very sensible and surely improve the clarity of structured specifications.

5. *Treating behavioural abstraction as the hiding of equality and/or generators for non-observable sorts is a sound strategy but is not always applicable.*

   Since abstraction can be understood as $T_\sigma D_\sigma$ in the appropriate institution, abstraction can be treated as the hiding of the equality symbols and/or generator functions. This strategy, if successful, requires the prover to define the "abstract" equality and/or the generators in the implementation (antecedent specification) before completing the proof, similarly as is generally done in verification w.r.t. abstract model specifications using an abstraction function (see example in 6.4.3).

6. *In order to treat subspecifications as single requirements some rules are needed for pulling whole subspecifications out of antecedent and consequent specifications.*

   Most of the times common subspecification, *e.g.* actual parameters, are repeated in the antecedent and the consequent of a purported entailment. However, a subspecification in the consequent cannot be proven to follow

from the same subspecification in the antecedent unless both subspecifications can be *pulled to the top* and are *in the same form* on both sides of the entailment (see section 7.3.1).

This conclusion generalizes common requirements in specification languages saying that parameters must be preserved/protected.

If we take a common approach to algebraic specification such as the initial approach, we can find that all these problems/solutions also apply. In the initial approach parameterized specifications are required to denote persistent functors, which means that subspecifications can be trivially pulled out of parameterized specification (if they are parameter specifications) and hidden enrichments are trivially persistent and independent (if they are parameterized specifications). Functor persistency also guarantees consistency and sufficient completeness of parameter specifications w.r.t. to their actual parameters, hence redundancy of certain functions and sorts can be explicitly stated. Moreover, initial semantics guarantees models to be reachable algebras and therefore observationality and well-behaviour follow for the standard notion of behavioural abstraction.

In general, we have to study each of the *verification problems* mentioned above in the context of any specification language we use. Hopefully, some potential problems do not arise due to the particular definition of the specification language and/or the institution used.

8.3 Negative results and trade-offs

> *"... we shall ... exhibit trade-offs between sensible verification methods and the expressive power of specifications and the flexibility of implementations".*

There are a number of negative results and trade-offs between flexibility and satisfactory verification. In the following we enumerate the most significant results.

1. *An inference system for constraints has to be developed for each institution.*

   Due to the definition of institution and *ASL*, there is no relation between morphisms in the category of models and the satisfaction relation. This results in a SBO $M_{\langle\sigma,\iota\rangle}$ without any sensible institution-independent verification treatment. Hence, the user proposing the use of an institution should develop an inference system for constraints in that institution.

2. *Persistency of an enrichment is undecidable.*

   Although persistency of hidden enrichments plays a key role in most of this thesis, there is no effective procedure for checking persistency of enrichments w.r.t. basic specifications. This is also a well-known problem in the initial approach where some proof theoretical conditions have been established [Gan 83, Pad 80, Pad 85].

3. *Lack of independence usually requires prover intervention.*

   Almost always the lack of independence of a hidden enrichment requires the prover to complete the specification with some extra axioms. This prevents proofs from being automatic when the generators of a visible sort are hidden or abstraction is treated as hiding.

4. *There is no known observational and well-behaved institution for standard behavioural abstraction.*

   Non-reachable values in observable sorts prevent most institutions from being observational. The only available observational institution "modeling" standard behavioural abstraction, **BehO(EQ)**, is not well-behaved.

5. *No inference system is available for inferring reachability constraints from sets of reachability constraints and equations.*

   Despite the satisfactory performance of $\Vdash$ (Definition 6.3.10), we have no complete inference system for inferring a reachability constraint from a finite set of reachability constraints. Moreover, if reachability constraints are intermixed with other kind of sentences the satisfactory performance of $\Vdash_\Phi$ (integrating $\Vdash$ and $\vdash^{\mathbf{EQ}}$) depends on a long list of assumptions.

6. *Although $\alpha + DATU \models ATU$ has a complete inference system, proofs may be very hard.*

   In this case, despite the availability of a complete inference system for proving theorems from specifications in *DATU*, such proofs may be terribly tedious in practice since only observable premises are allowed.

   Work in [Sch 87] (see section 7.5.1) solves this problem in the case of modular entailments, however, in general the problem persists.

8.4 Future developments

The results of this thesis cover the verification of specification entailments in the whole of *ASL* including mechanisms for abstraction and parameterization. Future developments should address the application of these results to real specification languages and every day verification.

In particular, the semantics of a specification language can be designed so that hiding is only used in the context of persistent and independent hidden enrichments, explicit means for enforcing modular development are made available, etc.

At the institution level, some common institutions can be fixed and an inference system for their constraints developed, like the inference system for reachability constraints in chapter 6.

The actual inference rules in $\vdash_M$ and in the appropriate $\vdash^{\mathbf{I}}$ can be incorporated into a software development environment so that specification entailment proofs are electronically supported. In this context, attention can be drawn to the support of some verification strategies such as the inheriting strategy.

In specification languages such as Extended ML [ST 86, ST 89], this environment can be used for proving the correctness of development steps and also for producing well-formed specifications. In particular, interface matching $SP2 : SP1$ between structures and signatures in Extended ML amounts to specification entailment $SP2 \models SP1$ as discussed in this thesis.

On a more theoretical side, results in this thesis can be used for analyzing other specification languages from the proof-theoretic point of view. Analogously to *ASL* which has been used for analyzing the semantics of other specification languages, *e.g.* by distinguishing the institutional from the institution-independent

aspects, we can use our collection of verification problems, inference systems and specification properties in order to classify different approaches to formal specification.

In the end, users will become aware of the trade-off between flexibility and verification techniques and, eventually, they should be able to identify which approach to formal program development is best suited for them.

Index

List of examples

The inference system $\vdash_M$

1. Basic proof rule:

$$\frac{\Phi 2 \vdash^{\mathrm{I}} \Phi 1}{A_{\Phi 2} SP \vdash_M A_{\Phi 1} \epsilon_{Sig[SP]}}$$

2. Pulling sentences (axioms) out of the consequent:

$A_\Phi(SP1\ U\ SP2) \vdash_M (A_\Phi SP1)\ U\ SP2$ $\qquad$ $A_{\sigma(\Phi)} T_\sigma SP \vdash_M T_\sigma A_\Phi SP$

$A_\Phi(SP1\ U\ SP2) \vdash_M SP1\ U\ (A_\Phi SP2)$ $\qquad$ $A_{\Phi 1} A_{\Phi 2} SP \vdash_M A_{\Phi 2} A_{\Phi 1} SP$

3. Pulling subspecifications out of the consequent:

$A_\Phi SP\ U\ SSP \vdash_M A_\Phi(SP\ U\ SSP)$

$M_{(\sigma,\iota)} SP\ U\ SSP \vdash_M M_{(\sigma,\iota)}(SP\ U\ SSP)$

$(SP1\ U\ SP2)\ U\ SSP \vdash_M SP1\ U\ (SP2\ U\ SSP)$

$(SP1\ U\ SP2)\ U\ SSP \vdash_M (SP1\ U\ SSP)\ U\ SP2$

$T_\sigma SP\ U\ T_\sigma SSP \vdash_M T_\sigma(SP\ U\ SSP)$

$D_\iota SP\ U\ SSP \vdash_M D_\iota(SP\ U\ T_\iota SSP)$

4. Pulling sentences (theorems) out of the antecedent:

$SP1\ U\ (A_\Phi SP2) \vdash_M A_\Phi(SP1\ U\ (A_\Phi SP2))$ $\qquad$ $T_\sigma A_\Phi SP \vdash_M A_{\sigma(\Phi)} T_\sigma A_\Phi SP$

$(A_\Phi SP1)\ U\ SP2 \vdash_M A_\Phi((A_\Phi SP1)\ U\ SP2)$ $\qquad$ $D_\sigma A_\Phi SP \vdash_M A_{\sigma^{-1}(\Phi)} D_\sigma A_\Phi SP$

$\langle P, \Lambda, \langle \iota, \sigma \rangle\rangle A_\Phi SP \vdash_M A_{\{\varphi\}} \langle P, \Lambda, \langle \iota, \sigma \rangle\rangle SP$ $\qquad$ provided $(\Phi, \varphi) \in \Lambda$

5. Pulling subspecifications out of the antecedent:

$A_\Phi(SP\ U\ SSP) \vdash_M A_\Phi(SP\ U\ SSP)\ U\ SSP$

$M_{\langle\sigma,\iota\rangle}(SP\ U\ SSP) \vdash_M M_{\langle\sigma,\iota\rangle}(SP\ U\ SSP)\ U\ SSP$

$SP1\ U\ (SP2\ U\ SSP) \vdash_M (SP1\ U\ (SP2\ U\ SSP))\ U\ SSP$

$(SP1\ U\ SSP)\ U\ SP2 \vdash_M ((SP1\ U\ SSP)\ U\ SP2)\ U\ SSP)$

$T_\sigma(SP\ U\ SSP) \vdash_M T_\sigma(SP\ U\ SSP)\ U\ T_\sigma SSP$

$D_\iota(SP\ U\ T_\iota SSP) \vdash_M D_\iota(SP\ U\ T_\iota SSP)\ U\ SSP$

$\langle P, \Lambda, \langle\iota,\sigma\rangle\rangle(SP\ U\ T_\iota SSP) \vdash_M (\langle P, \Lambda, \langle\iota,\sigma\rangle\rangle(SP\ U\ T_\iota SSP))\ U\ T_\sigma SSP$

6. Permuting D (equality $=_M$ means logical derivability $\vdash_M$ in both directions):

$A_\Phi D_\sigma SP =_M D_\sigma A_{\sigma(\Phi)} SP$

$SP1\ U\ D_\sigma SP2 =_M D_\sigma(T_\sigma SP1\ U\ SP2)$

$D_\sigma SP1\ U\ SP2 =_M D_\sigma(SP1\ U\ T_\sigma SP2)$

$T_\sigma D_{\sigma 1} SP =_M D_{\sigma 1'} T_{\sigma'} SP$ provided $\sigma, \sigma 1, \sigma'$ and $\sigma 1'$ form a pushout diagram as in proposition 4.2.1

7. Absorbing laws:

$T_\sigma \epsilon_\Psi =_M \epsilon_\Psi$ $\quad \epsilon_\Sigma\ U\ \epsilon_\Sigma =_M \epsilon_\Sigma$ $\quad A_\emptyset \epsilon_\Sigma =_M \epsilon_\Sigma$

$A_{\Phi 1} A_{\Phi 2} SP =_M A_{\Phi 1 \cup \Phi 2} SP$ $\quad D_\sigma D_{\sigma'} SP =_M D_{\sigma;\sigma'} SP$

8. Cut rule:

$$\frac{SP3 \vdash_M SP2 \quad SP2 \vdash_M SP1}{SP3 \vdash_M SP1}$$

9. Simplification rule:

$$\frac{SP2 \vdash_M SP1}{\xi SP2 \vdash_M \xi SP1}$$

In first order logic.

1. Structural induction

$$M_{\langle G,F,S\rangle} A_{\{\varphi\}} SP \vdash_M A_{\{\varphi'\}} M_{\langle G,F,S\rangle} A_{\{\varphi\}} SP$$

where φ is

$$\bigwedge_{s\in G} \bigwedge_{op\in F_s} \forall \overline{x} : S, \overline{y} : G.\ (\overline{Q}(\overline{y}) \Rightarrow Q_s(op(\overline{xy})))$$

φ' is $\bigwedge_{s\in G} \forall y : s.Q_s(y)$, Q is a G-indexed set of predicates, $\overline{x}$ a S-sorted sequence of variables, $\overline{y}$ a G-sorted sequence of variables, $\overline{xy}$ is the concatenation of $\overline{x}$ and $\overline{y}$, F_s those operations in F delivering a value of sort s, and $\overline{Q}$ the conjunction of the appropriate predicates in Q applied to a given G-sorted sequence of variables.

In EQ.

1. Pulling sentences through FQR-constructors:

$Q_{eq} SP \vdash_M A_{eq} Q_{eq} SP$
$Q_{eq} A_\Phi SP \vdash_M A_\Phi Q_{eq} A_\Phi SP$
$R_G A_\Phi SP \vdash_M A_\Phi R_G A_\Phi SP$
$F^{eq}_\sigma SP \vdash_M A_{eq} F^{eq}_\sigma SP$
$F^{eq}_\sigma A_\Phi SP \vdash_M A_{\sigma(\Phi)} F^{eq}_\sigma A_\Phi SP$

provided $F^{eq}_\sigma SP$ is sufficiently complete and consistent.

2. Pulling reachability constraints through FQR-constructors:

$Q_{eq} M_{\langle G,F,S\rangle} SP \vdash_M M_{\langle G,F,S\rangle} Q_{eq} M_{\langle G,F,S\rangle} SP$
$R_G SP \vdash_M M_{\langle G,\Omega,S\setminus G\rangle} R_G SP$ provided $Sig[SP] = \langle S, \Omega\rangle$
$R_{G'} M_{\langle G,F,S\rangle} SP \vdash_M M_{\langle G,F,S\rangle} R_{G'} M_{\langle G,F,S\rangle} SP$ provided $G' \cap S = \emptyset$
and $\ll G', \Omega', S' \setminus G' \gg$ for $\langle S', \Omega'\rangle = Sig[SP]$ is $\emptyset$-free

$F^{eq}_{\sigma} M_{\langle G,F,S\rangle} SP \vdash_M M_{\langle \sigma(G),\Omega',S'\rangle} F^{eq}_{\sigma} M_{\langle G,F,S\rangle} SP$ provided $\uparrow\sigma = \langle S',\Omega'\rangle$

$F^{eq}_{\sigma} SP \vdash_M M_{\langle G,\Omega',\sigma(S)\rangle} F^{eq}_{\sigma} SP$ provided $\sigma : \langle S,\Omega\rangle \to \langle \sigma(S) \uplus G,\Omega'\rangle$

3. Pulling subspecifications through FQR-constructors: Given a signature morphism $\iota : \langle S,\Omega\rangle \to \langle S',\Omega'\rangle$,

$R_{\{s\}}(SP\ U\ T_\iota SSP) \vdash_M R_{\{s\}}(SP\ U\ T_\iota SSP)\ U\ T_\iota SSP$

provided $s \in (\Omega' \setminus \iota(\Omega))$ or $Mod[SSP]$ is closed under subalgebras

$Q_{eq}(SP\ U\ T_\iota SSP) \vdash_M Q_{eq}(SP\ U\ T_\iota SSP)\ U\ T_\iota SSP$

provided ι forgets eq or $Mod[SSP]$ is closed under quotients

$F^{eq}_{\sigma}(SP\ U\ SSP) \vdash_M F^{eq}_{\sigma}(SP\ U\ SSP)\ U\ T_\sigma SSP$

provided $(Free^{eq}_{\sigma} A)|_\sigma = A$ for all $A \in Mod[SSP]$

Bibliography

[Bar 77] J. Barwise (ed.). *Handbook of Mathematical Logic.* North Holland (1977).

[Ber 87] G. Bernot: Good functors ... are those preserving philosophy. *Proc. Summer Conf. on Category Theory and Computer Science*, Edinburgh. Springer LNCS 283, 182–195 (1987).

[Bid 88] M. Bidoit: The stratified loose approach: a generalization of initial and loose semantics. *Recent Trends in Data Type Specification, Selected Papers from the 5th Workshop on Specification of Abstract Data Types*, Gullane, Scotland. Springer LNCS 332, 1–22 (1988).

[Bid 89] M. Bidoit: PLUSS, un langage pour le développement de spécifications algébriques modulaires. Thèse d'Etat, Université Paris-Sud, Orsay (1989).

[Bre 89] R. Breu: A normal form for structured algebraic specifications. Report MIP-8917, Universität Passau (1989).

[Bur 69] R. M. Burstall: Proving properties of programs by structural induction. *Computer Journal* 12, 41–48 (1969).

[Bur 87] R. M. Burstall: Inductively defined functions in functional programming languages. Report CSR-230-87, Dept. of Computer Science, Univ. of Edinburgh (1987).

[BBC 86] G. Bernot, M. Bidoit, C. Choppy: Abstract implementations and correctness proofs. *Proc. 3rd Symp. on Theoretical Aspects of Computer Science.* Springer LNCS 210, 236–251 (1986).

[BBTW 81] J.A. Bergstra, M. Broy, J.V. Tucker, M. Wirsing: On the power of algebraic specifications. *Proc. 1981 Symp. on Mathematical Foundations of Computer Science.* Springer LNCS 118, 193–204 (1981).

[BG 77] R.M. Burstall, J.A. Goguen: Putting theories together to make specifications. *Proc. 5th Intl. Joint Conf. on Artificial Intelligence*, Cambridge, Massachusetts, 1045–1058 (1977).

[BG 80] R.M. Burstall, J.A. Goguen: The semantics of CLEAR, a specification language. *Proc. Advanced Course on Abstract Software Specifications*, Copenhagen. Springer LNCS 86, 292–332 (1980).

[BGM 89] M. Bidoit, M.-C. Gaudel, A. Mauboussin: How to make algebraic specifications more understandable? An experiment with the PLUSS specification language. *Science of Computer Programming* 12, 1–38 (1989).

[BHK 86] J.A. Bergstra, J. Heering, R. Klint: Module algebra. Report CS:R8617, Centre voor Wiskunde en Informatica, Amsterdam (1986).

[BHK 89] J.A. Bergstra, J. Heering, R. Klint (eds.): *Algebraic Specification.* Addison-Wesley (1989).

[BHK 90] J.A. Bergstra, J. Heering, R. Klint: Module algebra. *Journal of the Assoc. for Computing Machinery* 37(2), 335–372 (1990).

[BJ 80] G. Boolos, R. Jeffrey: *Computability and Logic.* Cambridge University Press, second edition (1980).

[BM 79] R.S. Boyer, J.S. Moore: *A Computational Logic.* Academic Press (1979).

[BM 88] R.S. Boyer, J.S. Moore: *A Computational Logic Handbook.* Academic Press (1988).

[BP 90] P. Byers, D. Pitt: Conservative extensions: A cautionary note. *EATCS Bulletin* 41, 196–201 (1990).

[BT 83] J.A. Bergstra, J.V. Tucker: Initial and final algebra semantics: two characterization theorems. *SIAM Journal on Computing* 12(2), 366–387 (1983).

[BT 87] J.A. Bergstra, J.V. Tucker: Algebraic specifications of computable and semicomputable data types. *Theoretical Computer Science* 50, 137–181 (1987).

[Cohn 66] P. Cohn. *Universal Algebra.* Harper & Row, New York (1966).

[DGS 91] R. Diaconescu, J.A. Goguen, P. Stefaneas: Logical support for modularization. *Draft report*, Programming Research Group, Oxford University (1991).

[Ehr 81] H.-D. Ehrich: On realization and implementation. *Proc. 10th Symp. on Mathematical Foundations of Computer Science.* Springer LNCS 118, 271–280 (1981).

[EKMP 82] H. Ehrig, H.-J. Kreowski, B. Mahr, P. Padawitz: Algebraic implementation of abstract data types. *Theoretical Computer Science* 20, 209–263 (1982).

[EKTWW 84] H. Ehrig, H.-J. Kreowski, J. Thatcher, E. Wagner, J. Wright: Parameter passing in algebraic specification languages. *Theoretical Computer Science* 28, 45–81 (1984).

[EM 85] H. Ehrig, B. Mahr: *Fundamentals of Algebraic Specification 1. Equations and Initial Semantics.* EATCS Monographs on Theoretical Computer Science, Vol. 6. Springer (1985).

[EM 90] H. Ehrig, B. Mahr: *Fundamentals of Algebraic Specification 2. Module Specifications and Constraints.* EATCS Monographs on Theoretical Computer Science, Vol. 21. Springer (1990).

[EWT 83] H. Ehrig, E.G. Wagner, J.W. Thatcher: Algebraic specifications with generating constraints. *Proc. 10th Intl. Colloq. on Automata, Languages and Programming.* Springer LNCS 154, 188–202 (1983).

[Far 89] J. Farrés-Casals: Proving correctness of constructor implementations. *Proc. 1989 Symp. on Mathematical Foundations of Computer Science.* Springer LNCS 379, 225–235 (1989).

[Far 90] J. Farrés-Casals: Proving correctness w.r.t. specifications with hidden parts. *Proc. 1990 Second International Conf. of Algebraic and Logic Programming.* Springer LNCS 463, 25–39 (1990).

[FGJM 85] K. Futatsugi, J.A. Goguen, J.-P. Jouannaud, J. Meseguer: Principles of OBJ2. *Proc. 12th ACM Symp. on Principles of Programming Languages*, New Orleans, 52–66 (1985).

[Gan 83] H. Ganzinger: Parameterized specifications: parameter passing and implementation with respect to observability. *ACM Trans. on Programming Languages and Systems* 5(3), 318–354 (1983).

[Gau 78] M.-C. Gaudel. Spécifications incomplètes mais suffisantes de la représentation des types abstraits. Laboria Report 320 (1978).

[Gau 84] M.-C. Gaudel: A first introduction to PLUSS. Technical Report, LRI, Université de Paris-Sud, Orsay (1984).

[Gog 90] J.A. Goguen: Types as theories. Unpublished draft (1990).

[Gut 75] J.V. Guttag: The Specification and Application to Programming of Abstract Data Types. Ph.D. thesis, Univ. of Toronto (1975).

[GB 80] J.A. Goguen, R.M. Burstall: CAT, a system for the structured elaboration of correct programs from structured specifications. Technical report CSL-118, Computer Science Laboratory, SRI International (1980).

[GB 84] J.A. Goguen, R.M. Burstall: Introducing institutions. *Proc. Logics of Programming Workshop*, Carnegie-Mellon. Springer LNCS 164, 221–256 (1984).

[GB 86] J.A. Goguen, R.M. Burstall: A study in the foundations of programming methodology: specifications, institutions, charters and parchments. *Proc. Workshop on Category Theory and Computer Programming*, Guildford. Springer LNCS 240, 313–333 (1986).

[GB 90] J.A. Goguen, R.M. Burstall: Institutions: abstract model theory for specifications and programming. Report ECS-LFCS-90-106, University of Edinburgh (1990).

[GG 88] S.J. Garland, J.V. Guttag: Inductive methods for reasoning about abstract data types. *In Proc. 5th ACM-SIGPLAN*, San Diego, 219–228 (1988).

[GGM 76] V. Giarratana, F. Gimona, U. Montanari: Observability concepts in abstract data type specification. *Proc. 1976 Symp. on Mathematical Foundations of Computer Science*, Gdansk. Springer LNCS 45, 567–578 (1976).

[GH 78] J.V. Guttag, J.J. Horning: The algebraic specification of abstract data types. *Acta Informatica* 10, 27–52 (1978).

[GH 86] J.V. Guttag, J.J. Horning: Report on the Larch shared language. *Science of Computer Programming* 6(2), 103–134 (1986).

[GHW 85] J. V. Guttag, J.J. Horning, J.M. Wing: Larch in five easy pieces. Digital Systems Research Centre, Palo Alto (1985).

[GM 81] J.A. Goguen, J. Meseguer: Completeness of many-sorted equational logic. *SIGPLAN Notices* 16(7) 24–32 (1981).

[GM 82] J.A. Goguen, J. Meseguer: Universal realization, persistent interconnection and implementation of abstract modules. *Proc. 9th Intl. Colloq. on Automata, Languages and Programming*, Aarhus. Springer LNCS 140, 265–281 (1982).

[GTW 76] J.A. Goguen, J.W. Thatcher, E.G. Wagner: An initial algebra approach to the specification, correctness and implementation of abstract data types. Report RC-6487, IBM T.J. Watson Research Center, Yorktown Heights (1976). Also in: *Current Trends in Programming Methodology, Vol. 4: Data Structuring* (R.T. Yeh, ed.). Prentice-Hall, 80–149 (1978).

[GTWW 75] J.A. Goguen, J.W. Thatcher, E.G. Wagner, J.B. Wright: An introduction to Categories, Algebraic Theories and Algebras. Report RC-5369, IBM T.J. Watson Research Center, Yorktown Heights (1975).

[Hen 88] R. Hennicker: Beobachtungsorientierte Spezifikationen. Ph.D. thesis, Universität Passau (1988).

[Hen 89] R. Hennicker: Observational implementations. *Proc. 6th Symp. on Theoretical Aspects of Computer Science*, Paderborn. Springer LNCS 349 (1989).

[Hoa 72] C.A.R. Hoare: Proofs of correctness of data representations. *Acta Informatica* 1, 271–281 (1972).

[HST 89a] R. Harper, D. Sannella, A. Tarlecki: Structure and representation in LF. *Proc. 2nd IEEE Symp. on Logic in Computer Science*, Asilomar, 226–237 (1989).

[HST 89b] R. Harper, D. Sannella, A. Tarlecki: Logic representation. *Proc. 3rd Summer Conf. on Category Theory and Computer Science*, Manchester. Springer LNCS 389, 250–272 (1989).

[Jon 80] C. Jones. *Software Development: A Rigorous Approach.* Prentice-Hall (1980).

[Jon 86] C. Jones. *Systematic Software Development Using VDM.* Prentice-Hall (1986).

[Kam 83] S. Kamin: Final data types and their specification. *ACM Transactions on Programming Languages and Systems*, 5(1), 97–123 (1983).

[Kei 77] H. J. Keisler: Fundamentals of model theory. *Handbook of Mathematical Logic.* North Holland (1977).

[KK 67] G. Kreisel, J.L. Krivine. *Elements of Mathematical Logic (Model Theory).* North Holland (1967).

[KM 87] D. Kapur, D.R. Musser: Proof by consistency. *Artificial Intelligence* 31(2), 125–157 (1987).

[Law 63] F. Lawvere: Functorial semantics of algebraic theories. *Prôc. National Academy of Science* 50, 869–872 (1963).

[Lin 87] Lin Huimin. Relative completeness in algebraic specifications. Report ECS-LFCS-87-43, University of Edinburgh (1987).

[Mac 71] S. MacLane. *Categories for the Working Mathematician.* Springer-Verlag (1971).

[Maj 77] M.E. Majster: Limits of the algebraic specifications of abstract data types. *ACM-SIGPLAN Notices* 12, 37–42 (1977).

[Men 71] E. Mendelson. *Introduction to Mathematical Logic.* Van Nostrand (1971). References to 3rd edition, by Wadsworth & Brooks (1987).

[Mes 89] J. Meseguer: General logics. *Proc. Logic Colloquium '87,* Granada. North Holland (1989).

[Mos 89] P. Mosses: Unified algebras and modules. *Proc. 16th ACM Symp. on Principles of Programming Languages,* 329–343 (1989).

[MG 85] J. Meseguer, J.A. Goguen: Initiality, induction and computability. In: *Algebraic Methods in Semantics* (M. Nivat and J. Reynolds, eds.). Cambridge Univ. Press, 459–540 (1985).

[MM 84] B. Mahr, J. Makowsky: Characterizing specification languages which admit initial semantics. *Theoretical Computer Science* 31, 49–59 (1984).

[MS 85] D.B. MacQueen, D.T. Sannella: Completeness of proof systems for equational specifications. *IEEE Trans. on Software Engineering* SE-11(5), 454–461 (1985).

[Nip 86] T. Nipkow: Non-deterministic data types: models and implementations. *Acta Informatica* 22, 629–661 (1986).

[Niv 87] P. Nivela. Semantica de comportamiento para especificaciones algebraicas. Ph.D. Thesis, Universitat Politècnica de Catalunya, Barcelona (1987).

[NO 88] P. Nivela, F. Orejas: Initial behaviour semantics for algebraic specifications. *Recent Trends in Data Type Specification, Selected Papers from the 5th Workshop on Specification of Abstract Data Types*, Gullane, Scotland. Springer LNCS 332, 184–207 (1988).

[Ore 83] F. Orejas: Characterizing composability of abstract implementations. *Proc. 1983 Intl. Conf. on Foundations of Computation Theory*, Borgholm, Sweden. Springer LNCS 158 (1983).

[Ore 87] F. Orejas: A characterization of passing compatibility for parameterized specifications. *Theoretical Computer Science* 51, 205–214 (1987).

[OSC 89] F. Orejas, V. Sacristan, S. Clérici: Development of algebraic specifications with constraints. *Proc Workshop on Categorical Methods in Computer Science with Aspects from Topology.* Springer LNCS 393, 102–123 (1989).

[Pad 80] P. Padawitz: New results on completeness and consistency of abstract data types. *Proc. 9th Symp. on Mathematical Foundations of Computer Science*, Rydzyna. Springer LNCS 88, 460–473 (1980).

[Pad 85] P. Padawitz: Parameter preserving data type specifications. *Proc. Joint Conf. on Theory and Practice of Software Development*, Berlin. Springer LNCS 186, 323–341 (1985).

[Poi 86] A. Poigné: Algebra categorically. *Proc. Intl. Workshop on Category Theory and Computer Programming*, Guildford 1985. Springer LNCS 240, 76–102 (1986).

[Poi 89] A. Poigné. Foundations are rich institutions, but institutions are poor foundations. *Proc Workshop on Categorical Methods in Computer Science with Aspects from Topology.* Springer LNCS 393, (1989).

[Rei 81] H. Reichel: Behavioural equivalence — a unifying concept for initial and final specification methods. *Proc. 3rd. Hungarian Comp. Sci. Conference*, 27–39 (1981).

[Rei 87] H. Reichel: *Initial Computability, Algebraic Specifications, and Partial Algebras.* Oxford Univ. Press (1987).

[San 86] D.T. Sannella: Formal specification of ML programs. Report ECS-LFCS-86-15, University of Edinburgh (1986).

[Sch 87] O. Schoett: Data Abstraction and the Correctness of Modular Programming. Ph.D. thesis; Report CST-42-87, Dept. of Computer Science, Univ. of Edinburgh (1987).

[Sch 90] O. Schoett: Behavioural correctness of data representations. *Science of Computer Programming* 14, 43–57 (1990).

[Sch 91] O. Schoett: An observational subset of first-order logic cannot specify the behaviour of a counter. *Proc. 8th Symp. on Theoretical Aspects of Computer Science*, Springer LNCS, 499–510 (1991).

[Sco 66] D.S. Scott: Logic with denumerably long formulas and finite strings of quantifiers. *Theory of models.* North-Holland, 329–341 (1966).

[Spi 85] M.Spivey: Understanding Z. Ph.D. thesis. University of Oxford (1985).

[SB 83] D.T. Sannella, R.M. Burstall: Structured theories in LCF. *Proc. 1983 Colloq. on Trees in Algebra and Programming*, L'Aquila. Springer LNCS 159, 377–391 (1983).

[SST 90] D.T. Sannella, S. Sokołowski, A. Tarlecki: Toward formal development of programs from algebraic specifications: parameterisation revisited. Report 6/90, Universität Bremen (1990).

[ST 84] D.T. Sannella, A. Tarlecki: Building specifications in an arbitrary institution. *Proc. of the Intl. Symp. on Semantics of Data Types*, Sophia-Antipolis. Springer LNCS 173, 337–356 (1984).

[ST 85] D.T. Sannella, A. Tarlecki: Program specification and development in Standard ML. *Proc. 12th ACM Symp. on Principles of Programming Languages*, New Orleans, 67–77 (1985).

[ST 86] D.T. Sannella, A. Tarlecki: Extended ML: an institution-independent framework for formal program development. *Proc.*

Workshop on Category Theory and Computer Programming, Guildford. Springer LNCS 240, 364–389 (1986).

[ST 87] D.T. Sannella, A. Tarlecki: On observational equivalence and algebraic specification. *Journal of Computer and System Sciences* 34, 150–178 (1987).

[ST 88a] D.T. Sannella, A. Tarlecki: Specifications in an arbitrary institution. *Information and Computation* 76, 165–210 (1988).

[ST 88b] D.T. Sannella, A. Tarlecki: Toward formal development of programs from algebraic specifications: implementations revisited. Extended abstract in: *Proc. Joint Conf. on Theory and Practice of Software Development*, Pisa. Springer LNCS 249, 96–110 (1987); full version in *Acta Informatica* 25, 233–281 (1988).

[ST 89] D.T. Sannella, A. Tarlecki: Toward formal development of ML programs: foundations and methodology. *Proc. Joint Conf. on Theory and Practice of Software Development*, Barcelona. Springer LNCS 352, 375–389 (1989).

[ST 92] D.T. Sannella, A. Tarlecki: *Foundations of Algebraic Specification and Formal Program Development.* Cambridge Univ. Press, to appear.

[SW 83] D.T. Sannella, M. Wirsing: A kernel language for algebraic specification and implementation. *Proc. 1983 Intl. Conf. on Foundations of Computation Theory*, Borgholm, Sweden. Springer LNCS 158, 413–427 (1983).

[Tar 86a] A. Tarlecki: Bits and pieces of the theory of institutions. *Proc. Workshop on Category Theory and Computer Programming*, Guildford. Springer LNCS 240, 334–363 (1986).

[Tar 86b] A. Tarlecki: On the existence of free models in abstract algebraic institutions. *Theoretical Computer Science* 37, 269–304 (1986).

[TWW 78] J.W. Thatcher, E.G. Wagner, J.B. Wright: Data type specification: parameterization and the power of specification techniques. *SIGACT 10th Annual Symp. on the Theory of Computing*, San Diego (1978). Also in: *ACM Trans. on Programming Languages and Systems* 4, 711–773 (1982).

[Wan 79] M. Wand: Final algebra semantics and data type extensions. *Journal of Computer and System Sciences* 19, 27–44 (1979).

[Wan 82] M. Wand: Specifications, models and implementations of data abstraction. *Theoretical Computer Science* 20, 2–32 (1982).

[Wir 86] M. Wirsing: Structured algebraic specifications: a kernel language. *Theoretical Computer Science* 42, 123–249 (1986).

[Wir 91] M. Wirsing: Proofs in structured specifications. Technical Report MIP-9008, Universität Passau (1991).

[WB 89] M. Wirsing, M. Broy: A modular framework for specification and implementation. *Proc. Joint Conf. on Theory and Practice of Software Development*, Barcelona. Springer LNCS 351, 42–73 (1989).

where

$$f_B\colon |B|_{shape} \times |B|_{suit} \to |B|_{suit}$$

is defined by the following table:

f_B	1	2	3
□	2	3	3
△	1	3	2

Let $i\colon |A| \to |B|$ be the S-sorted function such that

$$i_{shape} =_{\text{def}} \{\square \mapsto \triangle, \triangle \mapsto \square\} \quad \text{and} \quad i_{suit} =_{\text{def}} \{\clubsuit \mapsto 1, \heartsuit \mapsto 2, \spadesuit \mapsto 3\}.$$

This defines a Σ-homomorphism $i\colon A \to B$ which is a Σ-isomorphism, so $A \cong B$. □

Exercise 1.3.10

Show that a homomorphism is an isomorphism iff it is bijective. □

Exercise 1.3.11

Show that there is an injective homomorphism $h\colon A \to B$ iff A is isomorphic to a subalgebra of B. □

Example 1.3.12

Let $\Sigma = \langle S, \Omega \rangle$ be the signature

sorts s
opns $a\colon s$
$f\colon s \to s$

and define Σ-algebras A and B by

$$|A|_s =_{\text{def}} Nat \text{ (the natural numbers)}, \quad a_A =_{\text{def}} 0 \in |A|_s,$$

$$f_A\colon |A|_s \to |A|_s =_{\text{def}} \{n \mapsto n+1 \mid n \in Nat\},$$

$$|B|_s =_{\text{def}} \{n \in Nat \mid \text{the Turing machine with Gödel number } n \text{ halts on all inputs}\},$$

$$a_B =_{\text{def}} \text{the smallest } n \in Nat \text{ such that the TM with Gödel number } n \text{ halts on all inputs},$$

and

$f_B\colon |B|_s \to |B|_s$ is defined by $f_B(n) =$ the smallest $m > n$ such that the TM with Gödel numb

Let $i\colon |A| \to |B|$ be the S-sorted function such that

$$i_s(n) =_{\text{def}} \text{the } (n+1)^{\text{st}} \text{ smallest element of } |B|_s$$

for all $n \in |A|_s$. The function i_s is well-defined since $|B|_s$ is infinite. This defines a Σ-homomorphism $i\colon A \to B$ which is an isomorphism.

Although $A \cong B$, the Σ-algebras A and B are not "the same" from the point of view of computability: everything in A is computable, in contrast to B ($|B|_s$ is not recursively enumerable and f_B is not computable). Isomorphisms capture *structural* similarity, ignoring what the values in the carriers are and what the functions actually compute. This example shows that, for some purposes, properties stronger than structural similarity are important.

□