Specification and Proof in Membership Equational Logic[1]

Adel Bouhoula§†, Jean-Pierre Jouannaud§‡ and José Meseguer§

§ SRI International
Computer Science Laboratory, 333 Ravenswood Avenue
Menlo Park, California 94025, USA
Email: {bouhoula, jouannau, meseguer}@csl.sri.com

† LORIA – INRIA Lorraine
615, rue du Jardin Botanique - B.P. 101
54602 Villers-lès-Nancy Cedex, France
Email: Adel.Bouhoula@loria.fr
http: //www.loria.fr/~bouhoula

‡ LRI, CNRS and Université de Paris-Sud
Bât 490, 91405 Orsay Cedex, France
Email: Jean-Pierre.Jouannaud@lri.fr
http: //www.lri.fr/~jouannau

[1]Supported by the ESPRIT Basic Research Working Group 22457 - Construction of Computational Logics II, the HCM Research Contract CONSOLE, the Office of Naval Research Contracts N00014-95-C-0225 and N00014-96-C-0114, National Science Foundation Grant CCR-9633363, and by the Advanced Software Enrichment Project of the Information Technology Promotion Agency, Japan (IPA).

Abstract: This paper is part of a long-term effort to increase expressiveness of algebraic specification languages while at the same time having a simple semantic foundation on which efficient execution by rewriting and powerful theorem-proving tools can be based. In particular, our rewriting techniques provide semantic foundations for Maude's functional sublanguage, where they have been efficiently implemented.

This effort started in the late seventies, led by the ADJ group, who promoted equational logic and universal algebra as the semantic basis of program specification languages. An important later milestone was the work around order-sorted algebras and the OBJ family of languages developed at SRI-International in the eighties. This effort has been substantially advanced in the mid-nineties with the development of Maude, a language based on membership equational logic.

Membership equational logic is quite simple, and yet quite powerful. Its atomic formulae are equations and sort membership assertions, and its sentences are Horn clauses. It extends in a conservative way both (a version of) order-sorted equational logic and partial algebra approaches, while Horn logic with equality can be very easily encoded.

After introducing the basic concepts of the logic, we give conditions and proof rules with which efficient equational deduction by rewriting can be achieved. We also give completion techniques to transform a specification into one meeting these conditions. We address the important issue of proving that a specification protects a subspecification, a property generalizing the usual notion of sufficient completeness. Using tree-automata techniques, we develop a test-set based approach for proving inductive theorems about a parameterized specification. We briefly discuss the extension of our techniques in presence of associative-commutative axioms and of extra variables in conditions, as well as solving goals by narrowing. Finally, we discuss the generality of our approach and how it extends several previous approaches.

Chapter 1

Introduction

This paper is part of an effort to increase the expressiveness of algebraic specification languages while at the same time having a simple semantic foundation on which both the operational semantics of such languages, and theorem proving tools supporting formal verification can be based. In particular, the semantic concepts and proof techniques that we propose have emerged out of, and provide foundation for, work on the functional sublanguage of Maude [78, 73, 65], which extends in substantial ways the OBJ language [32, 42, 39, 52].

Regarding expressiveness of algebraic specifications, it has for a long time been recognized that it is very important in practice to support subsorts, partiality, errors, and overloading of function symbols. Our ideas extend and unify within a simple semantic framework two different lines of work in algebraic specification, namely the order-sorted approach initiated by Goguen in the late 1970's, and different partial algebra approaches. The theoretical framework on which this unification is achieved is quite simple. We assume a family of *kinds*, $\mathcal{K}$, and a many $\mathcal{K}$-kinded signature of operations Σ. Each kind $K \in \mathcal{K}$ has an associated set of *sorts* S_K. Each sort $s \in S_K$ is interpreted as a unary membership predicate for the sort s, defining a subset $\mathcal{A}_s \subseteq \mathcal{A}_K$ at the level of an algebra $\mathcal{A}$. Atomic formulae are either $\mathcal{K}$-kinded Σ-equations $T = U$, or membership assertions $T : s$, and general sentences are Horn clauses on these atomic formulae. The intuitive interpretation is that data elements that have a kind K, but do not have a sort are *undefined*, or *error* elements. Axioms in a specification can prescribe subsort inclusions, as well as definedness of an overloaded operator for different arity and coarity sorts.

The simplicity of the membership algebra framework allows an efficient *operational semantics* by rewriting (or narrowing when a specification is seen as a logic program in the PROLOG sense) that makes specifications executable. Such a semantics, which justifies many of the design decisions made in the implementation of Maude [65], is investigated in detail in this paper, by deriving from the general deduction rules for the logic more efficient equivalent rules for rewriting under reasonable assumptions about the oriented equations. In this regard, the simplicity of our framework provides a satisfactory solution to many problems, like sort-decreasingness, that the more restrictive logics had to face. One of the main problems with the earlier approaches was that sort-decreasingness was not closed under completion. This is no more the case here, since we can easily add semantic-preserving membership axioms. This is a main advantage over previous (some of them quite complex) attempts to settle this question [37, 36, 22, 47].

Besides operational semantics and completion techniques, we also study in detail theorem proving techniques supporting verification of specifications in membership equational logics. Such techniques include methods for proving *sufficient completeness* of a specification relative to a subspecification of constructors, and *inductive proof techniques* that extend the many-sorted test-set based inductive theorem proving approach to the more expressive context of membership specifications. An important ingredient of this extension is the encoding of a relevant subset of membership equational logic specifications as *tree automata* with equality (and membership) tests introduced in [6, 16] and further studied in [15, 24]. We also consider the extension of these techniques to reason about *parameterized* specifications. In both cases, the main novel aspect of our technique is to refine a given conjecture step by step until it does not contain any more defined symbols. Tree automata techniques are then applied to the resulting constructor clauses [23].

Due to the overwhelming literature on algebraic specification and automated theorem proving techniques, the set of references and the discussion of related work in the present paper cannot be complete. We apologize for having made a necessary selection. However, the paper [75] studies in detail how a wide variety of equational specification formalisms can be faithfully represented in membership equational logic; we give a summary of these relationships in Section 11. For the moment, we can mention that our formalism extends to the many-kinded case the (one kinded) classified algebra ideas of Wadge and Stuart [86]; has some similarities with the order-sorted approaches in [84, 85]; generalizes and extends (an adequate version of) order-sorted algebra [40, 38, 41, 44, 83]; and admits faithful translations of typed algebra approaches such as those of Mosses [81], and of Manca, Salibra and Scollo [67]. In addition, it also admits faithful translations of many *partial* equational specification formalisms, including the wide range of such formalisms studied by Mossakowski in [80]. A particular approach to automated deduction for partial algebras with a similar motivation to ours in the treatment of sorts and subsorts, but different and more complex because of its explicit partiality –which requires the use of hard-to-read decorated terms in the design of the inference rules and at the implementation level as well– is [47]. On the other hand, this work restricts the use of membership declarations by only considering sort assertions for variables.

The present paper further develops ideas presented in a somewhat summarized form in three preliminary papers. The paper [74] studies in detail the model theoretic aspects of the logic and the semantic connections with order-sorted and partial equational logics. The paper [10] studies in detail the tree automata based inductive theorem proving techniques that are further developed here within the framework of membership equational logic. Finally, the paper [11] was the preliminary version of the present one. A definitive version of the model-theoretic aspects of membership equational logic, its generalization to partial membership equational logic, and a systematic study of its good properties as a logical framework in which a wide variety of equational specification formalisms can be represented can be found in [75].

We take this opportunity to warmly thank the TAPSOFT organizers, and in particular Max Dauchet, for inviting us to confront our ideas[1] with a quite large audience at Lille, and to develop them further in a journal setting.

We assume some familiarity with rewriting techniques [27] and universal many-sorted algebras [76].

We describe our Horn clause language in Section 3. Functional computations with these Horn clauses are described in Section 4, where we introduce three main properties, confluence, type-decreasingness, and regularity, which are further investigated in Section 6.

[1]It must be noted that the last two authors actually started discussing a preliminary order-sorted form of these ideas back in 1987, and advocated them in several circumstances as potential answers to various problems arising with the semantics of OBJ2 and OBJ3. Later, the last author and his coworkers went on with their elaboration as part of the Maude implementation.

Relationships with tree automata are investigated in Section 5. Logical computations are considered in Section 10.4. Theorem proving via completion techniques is dealt with in Section 7. Protecting extensions and its particular instance, sufficient completeness are addressed in Section 8 where we show a precise relationship between these properties and particular statements to be proved by induction. Proving inductive consequences, including the case of parameterized specifications, is done in Section 9. The framework is extended in Section 10 to the case where there are unorientable equations and possibly extra variables in conditions. Related work is discussed in Section 11, and concluding remarks appear in Section 12.

Chapter 2

Preliminaries

We first recall a few standard notions of universal algebra.

2.1 Many-Kinded Signatures and Terms

In this article, we will use the word *kind* instead of the more usual word *sort*, that we will reserve for another purpose. A *many-kinded signature* is a pair $(\mathcal{K}, \Sigma)$ in which $\mathcal{K}$ is a set of *kinds* and Σ is a $\mathcal{K}^* \times \mathcal{K}$-indexed family of sets $\{\Sigma_{\overline{K} \to K}\}_{(\overline{K} \in \mathcal{K}^*, K \in \mathcal{K})}$ so that each *function symbol* $f \in \Sigma_{\overline{K} \to K}$ is equipped with input kinds $\overline{K}$ and an output kind K. The case where $\overline{K}$ is empty yields the set $\{\Sigma_K\}_{K \in \mathcal{K}}$ of *constants*. We assume that $\Sigma_{\overline{K} \to K} \cap \Sigma_{\overline{K} \to K'} = \emptyset$ if $K \neq K'$.

Given now a $\mathcal{K}$-kinded set $\mathcal{X} = \{\mathcal{X}_K\}_{K \in \mathcal{K}}$ of *variables*, whose sets $\mathcal{X}_K$, for $K \in \mathcal{K}$ are all pairwise disjoint and denumerable (and disjoint from Σ), we define the set of *many-kinded* Σ*-terms* $\mathcal{T}_\Sigma(\mathcal{X})$ as usual: a variable of $\mathcal{X}_K$ is a term of kind K; $f(U_1, \ldots, U_n)$ is a term of kind K iff $f \in \Sigma_{K_1 \times \ldots \times K_n \to K}$ and $\forall i \in [1..n]$, U_i is a term of kind K_i. We will sometimes say that a term of kind K inhabits K. The capital letters $L, M, N, R, S, T, U, V, W$ will denote terms.

Note that $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_{\Sigma \cup \mathcal{X}}(\emptyset)$ (where $\Sigma \cup \mathcal{X}$ denotes the signature obtained by adding to Σ the variables $\mathcal{X}_K$ as extra constants of kind K for each $K \in \mathcal{K}$) are the same sets of terms. Note also that a term has a unique parse, hence a unique kind, thanks to our assumption that the output kind of a given function symbol is completely determined by its input kinds. This uniqueness property of parsing is not enough, though, to imply kind-inference for the variables due to the remaining possible overloading of function symbols having different input kinds.

Terms are identified with finite labelled trees as usual. *Positions* are strings of positive integers. Λ is the empty string (root position), $*$ is the concatenation of strings, and $|p|$ stands for the length of the string p. We use $\mathcal{P}os(U)$ for the set of positions in U, $\mathcal{FP}os(U)$ for its set of non-variable positions and $\mathcal{VP}os(U)$ for its set of variable positions. The *depth* (resp. *non-variable depth*) of a term t is the maximum length of a position $p \in \mathcal{P}os(t)$ (resp. $p \in \mathcal{FP}os(t)$).

The *subterm* of M at position p is denoted by $M|_p$, and we write $M \trianglerighteq M|_p$. The result of replacing $M|_p$ with N at position p in M is denoted by $M[N]_p$. This notation is also used to indicate that N is a subterm of M, in which case p may be omitted. We use $\mathcal{V}ar(M)$ for the set of variables of M. Terms without variables are called *ground*. We assume that each kind contains a ground term [1].

[1] This condition need actually not be satisfied, at the price of a slightly more complex deduction system

2.2 Many-Kinded Algebras and Homomorphisms

Given a $\mathcal{K}$-kinded signature Σ, a Σ-*algebra* is a $\mathcal{K}$-indexed set $\mathcal{A} = \{\mathcal{A}_K\}_{K\in\mathcal{K}}$ together with an assignment to each $f \in \Sigma_{K_1\times\ldots K_n\to K}$ of a function $\mathcal{A}_f : \mathcal{A}_{K_1} \times \ldots \times \mathcal{A}_{K_n} \to A_K$. A Σ-*homomorphism* $h : \mathcal{A} \to \mathcal{B}$ between two Σ-algebras is a $\mathcal{K}$-indexed family of functions $h = \{h_k\}_{K\in\mathcal{K}}$ such that for each $f \in \Sigma_{K_1\times\ldots K_n\to K}$, we have

$$h_K \circ \mathcal{A}_f = \mathcal{B}_f \circ (h_{K_1} \times \ldots \times h_{K_n}),$$

a condition which specializes to $h_K \circ \mathcal{A}_f = \mathcal{B}_f$ when f is a constant.

2.3 Many-Kinded Substitutions

Substitutions are endomorphisms of the term algebra $\mathcal{T}_\Sigma(\mathcal{X})$ that will apply here to finite terms, leaving fixed all variables but a finite number of them. Therefore, they can be written $\{x_1 \mapsto M_1, \ldots, x_n \mapsto M_n\}$, where M_i and x_i are assumed to be different terms of the same kind. We use Greek letters for substitutions and postfix notation for their application.

A term T is *subsumed* by a term S if $T = S\sigma$ for some substitution σ. We also say that S is *more general* than T. Subsumption is a quasi-ordering on terms denoted by $\stackrel{\bullet}{\geq}$, whose strict part is well-founded, and whose equivalence, $\stackrel{\bullet}{=}$, called *conversion*, is given by the (kind-preserving) renaming of variables. This ordering is extended to substitutions by letting $\sigma \stackrel{\bullet}{\geq} \tau$ if $\sigma = \tau\theta$ for some θ. We say that two many-kinded terms S and T *unify* if there exists a substitution σ such that $S\sigma = T\sigma$, and that they *overlap* if one of them unifies with a subterm of the other. The set of unifiers of two given terms S, T possesses a unique (up to conversion) minimal unifier with respect to subsumption, called the *most general unifier* of S and T, and denoted by $mgu(S,T)$.

We denote by $\overline{u}$ the list (or vector) $(u_1, \ldots, u_n)$. Given two vectors $\overline{u}$ and $\overline{v}$ of equal length over the respective sets E and E', and a binary relation $*$ over $E \times E'$, we will use the notation $\overline{u} * \overline{v}$ as an abbreviation for $u_1 * v_1 \wedge \ldots \wedge u_n * v_n$. Similarly, $\overline{(u,v)}$ will denote the vector of pairs $((u_1,v_1), \ldots, (u_n,v_n))$ and $\overline{x} \mapsto \overline{M}$ will denote the substitution $\{x_1 \mapsto M_1, \ldots, x_n \mapsto M_n\}$, with $\overline{x}$ denoting a set.

2.4 Many-Kinded Equations and Rewriting

We call a *rule* an ordered pair of many-$\mathcal{K}$-kinded terms of the same kind $K \in \mathcal{K}$, written $S \to T$. Given a set $\mathcal{R}$ of rules, we say that S *rewrites* to T with the rule $L \to R$ at position $p \in \mathcal{P}os(S)$, written $S \longrightarrow^p_{L\to R} T$ iff $S|_p = L\sigma$ for some substitution σ, and $T = S[R\sigma]_p$. The reflexive transitive closure of the rewrite relation on terms generated by a set of rules $\mathcal{R}$ is called *derivation*, written $S \longrightarrow^*_{\mathcal{R}} T$.

We call an *equation* an unordered pair of many-$\mathcal{K}$-kinded terms of the same kind $K \in \mathcal{K}$, written $S = T$. To each equation $S = T$, we can associate the two rules $S \to T$ and $T \to S$. The derivation relation generated by the set of rules associated to a given set $\mathcal{E}$ of equations is called the *equational theory* generated by $\mathcal{E}$, and is denoted by $\longleftrightarrow^*_{\mathcal{E}}$. It is actually the least congruence [2] generated by the instances of the equations in $\mathcal{E}$.

---

than the one shown in figure 3.3.

[2] A congruence on terms generated by a set $\{S_i = T_i\}_{i\in I}$ of pairs of terms of the same kind is the least transitive, reflexive, symmetric relation containing the pairs in $\{U[S_i] = U[T_i] \mid i \in I,\ S_i = T_i \in \mathcal{E}$ and $U[S_i]$ and $U[T_i]$ are $\mathcal{K}$-kinded terms$\}$. By our assumption that there is a ground term inhabiting

In the sequel, we will often use conditional equations and rules, whose precise definition is not necessary now.

2.5 Abstract Properties of Relations

Given a relation $\longrightarrow$ on a set S, we denote by $\longleftarrow$ its inverse, by $\longleftrightarrow$, $\stackrel{+}{\longrightarrow}$, $\stackrel{*}{\longrightarrow}$ and $\stackrel{*}{\longleftrightarrow}$ its symmetric, transitive, reflexive transitive, and reflexive symmetric transitive closures, respectively, and by $\downarrow$ and $\uparrow$ the relations $\stackrel{*}{\longrightarrow}\stackrel{*}{\longleftarrow}$, and $\stackrel{*}{\longleftarrow}\stackrel{*}{\longrightarrow}$, respectively.

Given a relation $\longrightarrow$ on the set S, an element $s \in S$ is in *normal form* if there is no $t \in S$ such that $s \longrightarrow t$. A *normal form* of $s \in S$ is an element in normal form $t \in S$ such that $s \longrightarrow^* t$. We denote by $s{\downarrow}$ the set of normal forms of s.

A relation $\longrightarrow$ is *terminating* if there are no infinite sequences of rewrites $t_1 \longrightarrow t_2 \longrightarrow \ldots \longrightarrow t_n \longrightarrow \ldots$. It is *confluent* if $s \downarrow t$ for all pairs (s,t) of elements in S such that $s \uparrow t$. It is *Church-Rosser* if $s \downarrow t$ for all pairs (s,t) of elements in S such that $s \stackrel{*}{\longleftrightarrow} t$. It is well-known that a relation is Church-Rosser iff it is confluent. When $\longrightarrow$ is confluent and terminating, there is a unique t in normal form such that $s \stackrel{*}{\longrightarrow} t$, called the *normal form* of s, and denoted again by $s{\downarrow}$.

2.6 Orderings

We will make extensive use of well-founded quasi-orderings for proving strong normalization properties. For us, a well-founded quasi-ordering will be a quasi-ordering whose strict part does not admit infinite decreasing chains. The following results will play a key role, see [27]:

Assume $\rightarrow_1$ and $\rightarrow_2$ are well-founded orderings on sets S_1, S_2. Then their lexicographic composition $(\rightarrow_1, \rightarrow_2)_{lex}$ is a well-founded ordering on $S_1 \times S_2$.

Assume $>$ is a well-founded ordering on a set S. Then $>_{mul}$ is a well-founded ordering on the set of multisets of elements of S. It is defined as the transitive closure of the following relation $>>$ on multisets (using $\cup$ for multiset union):

$$M \cup \{s\} >> M \cup \{t_1, \ldots, t_n\} \quad \text{if } s > t_i \ \forall i \in [1..n]$$

Assume $\rightarrow$ is a terminating rewrite relation. Then, $(\rightarrow \cup \rhd)^*$ is a well-founded ordering.

The above definitions extend easily to quasi-orderings, in which case well-foundedness applies to their strict part.

2.7 Bottom-up Tree Automata

Many-kinded signatures are bottom-up tree automata, in which kinds become states, with a subset of kinds chosen as accepting states, and operation declarations become transitions.

A *bottom-up tree automaton*, or simply *automaton*, is a triple $(\mathcal{K}, \Sigma, \mathcal{F})$, where $(\mathcal{K}, \Sigma)$ is a many-kinded signature whose kinds are called *states*, and whose declarations are called *transitions*. $\mathcal{F}$ is a subset of $\mathcal{K}$ whose elements are called *accepting states*.

Recognizing a term T is done by rewriting T according to the transitions, which are defined as follows: to an automaton $\mathcal{A} = (\mathcal{K}, \Sigma, \mathcal{F})$, we associate a many-kinded signature

each kind, it follows that such a congruence describes all the equations that are a logical consequence of the original set of equations.

$(\mathcal{K}, \Sigma_{\mathcal{A}})$ and a rewrite system $\mathcal{R}_{\mathcal{A}}$ over the signature $\Sigma_{\mathcal{A}}$:

$$\Sigma_{\mathcal{A}} = \{f_{\overline{K},K} : \overline{K} \to K\}_{f:\overline{K}\to K \in \Sigma}$$
$$\mathcal{R}_{\mathcal{A}} = \{f_{\overline{K},K}(\overline{K}) \to K\}_{f:\overline{K}\to K\ \in \Sigma}$$

A term $T \in \mathcal{T}_{\Sigma}(\emptyset)$ is said to be *recognized* by the automaton if it rewrites to an accepting state K using the rules in $\mathcal{R}_{\mathcal{A}}$. We will also say sometimes that T inhabits K. Note that a term is recognized iff it belongs to $\mathcal{T}_{\Sigma}(\emptyset)$, hence the automaton acts as a parser for ground terms over such a specification. Note also that the term T is destroyed along the run of the automaton. We will adopt a slightly different definition in Section 5 for which terms are carried along the run.

The example given in Figure 2.1 below represents a simple specification of natural numbers. Representing sorts, the nodes have sort names. The curved arrows between states represent operation declarations, while the straight decorated arrows coming in a state represent constant declarations.

```
fmod NAT
sorts Nat
op S_ : Nat  →  Nat
op 0 : Nat
end fmod
```

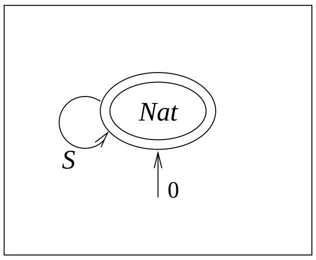


Figure 2.1: A specification of natural numbers and its associated automaton

Chapter 3

Language

Our language is a many-kinded first-order language whose only predicates are an infix equality, denoted by $\_ = \_$, and unary membership predicates, denoted by $\_ : s$, one for each sort s in the given signature. These predicates allow us to state two kinds of Horn clauses: conditional equations whose head is an equality atom; and conditional memberships, whose head is a membership atom.

3.1 Signatures and Axioms

Definition 1 *A* signature *in membership equational logic is a pair Ω consisting of a many-kinded signature $(\mathcal{K}, \Sigma)$, and of a disjoint $\mathcal{K}$-kinded family of sets of* sorts $\mathcal{S} = \{\mathcal{S}_K\}_{K \in \mathcal{K}}$.

We will denote sorts by the small letters s, t, u, v, w.

It is convenient (and faithful) to represent each kind K as the set $\mathcal{S}_K \cup \{K^?\}$, where $K^?$ acts as a built-in error sort for those computations taking place in kind K which do not return a value inhabiting a sort in $\mathcal{S}_K$. Identifying $\mathcal{S}_K$ with K itself would not be correct, since $\mathcal{S}_K$ could be empty for several kinds. In Maude, the sort $K^?$ is automatically generated. We may also use it in some examples.

From now on, we assume given a set $\mathcal{X}$ of $\mathcal{K}$-kinded variables that will be used in the axioms. We will denote by K_x the kind of a variable $x \in \mathcal{X}$.

Definition 2 *Atomic* Ω-formulas *in membership equational logic are either* equalities $S = T$ *or* memberships $S : s$*, where S, T are many-kinded Σ-terms of the same kind, and s is a sort in the kind of S.* Ω-sentences *are then conditional axioms of the form*

$$(\forall \overline{x})\ \phi\ \texttt{if}\ \phi_1 \wedge \ldots \wedge \phi_n$$

where $\phi, \phi_1, \ldots, \phi_n$ are atomic Ω-formulas, and the finite many-kinded set of variables $\overline{x} \subseteq \mathcal{X}$ contains all the variables occurring in $\phi, \phi_1, \ldots, \phi_n$. Such axioms are either conditional memberships*:*

$$\forall \overline{x}\ L(\overline{x}) : s\ \texttt{if}\ \overline{U}(\overline{x}) : \overline{t'} \wedge \overline{V}(\overline{x}) = \overline{W}(\overline{x})$$

where L is a many-kinded term of kind K, s is a sort of kind K, and $\overline{U}, \overline{V}, \overline{W}$ are vectors of many-kinded terms, or conditional equalities*:*

$$\forall \overline{x}\ L(\overline{x}) = R(\overline{x})\ \texttt{if}\ \overline{U}(\overline{x}) : \overline{t'} \wedge \overline{V}(\overline{x}) = \overline{W}(\overline{x})$$

where, as previously, L and R are many-kinded terms of the same kind K, and $\overline{U}, \overline{V}, \overline{W}$ are vectors of many-kinded terms.

$L = R$ *or* $L : s$ *is called the* head *of the axiom, while* $\overline{U}(\overline{x}) : \overline{t'} \wedge \overline{V}(\overline{x}) = \overline{W}(\overline{x})$ *is its* body *or* condition. *We will often omit the set of (universally quantified) many-kinded variables* $\overline{x}$ *when it is not necessary to carry it along.*

We insist on the fact that the universally quantified variables in the axioms belong to $\mathcal{X}$, hence are $\mathcal{K}$-kinded. The set $\mathcal{X}$ will first be considered as fixed, and each axiom will use a finite subset $\overline{x}$ of $\mathcal{X}$. This point of view will evolve slightly when environments will be used in proofs.

Conditional equations and conditional memberships complement each other: the language of conditional equations is used to specify the meaning of those functions that are not meant to be constructors, while the language of conditional memberships is used to define the sets (each one in some kind) on which these functions are total. This is therefore a language of partial functions that become defined on subdomains definable in the logic.

This language is indeed powerful enough so as to encode many (usually meta-theoretic) concepts. For example:

- *Subsorts declarations* $s' \leq s$ are membership axioms of the form $x : s \texttt{ if } x : s'$. In our framework, such memberships may be *deducible* from other axioms, which prevents us, in general, to define the *subsort ordering* before introducing deductions. In the `NUMBER` example given in Figure 3.1, we follow the tradition of using subsort declarations which should of course be interpreted as membership axioms of the above form.

- *Order-sorted operator declarations* of the form $f : s_1 \times \ldots \times s_n \rightarrow s$ where $s_1, \ldots, s_n, s$ are sorts, and not kinds, are conditional membership axioms of the form $f(\overline{x}) : s \texttt{ if } \overline{x} : \overline{s}$. The signature becomes *overloaded on sorts* when there are several axioms of the above form for a given function symbol f. Note that membership equational logic allows us to write a more general form of declaration, namely $f(\overline{x}) : s \texttt{ if } \overline{y} : \overline{s}$, where $\overline{y} \subset \overline{x}$, which is actually an error-recovery axiom.

- The requirement that a sort is contained in the intersection of two other sorts can be achieved by a membership axiom of the form $x : s \texttt{ if } x : s' \wedge x : s''$.

3.2 Membership Algebras and Satisfaction

The models of membership equational logic are membership algebras. They are many-kinded Σ-algebras with a specification of a subset for each sort s.

Definition 3 *For* $\Omega = ((\mathcal{K}, \Sigma), \mathcal{S})$ *a signature in membership equational logic, an* Ω-algebra *is a* Σ*-algebra* $\mathcal{A}$ *together with the assignment to each sort* $s \in \mathcal{S}_K$ *of a subset* $\mathcal{A}_s \subseteq \mathcal{A}_K$. *An* Ω*-homomorphism* $f : \mathcal{A} \rightarrow \mathcal{B}$ *between two such* Ω*-algebras is a* Σ*-homomorphism such that for each* $s \in \mathcal{S}_K$*, we have* $f_K(\mathcal{A}_s) \subseteq \mathcal{B}_s$. *This defines a category* Alg_Ω *in the obvious way.*

We may sometimes simplify our notation by writing the signature $\Omega = ((\mathcal{K}, \Sigma), \mathcal{S})$ in the flat form $\Omega = (\mathcal{K}, \Sigma, \mathcal{S})$.

We can then define satisfaction of an Ω-axiom in an Ω-algebra. Note that a $\mathcal{K}$-kinded map $a : \mathcal{X} \rightarrow \mathcal{A}$, called an *assignment*, extends in a unique way, by the freeness of the

$\mathcal{K}$-kinded algebra $\mathcal{T}_\Sigma(\mathcal{X})$, to a Σ-homomorphism $\overline{a} : \mathcal{T}_\Sigma(\mathcal{X}) \to \mathcal{A}$. Assignments are defined on $\mathcal{X}$, although we will use their restrictions to finite subsets $\overline{x}$ of $\mathcal{X}$ [1].

We then say that the Ω-algebra $\mathcal{A}$ with assignment a satisfies the equation $(\forall \overline{x})\ S = T$, where $\mathcal{V}ar(S,T) \subseteq \overline{x}$, iff $\overline{a}(S) = \overline{a}(T)$, and use the notation $\mathcal{A}, a \models_\Omega (\forall \overline{x})\ S = T$ to denote such satisfaction. Similarly, $\mathcal{A}, a \models_\Omega (\forall \overline{x})\ S : s$ holds iff $\overline{a}(S) \in \mathcal{A}_s$.

Definition 4 *An Ω-algebra A satisfies a conditional axiom $(\forall \overline{x})\ \phi$ `if` $\phi_1 \wedge \ldots \wedge \phi_n$, written $A \models_\Omega (\forall \overline{x})\phi$ `if` $\phi_1 \wedge \ldots \wedge \phi_n$, iff $A, a \models_\Omega \phi$ for each assignment $a : \mathcal{X} \to A$ such that $A, a \models_\Omega \phi_i$ for each $i \in [1..n]$. For $\mathcal{E}$ a set of such conditional axioms, we write $A \models_\Omega \mathcal{E}$ iff $A \models_\Omega \varphi$ for each $\varphi \in \mathcal{E}$. The Ω-algebras that satisfy a set $\mathcal{E}$ of conditional axioms define a full subcategory $Alg_{\Omega,\mathcal{E}}$ of Alg_Ω in the obvious way.*

3.3 Specifications

Definition 5 *A* specification *or* theory *in membership equational logic is a pair $(\Omega, \mathcal{E})$ consisting of a signature Ω in membership equational logic and a set of axioms $\mathcal{E}$ on this signature.*

It will often be convenient to name a theory after the set of rules it contains, writing for example $\mathcal{E} = (\Omega, \mathcal{E})$. This amounts to making the signature of the theory implicit, while being able to retrieve it if needed thanks to this convention.

The notion of a specification in membership equational logic generalizes the more familiar notion of an order-sorted specification. We already hinted at this relationship that we will later investigate in more detail. As a consequence, some particular specifications in membership equational logic can be seen as order-sorted specifications:

Definition 6 *A specification $(((\mathcal{K}, \Sigma), \mathcal{S}), \mathcal{E})$ in membership equational logic is* order-sorted *if its axioms are of the following forms:*

(i) subsort declarations: $x : t$ `if` $x : s$, also written $s \leq_\mathcal{E} t$, such that the set of these declarations generates a partial order $\leq_\mathcal{E}$,

(ii) signature declarations: $f(\overline{x}) : s$ `if` $\overline{x} : \overline{s}$, also written $f : \overline{s} \to s$,

(iii) order-sorted equations: $L(\overline{x}) = R(\overline{x})$ `if` $\overline{x} : \overline{s} \wedge \overline{U}(\overline{x}) = \overline{V}(\overline{x})$

The triple $(\mathcal{S} = \bigcup_{K \in \mathcal{K}} \mathcal{S}_K,\ \leq_\mathcal{E},\ \Sigma' = \bigcup_{\overline{s},s} f : \overline{s} \to s)$ is called the signature of the associated order-sorted specification.

We may sometimes simplify our notation by writing the specification $(((\mathcal{K}, \Sigma), \mathcal{S}), \mathcal{E})$ in the flat form $(\mathcal{K}, \Sigma, \mathcal{S}, \mathcal{E})$.

Notice that the above notion of order-sorted specification is somewhat more general than the usual notion, because we do not require the terms $L, R, \overline{U}, \overline{V}$ in conditional equations to actually have a sort in the corresponding order-sorted signature.

Ordinary order-sorted specifications do not actually make any reference to a notion of kind as in specifications in membership equational logic, although kinds surface at the semantic level in the form of the set of elements in a connected component of the subsort relationship. We will freely use the notion of order-sorted specification in the next sections, with or without kinds. In particular, the kind declaration will be frequently omitted when there is a single kind. The reader will find a precise characterization of the

[1] Note that the fact that an assignment $a : \overline{x} \longrightarrow \mathcal{A}$ can be regarded as the restriction of an assignment $a : \mathcal{X} \longrightarrow \mathcal{A}$ depends on our assumption that each kind is inhabited by a ground term. Again, this is a convenient, but not essential, assumption for our treatment, which could be dropped to allow empty kinds.

generalization of order-sorted specifications by specifications in membership equational logic in Section 11, together with a detailed discussion relating the classes of models in both kinds of specifications, which summarizes [75].

As many-kinded signatures, order-sorted ones can be easily encoded as automata, and indeed order-sorted signatures were already depicted as automata in [32]. For this, it suffices to add empty transitions encoding the subsort ordering to the underlying many-sorted signature. Again, we will often describe order-sorted signatures as automata by viewing subsort declarations as empty transitions. These automata will be represented as previously, with the additional convention that straight unlabelled arrows joining two states of the automaton represent subsort declarations, see, e.g. figure 5.1.

Order-sorted specifications have been the subject of numerous studies since their introduction by Joseph Goguen in the late seventies [40, 32, 44, 38, 83, 85]. This work extends the order-sorted framework while keeping its conceptual elegance. With this new framework, we make progress in four different directions. First, and this is most important, all terms are many-sorted, hence there is a well-defined syntactic notion of a term which makes sense. As a consequence, the order-sorted anomaly that two semantically equal terms may not be both well-formed has disappeared, making the use of *retracts* [32, 41, 44] unnecessary. Second, and this was already pointed out, our language allows us to specify partial functions which are indeed total on subdomains definable in Horn logic of equality and membership. Hence, partiality can be studied by proof theoretic means; this will be done in Section 3.4. Third, the logic is the simplest, yet most expressive, first-order logic we can think of for defining functions, a claim which is supported in Section 11. Fourth, as a Horn logic, it has a simple proof theory, and enjoys an initial algebra semantics. Order-sorted logic does enjoy an initial algebra semantics as well, but it appears that its proof theory is actually complicated by several technical anomalies that disappear in the richer framework of membership equational logic.

As a simple example of specification using the expressive power of membership equational logic, we give below a specification of numbers. We use a syntax inspired by those of OBJ [39] and Maude [78, 65], but we allow for a few space-saving conventions, while trying to keep as close as possible to our definition of membership equational logic. In particular, we have deliberately refrained from breaking this specification into smaller pieces by means of structuring mechanisms. As it is by now well understood, the issue of structuring is orthogonal to the logic itself. That is why it is indeed tackled in the last sections of the paper. After the header, giving the name `NUMBER` to the specification, come the imported modules, here `BOOL` whose kind is called `Bool?`, and the many-sorted signature, with one kind, `Number`, refined to three sorts, `Nat, Int, Complex`, each one being a subsort of the next. The keyword `fmod` stands for functional module. Constructors and defined symbols are introduced by the respective keywords `cop` and `op`. Memberships and equations are introduced by the respective keywords `mb` and `eq`.

This example illustrates the expressiveness of our language, that is, its ability to encode many properties of the specification, whether true in all models or in the initial one, as conditional equations or memberships. More precise comments are:

- Note how kinds are used to express that `+` and `*` are polymorphic operators. Expressions like `s : {Nat, Int}` are indeed syntactic sugar allowing us to avoid the repetition of an axiom. We could have used additional syntactic sugar by declaring variables with given sorts. For expository purposes, we decided against this facility here, since it would hide the actual format of the axioms in our logic.

- There are several categories of membership constraints. Memberships (1–4) encode

```
fmod NUMBER is protecting BOOL
kind Number[Nat < Int < Complex]
cop 0 : Number
cop S, P : Number → Number
cop <_, _> : Number × Number → Number
op _+_, _*_ : Number × Number → Number [comm]
op _-_ : Number × Number → Number
op _>_ : Number × Number → Bool?
op Conj_, |_| : Number → Number
var x,y,x',y' : Number
mb 0 : Nat                                                      (1)
mb S(x) : s if x : s and s : {Nat, Int}                         (2)
mb P(x) : Int if x : Int                                        (3)
mb <x,y> : Complex if x : Int and y : Int                       (4)
mb x+y, x*y : s if x : s and y : s and s : {Nat, Int, Complex}  (5)
mb x-y : Nat if x,y : Nat and x>y = T                           (6)
mb x-y : Int if x,y : Int                                       (7)
mb Conj(x) : Complex if x : Complex                             (8)
mb |x| : Nat if x : Complex                                     (9)
mb x*x : Nat if x : Int                                         (10)
mb x*y : Nat if x : Complex and y : Complex and y = Conj(x)     (11)
eq P(S(x)) = x if x : Int                                       (1)
eq S(P(x)) = x if x : Int                                       (2)
eq <x,0> = x if x : Int                                         (3)
eq (x>0) = T if x : Nat                                         (4)
eq (0>S(x)) = F if x : Nat                                      (5)
eq S(x)>S(y) = x>y if x,y : Int                                 (6)
eq P(x)>P(y) = x>y if x,y : Int                                 (7)
eq S(x)>P(y) = x>P(P(y)) if x,y : Int                           (8)
eq P(x)>S(y) = x>S(S(y)) if x,y : Int                           (9)
eq x+0 = x                                                      (10)
eq x+S(y) = S(x+y) if x : Int and y : Int                       (11)
eq x+P(y) = P(x+y) if x : Int and y : Int                       (12)
eq x-0 = x                                                      (13)
eq x-P(y) = S(x-y) if x : Int and y : Int                       (14)
eq x*0 = 0                                                      (15)
eq x*S(y) = (x*y)+x if x : Int and y : Int                      (16)
eq x*P(y) = (x*y)-x if x : Int and y : Int                      (17)
eq <x,y>+<x',y'> = <x+x',y+y'> if x,x',y,y' : Int               (18)
eq <x,y>*<x',y'> = <x*x'-y*y',x*y'+x'*y> if x,x',y,y' : Int     (19)
eq x+<x',y'> = <x+x',y> if x,x',y' : Int                        (20)
eq x*<x',y'> = <x*x',x*y'> if x,x',y' : Int                     (21)
eq Conj(<x,y>) = <x,-y> if x,y : Int                            (22)
eq Conj(x) = x if x : Int                                       (23)
eq |x| = x*Conj(x) if x : Complex                               (24)
end fmod
```

Figure 3.1: A specification of numbers in membership equational logic

the order-sorted signature of the constructors, while memberships (5–9) encode the order-sorted signature of the defined operations. The latter memberships are not necessary in theory, since the corresponding properties can indeed be deduced for the initial model from the equations defining the operations by using an inductive argument (for that reason, we did not add yet another membership constraint stating that $x > y : Bool$ `if` $x : Int \ \wedge \ y : Int$). It is therefore good practice to run a theorem prover in order to check their validity as inductive consequences of the remaining axioms. But they also specify on which sorts a function symbol should be completely defined, allowing the prover to check sufficient completeness at these sorts. This remark will be exploited in Section 8. Finally, memberships (10) and (11) express complex inductive properties of the multiplication operation, making their derivation as inductive consequences of the other axioms a complex task.

- There is no membership axiom for specifying the sort of `x-y` when `x,y` are of sort `Complex`, and indeed, the semantics of `x-y` is only defined for the case where `x` and `y` are in `Nat` or in `Int`. So, this operation is defined on the sorts `Nat`×`Nat` and `Int`×`Int`. Although we could have given an additional membership axiom for the case where `x,y` are in `Complex`, we chose not to do so, therefore saving us from the burden of giving semantics at all sorts when this is not really needed in a given specification. As a consequence, `x-y` becomes an *error in the order-sorted element* of kind `Number` when `x,y` are complex numbers. This is an example of use of kinds to catch error terms.

- Successor and predecessor are two non-free constructors, since they appear as top function symbols in the first two equations. The constructor for complex numbers is not free either, due to the third equation. 0 is the only free constructor in this specification. This shows the practical need for dealing with non-free constructors.

- The equation `x+0 = 0` does not specify the sort of `x`. But, since the signature is many sorted, we know that `x` has to range over some kind, here the kind `Number` as it is declared via the keyword `var`. This equation may in particular apply to a term of kind `Number` not belonging to any of the sorts `Nat, Int, Complex`. Such terms exist in this specification, as already discussed. An axiom using a variable which is solely constrained by a kind declaration may therefore act as an error-recovery axiom. The user should of course be aware of this potentially dangerous effect.

  This possibility for a variable in an axiom to belong to a kind is systematically exploited in Maude for the axioms of associativity and commutativity which always apply to terms in a kind. Note further that kind declarations could be omitted, since the kind of a variable can always be inferred from the expressions in which it appears.

- The last equation raises an interesting problem: the lefthand side has sort `Nat` by using the membership axiom encoding the order-sorted specification of the operation $|\_|$. But the righthand side is the product of two complex numbers, hence would normally have sort `Complex`. Such *sort-increasing* rules could result in a lack of completeness of the computation mechanism, and this is why it may seem wise to add the inductive property stating that `x*Conj(x)` has sort `Nat`. This is actually not necessary, as we shall see later.

The example of Figure 3.2 shows how a bounded stack of complex numbers (in the sense of the specification of Figure 3.1) with a recovery operator can be naturally specified

```
fmod BD-STACK is protecting NUMBER
kind Stack?[NeStack < Stack]
cop Push_ _ : Number × Stack?  →  Stack?
cop Empty :  →  Stack?
op Recover_ : Stack?  →  Stack?
op Top_ : Stack?  →  Number
op Pop_ : Stack?  →  Stack?
op Length_ : Stack  →  Number
op Bound :  →  Number
eq Bound = 999
mb Empty : Stack
mb Push(x,y) : NeStack if x : Complex and y : Stack and Bound > Length(y)
eq Length (Empty) = 0
eq Length (Push(x,y)) = S(Length(y))
eq Top(Push(x,y)) = x if Push(x,y) : NeStack
eq Pop(Push(x,y)) = y if Push(x,y) : NeStack
eq Recover(Push(x,y)) = Recover(y) if Length(y) > Bound
eq Recover(y) = y if y : Stack
end fmod
```

Figure 3.2: A specification of bounded stacks

in membership equational logic. This example also shows how sort constraints in order-sorted algebra [41, 77] can be viewed as a special case of the more general conditional axioms in membership equational logic. The module BD-STACK imports the NUMBER module discussed previously. We slightly abuse syntax by assuming that decimal notation is available to avoid a long list of successor symbols. Note that the statement `protecting NUMBER` applies to the sorts, not to the kind `Number` itself, since new error messages like `Length(Pop(Push(S(<0,0>),A)))`, where `A` is a stack exceeding the bound, can now be generated. This is closely related to the appropriate way of understanding sufficient completeness for membership algebra specifications, as discussed in Section 8.

3.4 Deduction, Soundness and Completeness

As we have already noted, membership equational logic coincides with the special case of many-sorted Horn logic with equality where the general notion of signature, that is, a triple $(\mathcal{K}, \Sigma, \Pi)$ with $(\mathcal{K}, \Sigma)$ a $\mathcal{K}$-kinded signature and $\Pi = \{\Pi_{\overline{K}}\}_{\overline{K} \in \overline{\mathcal{K}}}$ a signature of predicates, is restricted so that Π consists only of *unary* predicates, where we denote Π_K by S_K and adopt the postfix notation $t : s$ for $s(t)$.

Therefore, all the good properties enjoyed by many-sorted Horn logic with equality are automatically inherited by membership equational logic. In particular, given a specification $(\Omega, \mathcal{E})$ in membership equational logic, the (sound and complete) set of deduction rules given in [43] specializes to the (sound and complete) set of deduction rules of Figure 3.3, denoted by $\vdash_\Omega$, where the quantifier notation $(\forall \overline{x})$ assumes that $\overline{x}$ is a $\mathcal{K}$-kinded set with disjoint components.

Kinds are of a syntactic nature. Unlike kinds, sorts are of a semantic nature. Given a specification, for example the specification of Figure 3.1, a given term has one kind, for

$$\textbf{Subject Reduction: } \frac{(\forall \overline{x})N : s \quad (\forall \overline{x})M = N}{(\forall \overline{x})M : s}$$

$$\textbf{Membership: } \frac{(\forall \overline{x})\overline{U}\sigma : \overline{t'} \quad (\forall \overline{x})\overline{V}\sigma = \overline{W}\sigma}{(\forall \overline{x})L\sigma : s}$$

where $L(\overline{y}) : s$ `if` $\overline{U}(\overline{y}) : \overline{t'} \wedge \overline{V}(\overline{y}) = \overline{W}(\overline{y}) \in \mathcal{E}$

$$\textbf{Reflexivity: } \frac{}{(\forall \overline{x})M = M}$$

$$\textbf{Symmetry: } \frac{(\forall \overline{x})M = N}{(\forall \overline{x})N = M}$$

$$\textbf{Transitivity: } \frac{(\forall \overline{x})M = N \quad (\forall \overline{x})N = P}{(\forall \overline{x})M = P}$$

$$\textbf{Congruence: } \frac{(\forall \overline{x})M_1 = N_1 \ \ldots \ (\forall \overline{x})M_n = N_n \quad f(\overline{M}) : K \quad f(\overline{N}) : K}{(\forall \overline{x})f(\overline{M}) = f(\overline{N})}$$

$$\textbf{Replacement: } \frac{(\forall \overline{x})\overline{U}\sigma : \overline{t'} \quad (\forall \overline{x})\overline{V}\sigma = \overline{W}\sigma}{(\forall \overline{x})L\sigma = R\sigma}$$

where $L(\overline{y}) = R(\overline{y})$ `if` $\overline{U}(\overline{y}) : \overline{t'} \wedge \overline{V}(\overline{y}) = \overline{W}(\overline{y}) \in \mathcal{E}$

Figure 3.3: Rules of deduction

example `0` has kind `Number`. But using the axioms, the rules of Figure 3.3 may enable us to prove that the same term indeed inhabits some specific sort of that kind. For example, `0` has sort `Nat`. `x*y` has sort `Nat` when `x,y` have sort `Complex` and `y = Conj(x)`: the use of this axiom for proving the membership `x*y` : `Nat` requires therefore proving the equality `y=Conj(x)`. This example shows that deduction of sorts and deduction of equalities depend on each other in the way specified by the inference system described in Figure 3.3. It follows that it is in general undecidable, but of course semi-decidable, to test whether a given term has a given sort.

The following two related results relating deduction with satisfaction follow from [43]:

Theorem 7 *(Soundness and Completenes) For any atomic Ω-sentence φ, we have*

$$\mathcal{E} \vdash_\Omega \varphi \ \textbf{ iff } \ \mathcal{E} \models_\Omega \varphi.$$

Theorem 8 *(Initial and Free Algebras) For $(\Omega, \mathcal{E})$ a specification in membership equational logic such that $\Omega = (\mathcal{K}, \Sigma, \mathcal{S})$, there is an $(\Omega, \mathcal{E})$-algebra $\mathcal{T}_{\Omega,\mathcal{E}}(\mathcal{X})$ and an assignment $\eta_\mathcal{X} : \mathcal{X} \longrightarrow \mathcal{T}_{\Omega,\mathcal{E}}(\mathcal{X})$ such that for each assignment $a : \mathcal{X} \longrightarrow A$ with $A \in Alg_{\Omega,\mathcal{E}}$ there is a unique Ω-homomorphism $\overline{a} : \mathcal{T}_{\Omega,\mathcal{E}}(\mathcal{X}) \longrightarrow A$ such that $\overline{a} \circ \eta_\mathcal{X} = a$. In particular, for $\emptyset$ the empty $\mathcal{K}$-kinded set, $\mathcal{T}_{\Omega,\mathcal{E}}(\emptyset)$, denoted $\mathcal{T}_{\Omega,\mathcal{E}}$, is initial in the category $Alg_{\Omega,\mathcal{E}}$.*

In fact, the construction of $\mathcal{T}_{\Omega,\mathcal{E}}(\mathcal{X})$ follows in a straightforward way from the rules of deduction as the quotient Σ-algebra $\mathcal{T}_\Sigma(\mathcal{X})/\equiv^X_\mathcal{E}$, where, by definition,

$$t \equiv^\mathcal{X}_\mathcal{E} t' \ \textbf{ iff } \ \mathcal{E} \vdash_\Omega \ (\forall \overline{x})t = t'$$

which indeed defines a Σ-congruence by the reflexivity, symmetry, transitivity, and congruence rules of deduction. The sort structure is then defined by $[t] \in \mathcal{T}_{\Omega,\mathcal{E}}(\mathcal{X})_s$ **iff** $\mathcal{E} \vdash_\Omega (\forall \overline{x}) t : s$, which is independent of the choice of t by virtue of *Subject Reduction*.

In the sequel, the subscript Ω in $\vdash_\Omega$ will be omitted.

3.5 Deductions in an environment

We will now make use of the notion of an *environment* Γ, assigning sorts to finitely many variables occurring in a proof. The use of an environment fits the implementation needs much better than the use of an a priori given set $\mathcal{X}$ of $\mathcal{K}$-kinded variables by allowing the variables in a proof to be taken care of by need. Formally, an environment is a $\mathcal{K}$-kinded partial function $\Gamma : \overline{x} \longrightarrow \mathcal{S}$, where $\overline{x}$ is finite and has disjoint components. A useful notation for Γ is as a set of pairs $x : s$ with x a variable in $\overline{x}$ and $s \in \mathcal{S}_{K_x}$ the sort $\Gamma(x)$ if Γ is defined for x, or the kind K_x otherwise. For example, $\Gamma = \{x_1 : s, x_2 : K_{x_2}, x_3 : s'\}$.

Given an Ω-algebra $\mathcal{A}$ and an environment Γ, an assignment $a : \Gamma \longrightarrow \mathcal{A}$ is a $\mathcal{K}$-kinded map $a : \overline{x} \longrightarrow \mathcal{A}$ of the corresponding variables such that if $\Gamma(x) = s$ then $a(x) \in \mathcal{A}_s$. In the case where $\mathcal{A}$ is the free Ω-algebra $\mathcal{T}_\Omega(\mathcal{X})$, an assignment, now denoted by σ, is called an *admissible substitution* with respect to Γ. As announced in Section 2.3, we use the postfix notation $t\sigma$ to denote the result of applying the substitution σ to the term t. In this case, for each $x \in \overline{x}$, the admissible substitution σ must also provide with an environment $\Gamma_{\sigma,x} : \overline{\mathcal{V}ar(x\sigma)} \longrightarrow \mathcal{S}$ ensuring that $x\sigma$ has sort $\Gamma(x)$, that is, $\Gamma_{\sigma,x} \vdash x\sigma : \Gamma(x)$. We will call such a pair made of a term and an environment assigning sorts to (some of) the variables in the term, a *sorted term*. Given a substitution σ of domain $\overline{x}$, we will denote by Γ_σ the environment $\bigcup_{x\in\overline{x}} \Gamma_{\sigma,x}$, assuming that for all $x, y \in \overline{x}$, $\Gamma_{\sigma,x}$ and $\Gamma_{\sigma,y}$ agree on each $z \in \mathcal{V}ar(x\sigma) \cap \mathcal{V}ar(y\sigma)$. We will later see that admissible substitutions preserve sorts.

Noting that Γ gives rise to a signature $\Omega(\overline{x})$ by adding disjointly the variables $\overline{x}$ as new constants of their corresponding kind, and can then be viewed as the collection of axioms $Ax(\Gamma) = \{(\forall\emptyset) x : \Gamma(x) \mid x \in \overline{x}\}$, it is easy to see that, denoting by $\mathcal{T}_{\Omega,\mathcal{E}}(\Gamma)$ the underlying Ω-algebra of $\mathcal{T}_{\Omega(\overline{x}),\mathcal{E}\cup Ax(\Gamma)}$, each assignment $a : \Gamma \longrightarrow \mathcal{A}$ with $\mathcal{A} \in Alg_{\Omega,\mathcal{E}}$ extends uniquely to a Ω-homomorphism $\overline{a} : \mathcal{T}_{\Omega,\mathcal{E}}(\Gamma) \longrightarrow \mathcal{A}$. That is, we can easily extend the free algebra construction using environments Γ instead of just K-kinded variables $\overline{x}$, which is often convenient.

The soundness and completeness of the rules of deduction in Figure 3.4 follows from that of the original rules of deduction in Figure 3.3, once we observe that the notations $\Gamma \vdash_\mathcal{E} M = N$, and $\Gamma \vdash_\mathcal{E} M : s$ correspond to deductions $(\forall\emptyset) M = N$ and $(\forall\emptyset) M : s$ in the theory $(\Omega(X), \mathcal{E} \cup \Gamma)$ instead than in the theory $(\Omega, \mathcal{E})$, where X is the set of variables in Γ, and $\Omega(X)$ is the signature obtained by adding them to Ω as new (disjoint) constants. The rules of deduction in Figure 3.4 are then essentially the specialization of those in Figure 3.3, except that the original membership rule has now been split into two rules, namely the *Variable* rule for the memberships $x : s \in \Gamma$ and the standard *Membership* rule for the remaining memberships in $\mathcal{E}$.

Similarly, if in the rules of deduction for membership equational logic that we have given we replace the quantifications $\forall(\overline{x})$ by quantifications $\forall(\Gamma)$ over environments, we again obtain sound and complete rules of deduction in a more convenient form. This allows us to adopt a more familiar view of deduction by using environments and judgments: our typing judgments are written as $\Gamma \vdash_\mathcal{E} M : s$ if the term M can be proved to have the sort s in the environment Γ by using the equations and memberships in $\mathcal{E}$, and our equality judgments are written as $\Gamma \vdash_\mathcal{E} M = N$ if the term M can be proved equal to the term N in

Variable: $\dfrac{x : s \in \Gamma}{\Gamma \vdash_{\mathcal{E}} x : s}$

Subject Reduction: $\dfrac{\Gamma \vdash_{\mathcal{E}} N : s \quad \Gamma \vdash_{\mathcal{E}} M = N}{\Gamma \vdash_{\mathcal{E}} M : s}$

Membership: $\dfrac{\Gamma \vdash_{\mathcal{E}} \overline{U}\sigma : \overline{t'} \quad \Gamma \vdash_{\mathcal{E}} \overline{V}\sigma = \overline{W}\sigma}{\Gamma \vdash_{\mathcal{E}} L\sigma : s}$
where $L(\overline{y}) : s$ `if` $\overline{U}(\overline{y}) : \overline{t'} \wedge \overline{V}(\overline{y}) = \overline{W}(\overline{y}) \in \mathcal{E}$

Reflexivity: $\dfrac{}{\Gamma \vdash_{\mathcal{E}} M = M}$

Symmetry: $\dfrac{\Gamma \vdash_{\mathcal{E}} M = N}{\Gamma \vdash_{\mathcal{E}} N = M}$

Transitivity: $\dfrac{\Gamma \vdash_{\mathcal{E}} M = N \quad \Gamma \vdash_{\mathcal{E}} N = P}{\Gamma \vdash_{\mathcal{E}} M = P}$

Congruence: $\dfrac{\Gamma \vdash_{\mathcal{E}} M_1 = N_1 \ldots \Gamma \vdash_{\mathcal{E}} M_n = N_n \quad \Gamma \vdash_{\mathcal{E}} f(\overline{M}) : K \quad \Gamma \vdash_{\mathcal{E}} f(\overline{N}) : K}{\Gamma \vdash_{\mathcal{E}} f(\overline{M}) = f(\overline{N})}$

Replacement: $\dfrac{\Gamma \vdash_{\mathcal{E}} \overline{U}\sigma : \overline{t'} \quad \Gamma \vdash_{\mathcal{E}} \overline{V}\sigma = \overline{W}\sigma}{\Gamma \vdash_{\mathcal{E}} L\sigma = R\sigma}$
where $L(\overline{y}) : R(\overline{y})$ `if` $\overline{U}(\overline{y}) : \overline{t'} \wedge \overline{V}(\overline{y}) = \overline{W}(\overline{y}) \in \mathcal{E}$

Figure 3.4: Equality and membership judgments

the environment Γ by using the equations and memberships in $\mathcal{E}$. Note that we implicitly make use of our convention that $\mathcal{E}$ is the theory which contains the equations in the set $\mathcal{E}$.

A term M of kind K has sort $s \in K$ in the environment Γ if $\Gamma \vdash_{\mathcal{E}} M : s$ is provable in the above inference system. A term M has a sort in the environment Γ if there exists a sort s such that M has sort s in the environment Γ. A term M has a (not necessarily unique) sort if it has a sort in some environment Γ.

3.6 Linear Proofs

An equational deduction is usually seen as a (finite) sequence of steps, each one applying an instance of an axiom at some specific subterm, resulting in a linear proof tree. We now identify this notion of proof with particular deductions obtained with our inference system. For this, we will consider each declaration $x : s \in \Gamma$ as an additional axiom in which the variable x is considered to be a constant (hence, cannot be substituted). We now introduce new judgments which will also be written as relations:

Definition 9 *Given a theory $\mathcal{E} = (\Omega, \mathcal{E})$ in membership equational logic, we define an* equational proof step *from M to N, and a* membership proof step *from M to s in an environment $\Gamma : \overline{\mathcal{V}ar(M,N)} \longrightarrow \mathcal{S}$ as follows:*

$M \longleftrightarrow_{\Gamma,\mathcal{E}} N$ *iff there exist an equational axiom $L = R$* `if` $\overline{U} : \overline{s} \wedge \overline{V} = \overline{W}$ *in $\mathcal{E}$, a* *position* $p \in \mathcal{P}os(M)$ *and a substitution* γ *such that* $M|_p = L\gamma, N = M[R\gamma]_p, \overline{U}\gamma :^*_{\Gamma,\mathcal{E}} \overline{s}$ *and* $\overline{V}\gamma \longleftrightarrow^*_{\Gamma,\mathcal{E}} \overline{W}\gamma$

$M :_{\Gamma,\mathcal{E}} s$ *iff there exist a membership axiom* $L : s$ `if` $\overline{U} : \overline{s} \wedge \overline{V} = \overline{W}$ *in* $\mathcal{E}$ *and a substitution* γ *such that* $M = L\gamma, \overline{U}\gamma :^*_{\Gamma,\mathcal{E}} \overline{s}$ *and* $\overline{V}\gamma \longleftrightarrow^*_{\Gamma,\mathcal{E}} \overline{W}\gamma$.
where the relation $:^*_{\Gamma,\mathcal{E}}$ *is defined by*[2] $S :^*_{\Gamma,\mathcal{E}} s$ *iff there exists some term* T *such that* $S \longleftrightarrow^*_{\Gamma,\mathcal{E}} T$ *and* $T :_{\Gamma,\mathcal{E}} s$.

Note that we do not need to say that the matching substitution γ is admissible, since the environment Γ assigning sorts to the variables in the term M is given a priori. γ can therefore be considered as a many $\mathcal{K}$-kinded substitution.

We see that these relations are defined as fixpoints, as is usual when there are conditional axioms. In the sequel, we will omit Γ in the previous notations when it is empty, writing instead $:^*_{\mathcal{E}}$ and $\longleftrightarrow^*_{\mathcal{E}}$.

These relations actually define linear proofs in our deduction system:

Lemma 10 $\Gamma \vdash_{\mathcal{E}} M = N$ *iff* $M \longleftrightarrow^*_{\Gamma,\mathcal{E}} N$ *and* $\Gamma \vdash_{\mathcal{E}} S : s$ *iff* $S \longleftrightarrow^*_{\Gamma,\mathcal{E}} T :_{\Gamma,\mathcal{E}} t$ *for some term* T *and sort* $t \leq_{\mathcal{E}} s$.

Proof: The if statement follows from the fact that each kind of proof step can be easily encoded as a proof tree by using the deduction rules of Figure 3.3. For the converse, we prove both kinds of statements by a simultaneous induction on the size of the proof trees, and by case analysis with respect to the rule applied at the root of the proof tree of $\Gamma \vdash_{\mathcal{E}} M = N$ or $\Gamma \vdash_{\mathcal{E}} S : s$.

Assume that $\Gamma \vdash_{\mathcal{E}} M = N$ is a proof tree of size at most n whose membership subproof trees are of size strictly smaller than m, and that $\Gamma \vdash_{\mathcal{E}} S : s$ is a proof tree of size at most m whose equational subproof trees are of size strictly smaller than n. We now exhibit an equational proof and a membership proof for both proof trees.

We do it first for the proof tree of $\Gamma \vdash_{\mathcal{E}} M = N$. If the last rule used is *Reflexivity*, the result is true. If it is *Symmetry*, we can discard it and apply induction on the remaining subproof. If it is *Transitivity*, we can apply induction on both subproofs. If it is *Congruence*, we can apply induction on the subtrees and conclude easily again. If it is *Replacement*, we simply need to apply induction on the subproofs.

We now consider the proof tree $\Gamma \vdash_{\mathcal{E}} S : s$. If the last rule applied is *Variables*, the result is clear. If it is *Subjectreduction*, then, by induction hypothesis, $S \longleftrightarrow^*_{\Gamma,\mathcal{E}} T$ for some T and $T :_{\Gamma,\mathcal{E}} t$ for some $t \leq_{\mathcal{E}} s$. If it is *Membership*, we apply induction on subproofs as previously. □

The following substitution lemma follows easily:

Lemma 11 *Assume that* $S :^*_{\Gamma,\mathcal{E}} s$ *and* $M \longleftrightarrow^*_{\Gamma,\mathcal{E}} N$, *and let* σ *be an admissible substitution of domain* $\mathcal{V}ar(S, M, N)$, *such that* $x : t \in \Gamma$ *implies* $x\sigma :^*_{\Gamma',\mathcal{E}} t$ *for some environment* Γ'. *Then* $S\sigma :_{\Gamma',\mathcal{E}} s$, *and* $M\sigma \longleftrightarrow^*_{\Gamma',\mathcal{E}} N\sigma$.

[2] Note that, despite its notation, the relation $:^*_{\Gamma,\mathcal{E}}$ is not a transitive closure since $:_{\Gamma,\mathcal{E}}$ is not a binary relation on a set.

Chapter 4

Computations

The goal of this section is to generalize rewriting techniques to achieve efficient equational deduction methods for membership equational logic. Our deduction system for rewriting computations is obtained in two steps, by eliminating the two possible sources of don't know non-determinism from the inference system described in Figure 3.4. In fact, the methods that we discuss provide an operational semantics for the efficient computation by rewriting supported by Maude for functional modules.

In all this section we will assume given a signature $\Omega = ((\mathcal{K}, \Sigma), \mathcal{S})$, and a set $\mathcal{X}$ of $\mathcal{K}$-kinded variables.

4.1 Conditional Rewriting and Membership Rules

The idea of reductive conditional rules appeared first in [58], was then generalized in [57] and again slightly in [28]. We borrow from and adapt the latter.

Definition 12 *A* conditional rewriting/membership system *(CRMS in short) is defined by two kinds of rules,* (conditional) membership rules*:*

$$L(\overline{x}) : s \texttt{ if } \overline{U}(\overline{x}) : \overline{t} \wedge \overline{V}(\overline{x}) \downarrow \overline{W}(\overline{x})$$

and (conditional) rewrite rules*:*

$$L(\overline{x}) \to R(\overline{x}) \texttt{ if } \overline{U}(\overline{x}) : \overline{t} \wedge \overline{V}(\overline{x}) \downarrow \overline{W}(\overline{x})$$

For both kinds of rules, L is called the lefthand side of the rule. Again, we will usually omit mention of the set $\overline{x}$ of variables. The set of Horn clauses obtained by replacing the arrows and joinability symbols in the rules of $\mathcal{R}$ by the equality symbol $=$, will usually be denoted by $\mathcal{E}_{\mathcal{R}}$.

A reductive *CRMS $\mathcal{R}$ has two kinds of rules,* subsort inclusion membership rules *of the form $x : s \texttt{ if } x : t$, defining the subsort quasi-ordering $\leq_{\mathcal{R}}$ as the smallest ordering generated by the set of pairs $\{s \leq_{\mathcal{R}} t \mid x : s \texttt{ if } x : t \in \mathcal{R}\}$, and* reductive *(rewrite or membership)* rules *satisfying the following* reductivity *requirement: there exists a reduction ordering $\succ$ such that*

(i) $L \notin \mathcal{X}$ for each lefthand side L of a reductive rule in $\mathcal{R}$,

(ii) $L \succ R$ for each rewrite rule $L \to R \texttt{ if } \overline{U} : \overline{t} \wedge \overline{V} \downarrow \overline{W}$ in $\mathcal{R}$,

(iii) $L\ (\succ \cup \rhd)\ U, V, W,\quad \forall U \in \overline{U}, V \in \overline{V}, W \in \overline{W}$, such that $L \to R \texttt{ if } \overline{U} : \overline{t} \wedge \overline{V} \downarrow \overline{W}$ (resp. $L : s \texttt{ if } \overline{U} : \overline{t} \wedge \overline{V} \downarrow \overline{W}$) is a rewrite (resp. membership) rule in $\mathcal{R}$.

Variable: $\dfrac{x : s \in \Gamma}{\Gamma \vdash_{\mathcal{R}} x : s}$

Subject Reduction: $\dfrac{\Gamma \vdash_{\mathcal{R}} N : s \quad \Gamma \vdash_{\mathcal{R}} M \downarrow N}{\Gamma \vdash_{\mathcal{R}} M : s}$

Membership: $\dfrac{\Gamma \vdash_{\mathcal{R}} \overline{U}\sigma : \overline{t'} \quad \Gamma \vdash_{\mathcal{R}} \overline{V}\sigma \downarrow \overline{W}\sigma}{\Gamma \vdash_{\mathcal{R}} L\sigma : s}$

where $L(\overline{x}) : s$ `if` $\overline{U}(\overline{x}) : \overline{t'} \wedge \overline{V}(\overline{x}) \downarrow \overline{W}(\overline{x})$ is a reductive membership rule in $\mathcal{R}$

Subsort: $\dfrac{\Gamma \vdash_{\mathcal{R}} M : s \quad s \leq_{\mathcal{R}} t}{\Gamma \vdash_{\mathcal{R}} M : t}$

Reflexivity: $\dfrac{}{\Gamma \vdash_{\mathcal{R}} M \longrightarrow M}$

Transitivity: $\dfrac{\Gamma \vdash_{\mathcal{R}} M \longrightarrow N \quad \Gamma \vdash_{\mathcal{R}} N \longrightarrow P}{\Gamma \vdash_{\mathcal{R}} M \longrightarrow P}$

Congruence: $\dfrac{\Gamma \vdash_{\mathcal{R}} M_1 \longrightarrow N_1 \ldots \Gamma \vdash_{\mathcal{R}} M_n \longrightarrow N_n \quad \Gamma \vdash_{\mathcal{R}} f(\overline{M}) : K \quad \Gamma \vdash_{\mathcal{R}} f(\overline{N}) : K}{\Gamma \vdash_{\mathcal{R}} f(\overline{M}) \longrightarrow f(\overline{N})}$

Replacement: $\dfrac{\Gamma \vdash_{\mathcal{R}} \overline{U}\sigma : \overline{t'} \quad \Gamma \vdash_{\mathcal{R}} \overline{V}\sigma \downarrow \overline{W}\sigma}{\Gamma \vdash_{\mathcal{R}} L\sigma \longrightarrow R\sigma}$

where $L(\overline{x}) \to R(\overline{x})$ `if` $\overline{U}(\overline{x}) : \overline{t'} \wedge \overline{V}(\overline{x}) \downarrow \overline{W}(\overline{x})$ is a reductive rewrite rule in $\mathcal{R}$

Figure 4.1: Rules of deduction for CRMS's

From now on, the notation $\mathcal{R}$ may have two slightly different meanings. When considered as a set of rules, $\mathcal{R}$ will contain the set of reductive rewrite and membership rules only, the subsort inclusion rules being incorporated in the subsort ordering $\leq_{\mathcal{R}}$. When considered as a specification, however, $\mathcal{R}$ will contain all rules, including the subsort inclusion memberships. Sometimes, as in proposition 20 and its proof, both understanding appear in a same sentence. But the context will always allow to disambiguate.

Note that conditions (i) and (ii) are redundant for rewrite rules, and that reductivity implies that the variables in a rule must occur in its lefthand side. Hence, our definition therefore forbids extra variables in the conditions of a rule, a restriction that will be removed in Section 10.1. Note also that our definition of the subsort ordering $\leq_{\mathcal{R}}$ does not involve deduction. We will see in Section 4.2 that it agrees with the semantic definition of the subsort ordering obtained by using arbitrary deductions under some natural assumptions.

The `NUMBER` specification is a particular case of a reductive system when the equations are oriented from left to right.

Given a CRMS $\mathcal{R}$, we reformulate our inference system of Figure 3.4 in Figure 4.1: by replacing equalities by rewrites and joinability on the one hand; by replacing the subsort membership rules by the subsort ordering on the other hand.

Note that subject reduction applies with $M = N$.

In order to help showing that our two inference systems are equivalent for confluent CRMS's, we first linearize the proof trees by exhibiting the corresponding rewrite relations:

Definition 13 *Let $(\Omega, \mathcal{R})$ be a theory in membership equational logic in which $\mathcal{R}$ is a reductive CRMS, M and N two $\mathcal{K}$-kinded terms such that $\mathcal{V}ar(N) \subseteq \mathcal{V}ar(M)$, $\Gamma : \mathcal{V}ar(M) \longrightarrow \mathcal{S}$ an environment in which M and N inhabit the same kind K, and s a sort of K. We then define a* rewrite proof step *from M to N, and a* membership proof step *from M to s in the environment Γ as follows:*

$M \longrightarrow_{\Gamma,\mathcal{R}} N$ iff there exist a reductive rewrite rule $L \to R$ $\texttt{if}$ $\overline{U} : \overline{s} \wedge \overline{V} \downarrow \overline{W}$ in $\mathcal{R}$, a position $p \in \mathcal{P}os(M)$ and a substitution γ such that $M|_p = L\gamma, N = M[R\gamma]_p, \overline{U}\gamma :^*_{\Gamma,\mathcal{R}} \overline{s}$ and $\overline{V}\gamma \downarrow_{\Gamma,\mathcal{R}} \overline{W}\gamma$,

$M :_{\Gamma,\mathcal{R}} s$ iff either $M : s \in \Gamma$, or there exist a reductive membership axiom $L : s$ $\texttt{if}$ $\overline{U} : \overline{s} \wedge \overline{V} \downarrow \overline{W}$ in $\mathcal{R}$ and a substitution γ such that $M = L\gamma, \overline{U}\gamma :^*_{\Gamma,\mathcal{R}} \overline{s}$ and $\overline{V}\gamma \downarrow_{\Gamma,\mathcal{R}} \overline{W}\gamma$, where the relation $:^*_{\Gamma,\mathcal{R}}$ is defined by $S :^*_{\Gamma,\mathcal{R}} s$ iff there exists some term T and sort $t \leq_{\mathcal{R}} s$ such that $S \downarrow_{\Gamma,\mathcal{R}} T$ and $T :_{\Gamma,\mathcal{R}} t$.

Given a CRMS $\mathcal{R}$, a term M of kind K, and an environment Γ as above, we say that M is

(i) reducible *if there exists a term N such that $M \longrightarrow_{\Gamma,\mathcal{R}} N$,*

(ii) sortable *if there exists $s \in K$ such that $M :_{\Gamma,\mathcal{R}} s$.*

Again, despite its notation, the relation $:^*_{\Gamma,\mathcal{R}}$ is not the reflexive transitive closure of the relation $:_{\Gamma,\mathcal{R}}$. Besides the already encountered reason that neither one is a binary relation on a set, the use of the subsort membership axioms is built in the definition of the former as a matter of convenience, but not in the definition of the latter. This will be important in the forthcoming definition of sort-decreasingness.

An empty Γ can of course be safely omitted. Later on, we will sometimes omit Γ even when $\Gamma \neq \emptyset$, if it can be inferred from the context, using therefore the simplified notations $:^*_{\mathcal{R}}$ and $\longrightarrow^*_{\mathcal{R}}$.

These relations actually define linear proofs in our deduction system of Figure 4.1:

Lemma 14 *$\Gamma \vdash_{\mathcal{R}} M \longrightarrow N$ iff $M \longrightarrow^*_{\Gamma,\mathcal{R}} N$, and $\Gamma \vdash_{\mathcal{R}} S : s$ iff $S :^*_{\Gamma,\mathcal{R}} s$.*

Proof: The proof is similar to the proof of Lemma 10. □

Since we have now defined a rewrite relation on terms, we can speak of rewriting properties such as Church-Rosser, confluence, and termination. We can now easily conclude:

Proposition 15 *Let $\mathcal{R}$ be a confluent CRMS, and $\mathcal{E}_{\mathcal{R}}$ be its associated set of Horn clauses. Then*

$$\Gamma \vdash_{\mathcal{E}_{\mathcal{R}}} M : s \texttt{ iff } \Gamma \vdash_{\mathcal{R}} M : s$$

$$\Gamma \vdash_{\mathcal{E}_{\mathcal{R}}} M = N \texttt{ iff } \Gamma \vdash_{\mathcal{R}} M \downarrow N$$

Proof: We show instead that $M :^*_{\mathcal{E}_{\mathcal{R}}} s$ iff $M :^*_{\mathcal{R}} s$ and $M \longleftrightarrow^*_{\mathcal{R}_{\mathcal{E}}} N$ iff $M \downarrow_{\mathcal{R}} N$, which is done easily by induction on the length of the equational proofs. We conclude by applying Lemmas 10 and 14. □

Our inference system does not exploit yet the full power of rewriting: the subject reduction rule involves the search, given M, for some appropriate term N. We again replace this non-directed search by an incomplete directed one based on rewriting. Figure 4.2 gives the rules of deduction for our ultimate inference system. In these rules, the superscript

Variable: $\dfrac{x : s \in \Gamma}{\Gamma \vdash^{\succ}_{\mathcal{R}} x : s}$

Subject Reduction: $\dfrac{\Gamma \vdash^{\succ}_{\mathcal{R}} N : s \quad \Gamma \vdash^{\succ}_{\mathcal{R}} M \longrightarrow N}{\Gamma \vdash^{\succ}_{\mathcal{R}} M : s}$

Membership: $\dfrac{\Gamma \vdash^{\succ}_{\mathcal{R}} \overline{U}\sigma : \overline{t'} \quad \Gamma \vdash^{\succ}_{\mathcal{R}} \overline{V}\sigma \downarrow \overline{W}\sigma}{\Gamma \vdash^{\succ}_{\mathcal{R}} L\sigma : s}$

where $L(\overline{x}) : s$ `if` $\overline{U}(\overline{x}) : \overline{t'} \wedge \overline{V}(\overline{x}) \downarrow \overline{W}(\overline{x})$ is a reductive membership rule in $\mathcal{R}$

Subsort: $\dfrac{\Gamma \vdash^{\succ}_{\mathcal{R}} M : s \quad s \leq_{\mathcal{R}} t}{\Gamma \vdash^{\succ}_{\mathcal{R}} M : t}$

Reflexivity: $\dfrac{}{\Gamma \vdash^{\succ}_{\mathcal{R}} M \longrightarrow M}$

Transitivity: $\dfrac{\Gamma \vdash^{\succ}_{\mathcal{R}} M \longrightarrow N \quad \Gamma \vdash^{\succ}_{\mathcal{R}} N \longrightarrow P}{\Gamma \vdash^{\succ}_{\mathcal{R}} M \longrightarrow P}$

Congruence: $\dfrac{\Gamma \vdash^{\succ}_{\mathcal{R}} M_1 \longrightarrow N_1 \ldots \Gamma \vdash^{\succ}_{\mathcal{R}} M_n \longrightarrow N_n \quad \Gamma \vdash^{\succ}_{\mathcal{R}} f(\overline{M}) : K \quad \Gamma \vdash^{\succ}_{\mathcal{R}} f(\overline{N}) : K}{\Gamma \vdash^{\succ}_{\mathcal{R}} f(\overline{M}) \longrightarrow f(\overline{N})}$

Replacement: $\dfrac{\Gamma \vdash^{\succ}_{\mathcal{R}} \overline{U}\sigma : \overline{t'} \quad \Gamma \vdash^{\succ}_{\mathcal{R}} \overline{V}\sigma \downarrow \overline{W}\sigma}{\Gamma \vdash^{\succ}_{\mathcal{R}} L\sigma \longrightarrow R\sigma}$

where $L(\overline{x}) \to R(\overline{x})$ `if` $\overline{U}(\overline{x}) : \overline{t'} \wedge \overline{V}(\overline{x}) \downarrow \overline{W}(\overline{x})$ is a reductive rewrite rule in $\mathcal{R}$

Figure 4.2: Rules of deduction for sort-decreasing CRMS's

in $\vdash^{\succ}_{\mathcal{R}}$ is meant to express that we replace the search for an appropriate equal term in the *Subject Reduction* rule of the inference system $\vdash_{\mathcal{R}}$ by the computation of a reduct, possibly a normal form.

In order to relate deductions in the latter inference system to rewriting, the definition of rewrite and membership proof steps must be defined accordingly. For this, we simply need a new definition of the relation $:^*_{\Gamma,\mathcal{R}}$, which becomes now $S :^*_{\Gamma,\mathcal{R}} s$ iff there exists some term T and sort $t \leq_{\mathcal{R}} s$ such that $S \longrightarrow^*_{\Gamma,\mathcal{R}} T$ and $T :_{\Gamma,\mathcal{R}} t$.

Note that the meaning of the notation $S :^*_{\Gamma,\mathcal{R}} s$ (hence of the notation $\longrightarrow^*_{\Gamma,\mathcal{R}}$) actually depends on the kind of proof we consider between arbitrary terms S and T. It was successively $S \longleftrightarrow^*_{\Gamma,\mathcal{E}} T :_{\Gamma,\mathcal{E}} s$ when considering equations, then $S \downarrow_{\Gamma,\mathcal{R}} T :_{\Gamma,\mathcal{R}} t \leq_{\mathcal{R}} s$ when interpreting equality by joins, and finally $S \longrightarrow^*_{\Gamma,\mathcal{R}} T :_{\Gamma,\mathcal{R}} t \leq_{\mathcal{R}} s$. With the exception of Proposition 18, we will always consider the latter definition in the sequel.

Lemma 16 *$\Gamma \vdash^{\succ}_{\mathcal{R}} S \longrightarrow T$ iff $S \longrightarrow^*_{\Gamma,\mathcal{R}} T$, and $\Gamma \vdash^{\succ}_{\mathcal{R}} S : s$ iff $S :^*_{\Gamma,\mathcal{R}} s$, where now, $S :^*_{\Gamma,\mathcal{R}} s$ is defined by $S \longrightarrow^*_{\Gamma,\mathcal{R}} T :_{\Gamma,\mathcal{R}} t$ for some term T and sort $t \leq_{\mathcal{R}} s$.*

Proof: The proof is similar to the proof of Theorem 10. □

In order to relate now our inference systems, we need to further assume that sorts decrease along computations in the following sense:

Definition 17 *A CRMS $\mathcal{R}$ is* sort-decreasing *(resp.* ground sort-decreasing*) if whenever $M \longrightarrow^*_{\Gamma,\mathcal{R}} N$ for some terms (resp. ground terms) M, N and $M :_{\Gamma,\mathcal{R}} s$ for some s, then $N :^*_{\Gamma,\mathcal{R}} s$.*

Note that our definition of sort-decreasingness is more general than usual, by checking whether a reduct of N inhabits the sort s, rather than N itself. In particular, this reduct can be the normal form of M. We show now that under the sort-decreasing assumption, the two inference systems are equivalent:

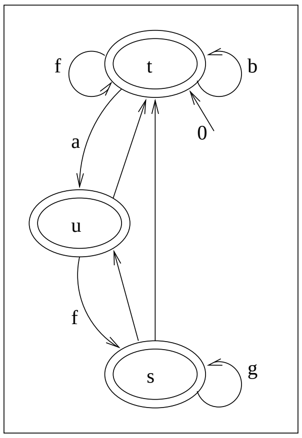
Proposition 18 *Assume that $\mathcal{R}$ is a reductive, sort-decreasing CRMS. Then, $\Gamma \vdash_{\mathcal{R}} S \longrightarrow T$ iff $\Gamma \vdash^{\succ}_{\mathcal{R}} S \longrightarrow T$, and $\Gamma \vdash_{\mathcal{R}} S : s$ iff $\Gamma \vdash^{\succ}_{\mathcal{R}} S : s$.*

Proof: We use Lemmas 14 and 16 and prove that the linear relations coincide. This is done by induction on the reduction ordering $\succ$ associated with $\mathcal{R}$. For the sake of this proof, we will use the superscript $\succ$ to single out the rewrite and membership relations introduced in Lemma 16. By induction hypothesis, we can assume that the property to be proved is satisfied for the conditions to be checked, which shows that $\longrightarrow_{\Gamma,\mathcal{R}}$ and $:_{\Gamma,\mathcal{R}}$ coincide with respectively $\longrightarrow^{\succ}_{\Gamma,\mathcal{R}}$ and $:^{\succ}_{\Gamma,\mathcal{R}}$. We are left with the relation $S :^*_{\Gamma,\mathcal{R}} s$, that is, by definition, $S \downarrow_{\Gamma,\mathcal{R}} T :_{\Gamma,\mathcal{R}} t \leq_{\mathcal{R}} s$, that is, $S \downarrow^{\succ}_{\Gamma,\mathcal{R}} T :^{\succ}_{\Gamma,\mathcal{R}} t \leq_{\mathcal{R}} s$ by the previous token. Now, $S \longrightarrow^{*,\succ}_{\Gamma,\mathcal{R}} T' :^{*,\succ}_{\Gamma,\mathcal{R}} t$ by sort-decreasingness (taking for T' the join of S and T), hence $S :^{*,\succ}_{\Gamma,\mathcal{R}} s$ as expected. □

As a corollary of Propositions 15 and 18 and Lemma 16, we obtain:

Theorem 19 *Assume that $\mathcal{R}$ is a confluent, reductive, sort-decreasing CRMS. Then, $\Gamma \vdash_{\mathcal{E}_{\mathcal{R}}} M = N$ iff $M \downarrow_{\Gamma,\mathcal{R}} N$, and $\Gamma \vdash_{\mathcal{E}_{\mathcal{R}}} S : s$ iff $S :^*_{\Gamma,\mathcal{R}} s$.*

We could have adopted a slightly different presentation of the inference system, closer to practice, by eliminating *Subject Reduction* and rewriting instead the terms $\overline{U}\sigma$ before sorting them in the conditions of *Membership* and *Replacement*. Both systems are of course



a(x) → b(x) `if` x:t

Figure 4.3: An order-sorted specification which is not sort-decreasing

equivalent in some sense: if a term M has sort s in the above system, then some reduct N of M has sort s in the modified one. The modified system eliminates a lot of redundant computations in conjunction with a bottom up evaluation strategy. This is the actual deduction system used in the Maude implementation, when no evaluation strategies are declared for the operators.

We illustrate now our inference systems with a simple example of a set of rules which is not sort-decreasing. We assume given the specification of Figure 4.3 in the form of an automaton together with a single rewrite rule. As mentioned in Section 2, we will not distinguish the transitions of the automaton from their associated membership rules. In all three inference systems given so far, $a(0)$ and $b(0)$ have respective sorts u and t, hence $f(a(0))$ has sort s, by applying the membership inference rule with the membership axiom $f(x) : s \texttt{ if } x : u$. Also, $f(a(0))$ rewrites (or is equal) to $f(b(0))$ in all three systems. As a consequence, $f(b(0))$ has sort s (and sort t as well, of course) in the inference systems $\vdash_{\mathcal{E}_\mathcal{R}}$ and $\vdash_\mathcal{R}$. In the inference system $\vdash^{\succ}_{\mathcal{R}}$, $f(b(0))$ is in normal form. Since $f(b(0))$ cannot inherit a sort from a reduct, it has the only sort t obtained by applying the membership inference rule with the membership rule $f(x) : t \texttt{ if } x : t$. As a consequence, the rewrite $f(a(0)) \longrightarrow f(b(0))$ is sort increasing in the inference system $\vdash^{\succ}_{\mathcal{R}}$, and the term $g(f(a(0)))$ is therefore in normal form, showing that rewriting is not a precongruence in this setting.

4.2 Static computation of the subsort ordering

We will now use the previous inference system for proving an important property of confluent, sort-decreasing, reductive systems: the set of valid subsort relationships $x : s \texttt{ if } x : t$ coincides with the ordering $\leq_\mathcal{R}$. This result motivates a posteriori our definition of a reductive CRMS.

Proposition 20 *Assume that $\mathcal{R}$ is a reductive, confluent, and sort-decreasing CRMS. Let s and s' be two sorts of the same kind K, and let x be a variable of kind K. Then $(x : s \texttt{ if } x : s')$ is valid in the specification $\mathcal{R}$ iff $s' \leq_\mathcal{R} s$.*

Proof: The sentence $(x : s \texttt{ if } x : s')$ is valid iff the sentence $x : s$ is satisfied in all models of the specification $\mathcal{R}$ for which $x : s'$. By the completeness theorem, this is the case iff $x : s'$ can be deduced in the environment $\{x : s'\}$, that is, iff $\{x : s'\} \vdash^{\succ}_{\mathcal{R}} x : s$, given that

$\mathcal{R}$ is reductive, confluent and sort-decreasing. The only if direction is straightforward. For the if direction, we reason by induction on the proof tree of the judgment $\{x : s'\} \vdash^{\succ}_{\mathcal{R}} x : s$. There are three cases:

- if the proof ends by using *Subject reduction*, then $\{x : s'\} \vdash^{\succ}_{\mathcal{R}} x : s$ is derived via some term M such that $\{x : s'\} \vdash^{\succ}_{\mathcal{R}} x \longrightarrow M$ and $\{x' : s'\} \vdash^{\succ}_{\mathcal{R}} M : s$. Since a variable cannot be rewritten in a decreasing system, $M = x$, and we conclude by induction;

- The proof cannot end by using a *Membership* rule, since, for reductive systems, any membership whose head is of the form $y : s$ must be a subsort declaration;

- If the proof ends by using *Subsort*, then we obtain that $\{x : s'\} \vdash^{\succ}_{\mathcal{R}} x : s''$ and $s'' \leq_{\mathcal{R}} s$. By induction hypothesis, $s' \leq_{\mathcal{R}} s''$, and by transitivity $s' \leq_{\mathcal{R}} s$. □

Assuming that each kind has a finite set of sorts, the subsort relationship can therefore be computed at compile time for confluent, sort-decreasing, reductive systems. The reader may think that there is a vicious circle here, since sort-decreasingness is defined with respect to the static subsort ordering, which is itself well-defined provided sort-decreasingness is satisfied. This is not the case. Once the static subsort ordering has been computed and sort decreasingness has been checked, then, assuming reductivity and confluence, we know from the above result that the static subsort ordering captures all subsort relations derivable from the specification. This is exactly what is done in the Maude implementation [65], where efficient Boolean representation of the subsort ordering and of the membership axioms corresponding to operator declarations are computed at compile time. This representation is of course strongly related to the automaton associated to a shallow conditional specification in membership equational logic, as discussed in Section 5.

4.3 Decidability of Equality and Membership Statements

The most important properties as far as functional computations are concerned are decidability of rewriting and of computation of normal forms, termination, and confluence, which in turn imply decidability of equality and membership statements. Among these properties, termination is satisfied for reductive CRMS's:

Proposition 21 *(Termination) A reductive CRMS is terminating.*

This is so because each lefthand side is bigger than the corresponding righthand side in the associated reduction ordering. However, this alone does not imply that normal forms can be computed, since we need to make sure that the recursive evaluation of the condition of each applied rule terminates. This is where our assumption that rewriting systems are reductive comes in. Indeed, most properties of sort-decreasing, reductive rewrite systems are decidable, and in case of confluence, make sort membership and equality properties of terms decidable with respect to the initial set of Horn clauses:

Lemma 22 *Assume that $\mathcal{R}$ is a reductive CRMS. Then $S :_{\Gamma,\mathcal{R}} s$, $S :^*_{\Gamma,\mathcal{R}} s$, $M \longrightarrow_{\Gamma,\mathcal{R}} N$, $M \longrightarrow^*_{\Gamma,\mathcal{R}} N$, $M \downarrow_{\Gamma,\mathcal{R}} N$ and $N \in M{\downarrow}_{\Gamma,\mathcal{R}}$ are decidable properties of $\mathcal{R}$.*

Proof: Routine induction on $(\succ \cup \rhd)$. □

Note that, on the one hand we require that $L \succ R$, and on the other hand that $L(\succ \cup \rhd)U, V, W$ for each $U \in \overline{U}, V \in \overline{V}, W \in \overline{W}$. An example showing that requiring only $L \succ U, V, W$ may not be enough in practice appears in [29]. Deciding equality requires another property, namely confluence:

Theorem 23 *Assume that $\mathcal{R}$ is a confluent, reductive, sort-decreasing CRMS, and let $\mathcal{E}_\mathcal{R}$ be its associated set of Horn clauses. Then $\Gamma \vdash_{\mathcal{E}_\mathcal{R}} M = N$ iff $M{\downarrow_{\Gamma,\mathcal{R}}} = N{\downarrow_{\Gamma,\mathcal{R}}}$, and $\Gamma \vdash_{\mathcal{E}_\mathcal{R}} M : s$ iff $M{\downarrow_{\Gamma,\mathcal{R}}} \; :_{\Gamma,\mathcal{R}} s$, hence equality and membership are decidable.*

Proof: Classical induction on the length of the equational proof, and use of Proposition 15, Proposition 18, and Lemma 22. □

4.4 Efficient Computations

An important property of rewriting, useful for implementation purposes, is the possibility of rewriting concurrently non-overlapping redexes. In particular, when lefthand sides of rules do not overlap, then any two redexes can usually be rewritten concurrently. In the case of conditional rules, this property assumes confluence. Unfortunately, it is no more true here: rewriting in the substitution instance σ of the lefthand side instance $L\sigma$ of a rule of left-hand side L may inhibit that rule. This is so because some condition $U\sigma : t'$ may no more be true after rewriting in σ. But this happens only when rewriting may increase sorts, allowing us to have a generalized substitution lemma under these assumptions:

Lemma 24 *Assume that $\mathcal{R}$ is a confluent sort-decreasing CRMS such that $M \longrightarrow_{\Gamma,\mathcal{R}} N$, and let σ be an admissible substitution of domain $\mathcal{V}ar(M,N)$, with $x : t \in \Gamma$ implies $x\sigma :^*_{\Gamma',\mathcal{R}} t$ for some environment $\Gamma' : \bigcup_{\mathcal{D}om(\sigma)} \mathcal{V}ar(x\sigma) \longrightarrow \mathcal{S}$, and such that $\sigma \longrightarrow^*_{\Gamma,\mathcal{R}} \tau$. Then, $M\sigma \longrightarrow^*_{\Gamma',\mathcal{R}} M\tau \longrightarrow N\tau$.*

Proof: Because an instantiated term $M\tau$ satisfies all sort memberships that M satisfies. □

Again, we can use the substitution lemma directly on derivations.

In practice, however, it is useful to adopt a bottom-up evaluation strategy which allows obtaining reduced substitutions when matching a lefthand side of a rule. Having reduced substitutions is helpful if their sorts have been computed once and for all, and have been stored in the term structure. But having multiple sorts for a given term can be the source of a combinatorial explosion when checking the sort memberships in the condition of a rule, although Maude uses a very efficient implementation by Boolean vectors for this purpose. It is therefore most interesting to have a kind of unique sort property:

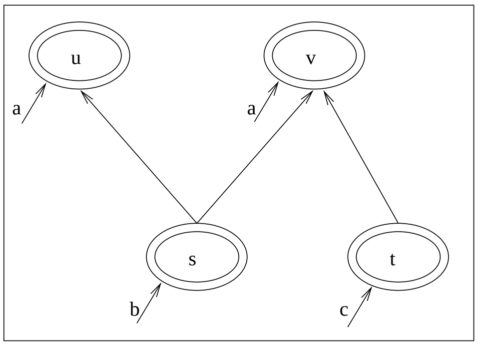
Definition 25 *A specification $(\Omega, \mathcal{R})$ is* regular *if for each pair (Γ, S), the set $\{s \in \mathcal{S} \mid S :^*_{\Gamma,\mathcal{R}} s\}$ is either empty or else has a minimal element with respect to the subsort ordering $\leq_\mathcal{R}$.*

A similar property plays an essential role in the various extensions of Girard's system F using subtypes, as well as in OBJ [32].

In practice, we prefer reducing the property to terms in normal form in order to be able to use the memberships without further computations:

Definition 26 *A specification $(\Omega, \mathcal{R})$ is* weakly regular *if for each pair (Γ, S) such that the term S is in normal-form in the environment Γ, the set $\{s \in \mathcal{S} \mid S :_{\Gamma,\mathcal{R}} s\}$ is either empty or else has a minimal element with respect to the subsort ordering $\leq_\mathcal{R}$.*

Although we have restricted our attention to normal forms, the minimal sort property extends to all terms under the assumption that $\mathcal{R}$ is confluent and sort-decreasing:



a → b if a : u
a → c if a : u

Figure 4.4: Regularity and weak-regularity versus strong regularity

Lemma 27 *Assume $\mathcal{R}$ is a confluent, sort-decreasing CRMS. Then, it is regular iff it is weakly regular.*

Proof: First, notice that regularity implies weak-regularity. For the converse, let $S \longrightarrow^*_{\Gamma,\mathcal{R}} T :_{\Gamma,\mathcal{R}} t$ and $S \longrightarrow^*_{\Gamma,\mathcal{R}} U :_{\Gamma,\mathcal{R}} u$ such that t and u are both minimal. By confluence $T \downarrow_{\Gamma,\mathcal{R}} = U \downarrow_{\Gamma,\mathcal{R}}$. By sort-decreasingness and minimality of t and u, $T \downarrow_{\Gamma,\mathcal{R}} :_{\Gamma,\mathcal{R}} t$ and $U \downarrow_{\Gamma,\mathcal{R}} :_{\Gamma,\mathcal{R}} u$. By weak-regularity, $t = u$. □

A stronger version of weak-regularity appears to be a localization of regularity:

Definition 28 *A specification $(\Omega, \mathcal{R})$ is* strongly regular *if for each pair (Γ, S), the set $\{s \in \mathcal{S} \mid S :_{\Gamma,\mathcal{R}} s\}$ is either empty or else has a minimal element with respect to the subsort ordering $\leq_{\mathcal{R}}$.*

It is indeed surprising to notice that strong regularity is not a particular case of regularity as it seems to be the case. Indeed, strong regularity and regularity both imply weak regularity, but are unrelated among themselves. Consider the specification of Figure 4.4 which is weakly regular, but neither regular nor strongly regular. Removing the rule $a \to c$ **if** $a : u$ makes it regular, but not strongly regular. Removing instead the membership $a : v$ makes it strongly regular, but non-regular.

Relating regularity and strong regularity needs therefore additional assumptions about $\mathcal{R}$:

Proposition 29 *Assume $\mathcal{R}$ is a confluent, sort-decreasing, strongly regular CRMS. Then, it is regular.*

Proof: Because strong regularity implies weak regularity and use of Lemma 27. □

There are indeed three good reasons to promote strong regularity as the main notion. The first is that it yields a natural notion of critical pairs between membership rules, as we will see later. The second is practical. When $\mathcal{R}$ is sort-decreasing, computing normal forms for sort memberships is not necessary in practice. It suffices to compute enough to be able to satisfy the necessary sort memberships which allow making progress in the computation. Then, it is best if all sortable terms have a unique minimal sort with respect to $:_{\Gamma,\mathcal{R}}$, because this allows us to maintain a sort information associated with each term all along the computation. This stronger notion of weak regularity appeared already in OBJ for the case of order-sorted equational logic, and is the one used in Maude. The third is that strong regularity allows an increase in efficiency for arbitrary computation strategies, which is not true of regularity. The point is that the truth of a membership statement $U : s$ in the condition of a rule necessitates the existence of a membership whose head matches U. If this is not the case, then we know that $U : s$ cannot be true, therefore allowing us to avoid wasting time in normalizing the condition of the rule.

In Maude [78, 65], rewriting is not only seen as a model of functional computation, but also as a rewriting logic model of concurrent computations [72], in which (concurrent) rewrite rules are meant as expressing transitions between states. In this model, confluence is not a requirement. On the other hand, it assumes the parallel execution of the concurrent computations. In particular, rules that do not overlap, hence have nothing to do with each other, should be rewritten concurrently. We can see that this will require confluence and sort-decreasingness of the rules used for evaluating the conditions. These rules should therefore operate on the underlying data structure, for which a convergent rewrite system is assumed.

We will now use our rewriting notations only. We will sometimes superscript the arrow by the position at which the rewrite takes place. We may sometimes omit Γ, $\mathcal{R}$, or both, if there is no ambiguity. In most cases, we will ease (and overload) our notations by using the set of rules $\mathcal{R}$ to denote the theory $(\Omega, \mathcal{R})$ itself.

Chapter 5

Bottom-up Tree Automata

As we have already seen, many-sorted signatures are bottom-up tree automata, in which sorts become states, and signature declarations become transitions. For the case of order-sorted signatures, subsort declarations become empty transitions. Since signatures and subsort declarations are membership axioms of a particularly simple form, a natural question is whether more complex axioms can be encoded as transitions of the automaton. The answer is positive for non-conditional left-linear rewrite rules provided that they define unique normal forms [21]. Accounting for non-left linear rules is possible, at the price of using a more general form of automaton, called *bottom-up tree automaton with equality tests* [16]. We will see that these automata allow us to encode non-conditional memberships as well. We start recalling the definition of a non-deterministic tree automaton with empty transitions.

Definition 30 *A non-deterministic* bottom-up tree automaton *with empty transitions, or simply an* automaton*, is a quadruple* $(\mathcal{S}, \leq_{\mathcal{R}}, \Sigma, \mathcal{F})$*, where* $(\mathcal{S}, \leq_{\mathcal{R}}, \Sigma)$ *is an order-sorted signature whose sorts are called* states*, whose membership declarations are called* transitions*, and whose subsort declarations are called* empty transitions*.* $\mathcal{F}$ *is a subset of* $\mathcal{S}$ *whose elements are called* accepting states*.*

Recognizing a term T is done by rewriting T according to the transitions, as already explained:

Definition 31 *To an automaton* $\mathcal{A} = (\mathcal{S}, \leq_{\mathcal{R}}, \Sigma, \mathcal{F})$*, we associate a many-sorted signature* $(\mathcal{S}, \Sigma_{\mathcal{A}})$ *and a rewrite system* $\mathcal{R}_{\mathcal{A}}$ *over the signature* $\Sigma_{\mathcal{A}}$*:*

$$\begin{array}{c} \Sigma_{\mathcal{A}} = \{f_{\overline{s},s} : \overline{s} \to s\}_{f:\overline{s}\to s\in\Sigma} \cup \{s : s \to s\}_{s\in\mathcal{S}} \\ \mathcal{R}_{\mathcal{A}} = \{f_{\overline{s},s}(s_1(x_1), \ldots, s_n(x_n))) \to s(f(x_1, \ldots, x_n))\}_{f:\overline{s}\to s\ \in\Sigma} \\ \cup \\ \{s(x) \to t(x)\}_{(s,t)\in\leq_{\mathcal{R}}} \end{array}$$

A term $T \in \mathcal{T}_{\Sigma}(\emptyset)$ *is said to be* recognized *by the automaton if it rewrites with the rules in* $\mathcal{R}_{\mathcal{A}}$ *to a term of the form* $s(T)$ *where* s *is an accepting state. We also say that* T inhabits *the sort* s*.*

Bottom up tree automata are closed under Boolean operations, determinization and cylindrification, their emptiness problem is decidable, and they can encode order-sorted specifications whose axioms are left linear rules [50]:

Theorem 32 *Let* $(\Sigma, \mathcal{R})$ *be an order-sorted specification for which* $\mathcal{R}$ *is a set of left linear rewrite rules. Then, there exists a (non-deterministic) bottom-up tree automaton* $\mathcal{A}_{\mathcal{R}}$*, called the* normal form *automaton of* $(\Sigma, \mathcal{R})$ *such that:*

(i) Each $\mathcal{R}$*-irreducible ground term* S *is recognized at an accepting state* u *of the automaton, such that* s *is accessible from* u *by empty transitions iff* $S :_{\mathcal{R}} s$*.*

(ii) Each $\mathcal{R}$*-reducible ground term* T *is recognized at the non-accepting state* t *of the automaton iff* $T :_{\mathcal{R}} t$*. Besides, if* $\mathcal{R}$ *is sort decreasing, then the normal form* S *of* T *is recognized at a state* t' *such that* t *is accessible from* t' *by empty transitions.*

To our knowledge, the remark that the language of ground terms in normal form is recognizable is due to Gallier and Book for the simple case of many-sorted specifications [34], and to Comon for the more general case of order-sorted specifications [20, 21]. We give below the sketch of a simple proof inspired by [21].

Proof: The simplest proof makes heavy use of closure properties of bottom-up tree automata. Let $\mathcal{A}_\Sigma$ be the automaton associated with the signature Σ. It is easy to construct an automaton which recognizes the set of sortable terms which contain as a subterm an instance of a given term T in an environment Γ (T may be an error-term in Γ), let us call it $\mathcal{A}_{T,\Gamma}$. For this, we first remove all accepting states from $\mathcal{A}_\Sigma$ before adding the subterms of T as new states, T itself being the single accepting state. We then encode each declaration $x : s \in \Gamma$ as a transition from the state s to the new state x, the term structure as appropriate labelled transitions between the new states, and containment as appropriate labelled transitions originating from and ending in T.

Given a set of rules $\mathcal{R}$ whose set of lefthand sides along with their associated environment is $\{L_i, \Gamma_i\}_i$ we can compute the automaton $\bigvee_{T \in \{L_i, \Gamma_i\}_i} \mathcal{A}_{T,\Gamma}$ recognizing the set of sortable terms which are reducible by R. The set of sortable irreducible terms is then obtained as its complement with respect to $\mathcal{A}_\Sigma$. $\Box$

Of course, this construction is not quite satisfactory in several respects. First, it does not achieve the optimal EXPTIME complexity [30]. Second, it messes up the structure of the original specification completely. An optimal algorithm which respects the structure of the original order-sorted specification is given in [79]. It is based on a result of Comon and Delor [23].

A normal form automaton has several kinds of states: (i) the states inhabited by irreducible ground terms only; (ii) the states inhabited by reducible ground terms only; (iii) and the other states which are inhabited by terms of both kinds. The latter states are actually states of the original automaton from which the normal form automaton is computed.

Definition 33 *A sort* s *is said to be* free *iff all ground terms inhabiting* s *are irreducible.*

Freeness makes induction easy on the corresponding sorts. On the other hand, non-free sorts can be decomposed into free ones:

Definition 34 *(Cover Sort) Given a sort* s*, a set* $s_\leq$ *of free subsorts of* s *is a* cover sort *of* s *if every irreducible ground constructor term* T *inhabiting* s *inhabits a unique sort in* $s_\leq$*. A sort* s *is* minimal *if the only cover sort of* s *is* $\{s\}$*. Cover sorts are extended to Cartesian products of sorts as expected.*

Proposition 35 *Given an automaton* $\mathcal{A}$ *and an associated normal-form automaton* $\mathcal{A}_{nf}$ *extending* $\mathcal{A}$*, each sort* s *of* $\mathcal{A}$ *is either free, or it has a cover sort in* $\mathcal{A}_{nf}$*.*

Proof: We need only to care about those sorts s inhabited by both irreducible and reducible ground terms. By construction of the normal form automaton, free subsorts of s are added to $\mathcal{A}$ in such a way that an irreducible term inhabiting s will inhabit a unique free subsort of s, see [30, 79] for details.

Up to now, we have only considered ground terms. The same observations apply as well to terms with variables, therefore relating normal form automata with the notions of ground-reducibility and ground-irreducibility [54]:

Definition 36 *Given a CRMS $\mathcal{R}$, a term T is said to be* ground reducible *(resp.* ground irreducible, ground sortable*) in an environment Γ if $T\gamma$ is reducible (resp. irreducible, sortable) for each irreducible admissible ground substitution γ. We also say that (T,Γ) is ground reducible (resp. irreducible, sortable).*

Ground-irreducible terms will inhabit the free sorts of the normal form automaton, provided, of course, that the associated environment associating sorts to their variables are encoded in the normal form automaton as transitions to these sorts when reading these variables. Under the same assumption, ground-reducible terms will of course inhabit the sorts of type (ii), while a term having both reducible and irreducible instances will inhabit a sort of type (iii), therefore a sort of the original automaton $\mathcal{A}$.

We now relate normal form automata and induction via initial algebras. When the normal form algebra is initial, which is the case when the set of rules enjoys the unique normal form property, which is the case here if the rewrite relation is confluent, then the automaton can be seen as a realization of the initial algebra:

Corollary 37 *Let $(\Sigma,\mathcal{R})$ be an order-sorted specification for which $\mathcal{R}$ is a set of left linear rewrite rules. Assume further that each ground term has a unique normal form with respect to $\mathcal{R}$. Then, the ground terms accepted by the normal form automaton of $(\Sigma,\mathcal{R})$ define an order-sorted algebra, called the* canonical term algebra *of $\mathcal{R}$ that is initial among all Σ-algebras that are models of $(\Sigma,\mathcal{R})$.*

We now give a simple example of specification of integers together with its associated automaton. We have added some syntactic sugar, such as omitting the name of the kind, and declaring arity and coarity sorts for operators instead of giving the corresponding membership assertions. This specification will be used in later sections to show the adequacy of membership equational logic for inductive reasoning. The algorithm described in [79] yields the automaton shown in Figure 5.1, where all states are accepting, except *Int*.

What is interesting about this construction and its variants is that the resulting specification has axioms corresponding to subsort declarations, which cannot be expressed in many-kinded logic without adding new function symbols. The question now arises whether other axioms of membership equational logic can also be expressed by the formalism of automata. A positive answer can be obtained from [16]:

Definition 38 *A* bottom-up tree automaton with membership/equality tests *is a quintuple $(\mathcal{S},\leq_{\mathcal{R}},\Sigma,\mathcal{F},\mathcal{R})$, such that $(\mathcal{S},\leq_{\mathcal{R}},\Sigma,\mathcal{F})$ is a bottom-up tree automaton, and $\mathcal{R}$ is a set of* conditional transitions *of the form*

$$f(s_1(x_1),\ldots,s_n(x_n)) \rightarrow s(f(x_1,\ldots,x_n))$$
$$\texttt{if}$$
$$\bigwedge_{i\in[1..k]} p_i = q_i \wedge \bigwedge_{i\in[1..l]} p'_i \neq q'_i \wedge \bigwedge_{i\in[1..m]} p''_i : s \wedge \bigwedge_{i\in[1..n]} \neg\, q''_i :\ s$$

```
fmod INTEGERS
sorts Nat<Int
op S_,P_ : Int → Int
op S_ : Nat → Nat
op 0 : Nat
eq P(S(x)) → x if x : Int
eq S(P(x)) → x if x : Int
end fmod
```

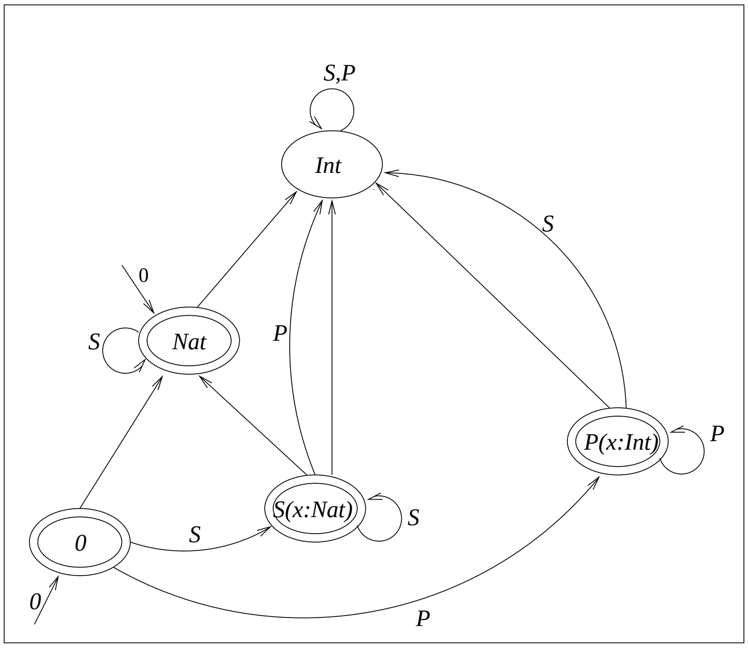


Figure 5.1: An order-sorted specification of integers and its associated normal form automaton

where $p_i, q_i, p'_i, q'_i, p''_I, q''_i$ *denote non-empty positions in (the instance of) the term* $f(x_1, \ldots, x_n)$*.*

Although membership conditions as above are not considered in [16], they can be added without changing the formalism. An example is given in Section 10, where membership conditions are used for the constructors themselves.

The intuitive meaning of these additional transitions is that the automaton has to verify conditions on the terms recognized so far at states in $\overline{s}$ before applying the transition from $\overline{s}$ to s labelled by the function symbol f. These runs will again be defined by a many-sorted rewrite system:

Definition 39 *To a bottom up tree automaton with membership/equality tests* $\mathcal{A} = (\mathcal{S}, \Sigma, \leq_{\mathcal{R}}, \mathcal{F}, \mathcal{R})$*, we associate a many-sorted signature* $(\mathcal{S}, \Sigma_{\mathcal{A}})$ *such that* $\Sigma_{\mathcal{A}} = \{f_{\overline{s},s} : \overline{s} \to s\}_{f:\overline{s}\to s \in \Sigma} \cup \{s : s \to s\}_{s\in\mathcal{S}}$*, and a reductive CRMS* $\mathcal{R}_{\mathcal{A}}$ *over the signature* $\Sigma_{\mathcal{A}}$ *defined as follows:*

(i) to each empty transition from s *to* t*, we associate the rule* $s(x) \to t(x)$*,*

(ii) To each non-empty transition (conditional or not) of the form

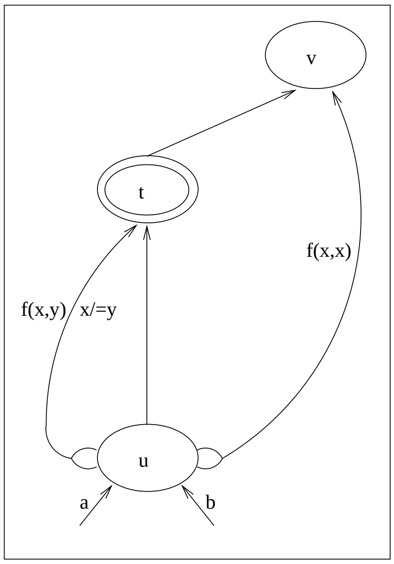
$$f(\overline{s}) \to s \ \mathtt{if} \bigwedge_{i\in[1..k]} p_i = q_i \wedge \bigwedge_{i\in[1..l]} p'_i \neq q'_i \wedge \bigwedge_{i\in[1..m]} p''_i : s \wedge \bigwedge_{i\in[1..n]} q''_i \ /: s$$

we associate the rule schema

$$f_{\overline{s},s}(s_1(x_1), \ldots, s_n(x_n))) \to s(f(x_1, \ldots, x_n)) \ \mathtt{if}$$
$$\bigwedge_{i=1}^{k} f(\overline{x})|_{p_i} = f(\overline{x})|_{q_i} \wedge \bigwedge_{i=1}^{l} f(\overline{x})|_{p'_i} \neq f(\overline{x})|_{q'_i} \wedge \bigwedge_{i=1}^{m} f(\overline{x})|_{p''_i} : s \wedge \bigwedge_{i=1}^{n} f(\overline{x})|_{q''_i} \ /: s$$

A term $T \in \mathcal{T}_\Sigma(\emptyset)$ is said to be recognized *by the automaton if it rewrites with the rules in $\mathcal{R}_\mathcal{A}$ to a term of the form $s(T)$ where s is an accepting state. We also say that T* inhabits *the sort s.*

The equality and membership conditions arising in the above rule schema cannot be defined concisely in the syntax of many-sorted logic, since it would require axiomatizing the data type of terms together with the appropriate defined functions (among which subterm at a given position). It is mathematically easier to define directly the (infinite) set of rules operating on ground terms via a rule schema. The corresponding ground rules are obtained by replacing the variables in $\overline{x}$ by ground terms that satisfy the conditions. In particular, these ground terms must have subterms at the positions indicated in the conditions. An example is provided in Figure 5.2.



Signature:

$$\begin{array}{lclcl} s & : & & \to & s \\ t & : & & \to & t \\ v & : & & \to & v \\ a & : & & \to & s \times s \\ b & : & & \to & s \times s \\ f & : & (s \times s) \times (s \times s) & \to & t \times t \\ f & : & (s \times s) \times (s \times s) & \to & v \times v \end{array}$$

Rules:

$$\begin{array}{rcl} s & \to & t \\ t & \to & v \\ a & \to & (a, s) \\ b & \to & (b, s) \\ f((x,s),(y,s)) & \to & (f((x,s),(y,s)),t) \texttt{ if } x \neq y \\ f((x,s),(y,s)) & \to & (f((x,s),(y,s)),v) \texttt{ if } x = y \end{array}$$

Example of a successful run:

$f(a,b) \to f((a,s),b) \to f((a,s),(b,s)) \to (f((a,s),(b,s)),t)$

Example of a non-successful run:

$f(a,a) \to f((a,s),a) \to f((a,s),(a,s)) \to (f((a,s),(a,s)),v)$

Figure 5.2: Automaton with membership/equality tests

Tree automata with membership/equality tests can of course encode non-left linear rules that could not be taken care of by usual bottom-up tree automata, and they were actually designed for this purpose. But they can do more than that: they can encode as well *partial operators*, that is, operators that are defined on a subset of their domain specified by means of equations and memberships of a simple kind that we proceed to define now:

Definition 40 *A specification $(\Omega, \mathcal{R})$ in membership equational logic is said to be* shallow conditional *if:*

(i) its rules and memberships are of the three following forms:

$$\begin{array}{lrll} (a) & x : s & \texttt{if} & x : t \\ (b) & L : s & \texttt{if} & \overline{x} : \overline{t} \wedge \overline{y} = \overline{y'} \wedge \overline{z} \neq \overline{z'} \wedge \overline{w} \neq \overline{G}, \text{ where } L \notin \mathcal{X} \\ (c) & L \to R & \texttt{if} & \overline{x} : \overline{t} \wedge \overline{y} = \overline{y'} \wedge \overline{z} \neq \overline{z'} \wedge \overline{w} \neq \overline{G}, \text{ where } L \notin \mathcal{X} \end{array}$$

where L and R are many-kinded terms of kind K, s is a sort of kind K, $\overline{x}, \overline{y}, \overline{z}, \overline{w}$ are vectors of many-kinded variables occurring in L, and $\overline{G}$ is a vector of ground terms.

(ii) the rewrite relation generated by $\mathcal{R}$ is confluent and sort-decreasing.

(iii) the rewrite relation generated by the rules $L \to R$ obtained from the rules of type (c) by removing their condition is terminating.

Some explanation is in order about the negative conditions in the memberships and rewrite rules of a shallow conditional specification, which, strictly speaking, bring such a specification outside the class of Horn specifications. However, shallow conditional specifications can always be seen as abbreviated descriptions of somewhat lengthier Horn specifications, in which equality has been equationaly axiomatized. First of all, we must explain the operational meaning of the inequalities in a shallow conditional specification. Operationally we compute with such specifications as usual, except that inequalities between ground terms are settled by bringing both sides to normal form and checking whether the normal forms are syntactically different. Because of the terminating, sort-decreasing and confluent properties of shallow specifications, comparison for syntactic equality after reduction to normal form is a computable equality predicate. Therefore, by general results of Bergstra and Tucker [5], such computable equality predicates (one for each kind) can be equationally axiomatized by a finite set of Church-Rosser and terminating many-kinded equations, just by adding a new kind for the Booleans, and some auxiliary operations to define the equality predicates for each kind. We can then replace each inequality $U \neq V$ in a membership or a rule of our original shallow specification $(\Omega, \mathcal{R})$ by an equality $(U \equiv_K V) = false$, where $\_ \equiv_K \_$ denotes the equationally-defined equality predicate of kind K. In this way, combining the transformed axioms of $\mathcal{R}$ with the equational axioms for the equality predicates, we can obtain an equivalent Horn specification $(\Omega^b, \mathcal{R}^b)$ that is terminating, sort-decreasing, and confluent, where Ω^b is obtained from Ω by adding a new kind for Booleans, and such that for any two ground terms t and t' of the same kind, we have:

(i) $\mathcal{R}^b \vdash t = t'$ iff $\mathcal{R}^b \vdash (t \equiv_K t') = true$ iff $t{\downarrow_{\mathcal{R}}} = t'{\downarrow_{\mathcal{R}}}$

(ii) $\mathcal{R}^b \vdash (t \equiv_K t') = flase$ iff $t{\downarrow_{\mathcal{R}}} \neq t'{\downarrow_{\mathcal{R}}}$

(iii) $\mathcal{R}^b \vdash t : s$ iff $t{\downarrow_{\mathcal{R}}} : s$

where, as usual, $t{\downarrow_{\mathcal{R}}}$ denotes the $\mathcal{R}$-normal form of t.

We can now generalize the above results:

Theorem 41 *Let $(\Omega, \mathcal{R})$ be a shallow conditional specification in membership equational logic. Then, there exists a bottom-up tree automaton $\mathcal{A}$, called the* normal form *automaton of $(\Sigma, \mathcal{R})$ which recognizes the set of ground terms on the signature Ω which are in normal form with respect to $\mathcal{R}$.*

Proof: Our previous proof generalizes to this case without difficulty.

Corollary 42 *Let $(\Omega, \mathcal{R})$ be a shallow-conditional specification in membership equational logic. Then, the ground terms accepted by the normal form automaton of $(\Omega, \mathcal{R})$ define a membership algebra, called the* canonical term algebra *of $\mathcal{R}$ that is initial among all Ω^b-algebras that are models of $(\Omega^b, \mathcal{R}^b)$.*

Note that, according to the definition of a shallow conditional specification $\Omega, \mathcal{R})$, each ground term has a unique normal form with respect to $\mathcal{R}$.

Given a specification in membership equational logic, it is therefore possible to compile its largest shallow conditional subspecification in the form of an automaton so as to efficiently decide the sort of an irreducible term. The same technique will of course apply to reducible terms by first reducing them to normal form in a bottom-up way.

There is however a slight difficulty: the bottom-up automata with membership/equality test that we have defined may have an undecidable emptiness problem. Restricted classes suffice however for all practical purposes, for which emptiness is decidable [16], and can even be achieved efficiently [6].

Another problem arises with the axioms of associativity, commutativity, idempotency and identity that are often built in the rewriting process by using an adequate pattern matching algorithm. The same kind of automata as above allows handling of this case, by using formulae of Presburger's arithmetic to control the transitions [64] when some rules are not left linear rules, and by non-deterministically using commutativity along transitions of the automaton. A first implementation has been recently developed to experience with this automata [79].

Chapter 6

Confluence Properties

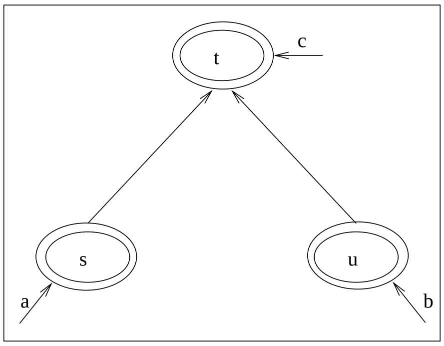
The central notion of Church-Rosser property is somewhat more subtle that in a total framework, because it has to take into account all aspects of the computation, namely equalities *and* memberships. A main principle is that rewriting should be monotonic with respect to checking validity of the equalities and memberships, by allowing us to blindly compute normal forms before doing any checking. Here, we want to be able to have rewrite proofs for arbitrary proofs, whether or not the terms involved have sorts. However, if they do have sorts, we also want sorts to decrease along rewrite derivations, allowing us to check both equality and sort-membership via the computation of normal forms and of sorts of normal forms. This suggests the following definition:

Definition 43 *A specification $\mathcal{R}$ is Church-Rosser iff $\forall \Gamma, S, T$ such that $S \longleftrightarrow^*_{\Gamma,\mathcal{R}} T :_{\Gamma,\mathcal{R}} s$, then:*
(i) $S \downarrow_{\Gamma,\mathcal{R}} T$,
(ii) $S :^*_{\Gamma,\mathcal{R}} s$.

Note that taking $T :^*_{\Gamma,\mathcal{R}}$ instead of $T :_{\Gamma,\mathcal{R}}$ yields an equivalent definition.

As in many other cases, such as rewriting modulo [53], the confluence and Church-Rosser properties do not coincide. For example, the specification of Figure 6.1 is confluent but not Church-Rosser: although a and b can be proved equal, we cannot rewrite $a : s$ to its normal form $c : s$, because the property that a is of sort s has then been lost once it is rewritten to $c : t$.

Theorem 44 *A specification $\mathcal{R}$ is Church-Rosser iff it is confluent and sort-decreasing.*



a	$\to$	c
b	$\to$	c

Figure 6.1: A specification which is not Church-Rosser

Proof: Routine induction on the length of proofs to show property (i) of Church-Rosser from confluence, and then application of sort-decreasingness to show (ii). Conversely, confluence and sort-decreasingness are particular cases of our Church-Rosser property. □

6.1 Local properties

As usual, we localize our properties.

Definition 45 *Given a specification $(\Omega, \mathcal{R})$, we say that the CRMS $\mathcal{R}$ is*
locally confluent *if $U \longrightarrow_{\Gamma,\mathcal{R}} S$ and $U \longrightarrow_{\Gamma,\mathcal{R}} T$ implies the existence of some V such that $S \longrightarrow^*_{\Gamma,\mathcal{R}} V$ and $T \longrightarrow^*_{\Gamma,\mathcal{R}} V$, and*
locally sort-decreasing *if $S :_{\Gamma,\mathcal{R}} s$ and $S \longrightarrow_{\Gamma,\mathcal{R}} T$ implies that $T :^*_{\Gamma,\mathcal{R}} s$,*

We then have the analog of Newman's lemma:

Lemma 46 *A reductive CRMS $\mathcal{R}$ is Church-Rosser iff it is locally confluent and locally sort-decreasing.*

Proof: Since Newman's lemma implies confluence from local confluence for terminating systems, we are left to show that local sort-decreasingness implies sort-decreasingness for a confluent CRMS. This is done by induction on the rewrite relation.

Assume $S :_{\Gamma,\mathcal{R}} s$ and $S \longrightarrow_{\Gamma,\mathcal{R}} T \longrightarrow^*_{\Gamma,\mathcal{R}} U$. By assumption, $T \longrightarrow^*_{\Gamma,\mathcal{R}} V :_{\Gamma,\mathcal{R}} s$. By confluence, there exist a term W such that $V \longrightarrow^*_{\Gamma,\mathcal{R}} W$ and $U \longrightarrow^*_{\Gamma,\mathcal{R}} W$. By induction hypothesis applied to T, $W :^*_{\Gamma,\mathcal{R}} s$, and therefore $U :^*_{\Gamma,\mathcal{R}} s$. □

6.2 Critical Inferences

We now characterize confluence, sort-decreasingness and regularity by means of critical inference steps of several kinds:

Definition 47 *Given two conditional rewrite rules $L \to R$ $\texttt{if}$ $\overline{U} : \overline{s'} \wedge \overline{V} \downarrow \overline{W}$ and $G \to D$ $\texttt{if}$ $\overline{U'} : \overline{t'} \wedge \overline{V'} \downarrow \overline{W'}$ such that $\mathcal{V}ar(L) \cap \mathcal{V}ar(G) = \emptyset$ and $L|_p\sigma = G\sigma$, for some non-variable position $p \in \mathcal{FP}os(L)$ and most general (many-kinded) unifier σ, then*

$$L\sigma[D\sigma]_p = R\sigma \ \texttt{if} \ \overline{U}\sigma : \overline{s'} \wedge \overline{U'}\sigma : \overline{t'} \wedge \overline{V}\sigma \downarrow \overline{W}\sigma \wedge \overline{V'}\sigma \downarrow \overline{W'}\sigma$$

is a critical pair*.*
Given a conditional membership rule $L : s$ $\texttt{if}$ $\overline{U} : \overline{s'} \wedge \overline{V} \downarrow \overline{W}$ and a conditional rewrite rule $G \to D$ $\texttt{if}$ $\overline{U'} : \overline{t'} \wedge \overline{V'} \downarrow \overline{W'}$ such that $\mathcal{V}ar(L) \cap \mathcal{V}ar(G) = \emptyset$ and $L|_p\sigma = G\sigma$ for some non-variable position $p \in \mathcal{FP}os(L)$ and most general unifier σ, then

$$L\sigma : s \to L\sigma[D\sigma]_p \ \texttt{if} \ \overline{U}\sigma : \overline{s'} \wedge \overline{U'}\sigma : \overline{t'} \wedge \overline{V}\sigma \downarrow \overline{W}\sigma \wedge \overline{V'}\sigma \downarrow \overline{W'}\sigma$$

is a critical reduced membership*.*
Given two conditional membership rules $L : s$ $\texttt{if}$ $\overline{U} : \overline{s'} \wedge \overline{V} \downarrow \overline{W}$ and $G : t$ $\texttt{if}$ $\overline{U'} : \overline{t'} \wedge \overline{V'} \downarrow \overline{W'}$ such that $\mathcal{V}ar(L) \cap \mathcal{V}ar(G) = \emptyset$ and $L\sigma = G\sigma$ for some most general unifier σ, then

$$L\sigma : s, t \ \texttt{if} \ \overline{U}\sigma : \overline{s'} \wedge \overline{U'}\sigma : \overline{t'} \wedge \overline{V}\sigma \downarrow \overline{W}\sigma \wedge \overline{V'}\sigma \downarrow \overline{W'}\sigma$$

is a critical membership*.*

These definitions need several remarks. Firstly, remember that subsort membership rules do not belong to $\mathcal{R}$, since they are taken care of by the subsort ordering. Anyhow, unifying a subsort membership rule $x : s'$ `if` $x : s$ with a rewrite rule would not produce any interesting critical reduced membership, since the sort of the same lefthand side L would be increased. Secondly, critical reduced memberships obtained by overlapping G on L at the root were already used in OBJ to check for sort decreasingness [41]. Finally, and most importantly, note that we use plain many-kinded unification in the definition of our critical pairs and memberships, but also that we accumulate the conditions of both rules and/or memberships they originate from. We will come back to this issue at the end of this section.

Definition 48 *Given a CRMS $\mathcal{R}$, a critical pair $P = Q$ `if` $\overline{U} : \overline{s} \wedge \overline{V} \downarrow \overline{W}$ is* confluent *if $P\gamma \downarrow_{\mathcal{R}} Q\gamma$ for all substitutions γ such that $\overline{U}\gamma :_{\mathcal{R}} \overline{s}$ and $\overline{V}\gamma \downarrow_{\mathcal{R}} \overline{W}\gamma$.*

A critical reduced membership $L : s \to R$ `if` $\overline{U} : \overline{s} \wedge \overline{V} \downarrow \overline{W}$ is sort-decreasing *if for each substitution γ such that $\overline{U}\gamma :_{\Gamma_\gamma, \mathcal{R}} \overline{s}$ and $\overline{V}\gamma \downarrow_{\Gamma_\gamma, \mathcal{R}} \overline{W}\gamma$, then $R\gamma :^*_{\Gamma_\gamma, \mathcal{R}} s$.*

A critical membership $L : s, t$ `if` $\overline{U} : \overline{s} \wedge \overline{V} \downarrow \overline{W}$ is regular *if for each substitution γ such that $L\gamma$ is irreducible, $\overline{U}\gamma :_{\Gamma_\gamma, \mathcal{R}} \overline{s}$ and $\overline{V}\gamma \downarrow_{\Gamma_\gamma, \mathcal{R}} \overline{W}\gamma$, there exists a sort $u \leq_{\mathcal{R}} s, t$ such that $L\gamma :^*_{\Gamma_\gamma, \mathcal{R}} u$.*

A critical membership $L : s, t$ `if` $\overline{U} : \overline{s} \wedge \overline{V} \downarrow \overline{W}$ is strongly regular *if for each substitution γ such that $\overline{U}\gamma :_{\Gamma_\gamma, \mathcal{R}} \overline{s}$ and $\overline{V}\gamma \downarrow_{\Gamma_\gamma, \mathcal{R}} \overline{W}\gamma$, there exists a sort $u \leq_{\mathcal{R}} s, t$ such that $L\gamma :^*_{\Gamma_\gamma, \mathcal{R}} u$.*

Remark how easy it is to express regularity as well as strong regularity. On the other hand, all the above properties of critical pairs and memberships are undecidable, since the universal quantifiers operate on non-recursive sets.

Note also that the sort-decreasingness of a sort membership is a strong requirement: no rewrite step is allowed from $R\gamma$.

6.3 Critical Properties

We start with the critical pair lemma:

Lemma 49 *(Critical Pair Lemma) Assume that S rewrites to P at the root with the rewrite rule $L \to R$ `if` $\overline{U} : \overline{s'} \wedge \overline{V} \downarrow \overline{W}$ applied with substitution θ, and to Q at position q with the rewrite rule $G \to D$ `if` $\overline{U'} : \overline{t'} \wedge \overline{V'} \downarrow \overline{W'}$ and substitution θ such that $\mathcal{V}ar(L) \cap \mathcal{V}ar(G) = \emptyset$, and $q \in \mathcal{FP}os(L)$. Assume further that all critical pairs in $\mathcal{R}$ are confluent. Then $P \downarrow Q$.*

Proof: From the above assumptions we obtain that $S = L\theta$ with condition (i) $\overline{U}\theta : \overline{s'}$ and $\overline{V}\theta \downarrow \overline{W}\theta$, and $S|_q = G\theta$ with condition (ii) $\overline{U'}\theta : \overline{t'}$ and $\overline{V'}\theta \downarrow \overline{W'}\theta$. It follows that $(L\theta)|_q = G\theta$, and since $q \in \mathcal{FP}os(L)$, $L|_q$ and G unify with most general unifier σ. This shows the existence of the critical pair

$$L\sigma[D\sigma]_q = R\sigma \texttt{ if } \overline{U}\sigma : \overline{s'} \wedge \overline{U'}\sigma : \overline{t'} \wedge \overline{V}\sigma \downarrow \overline{W}\sigma \wedge \overline{V'}\sigma \downarrow \overline{W'}\sigma.$$

Now, $\theta = \sigma\tau$ for some substitution τ which satisfies the conditions of the critical pair thanks to (i) and (ii). Hence, $P = R\sigma\tau = (L\sigma[D\sigma]_q)\tau = Q$ is confluent by assumption. □

We are now ready for the Church-Rosser property. Unfortunately, proving local confluence requires sort-decreasingness and vice-versa. We will therefore prove directly the Church-Rosser property.

Theorem 50 *(Church-Rosser) Let $\mathcal{R}$ be a reductive CRMS. Then $\mathcal{R}$ is Church-Rosser iff its critical pairs are confluent and its critical reduced memberships are sort-decreasing.*

Proof: We prove by induction on the proof $S \longleftrightarrow^* T : s$ that (i) $S \downarrow T$ and (ii) $S :^* s$, taking the liberty to systematically omit the subscript $\Gamma, \mathcal{R}$ when possible. For this, we interpret a proof by the multiset of its terms, and compare two multisets in the ordering $(\succ \cup \rhd)_{mul}$.

Ruling out the trivial empty proof, $S \longleftrightarrow^* T$ is written as $S \longleftrightarrow^* T' \longleftrightarrow T$. By induction hypothesis, there exists P' such that $S \longrightarrow^* P' \longleftarrow^* T'$. We are now going to show that there exists some P such that $P' \longrightarrow^* P \longleftarrow^* T$, and $P :^* s$, which yields both (i) and (ii). There are three cases:

1. If T rewrites to T', we can take P' for P. We are left to show that $P' :^* s$.

   Assume that $T :_{\Gamma,\mathcal{R}} s$ by using the membership rule $L : s$ `if` $\overline{U} : \overline{s'} \wedge \overline{V} \downarrow \overline{W}$ with substitution θ, and $T \longrightarrow^p_{\Gamma,\mathcal{R}} T'$ via the rewrite rule $G \to D$ `if` $\overline{U'} : \overline{t'} \wedge \overline{V'} \downarrow \overline{W'}$ and substitution θ, with $\mathcal{V}ar(L) \cap \mathcal{V}ar(G) = \emptyset$. There are two cases:

   (a) $p \notin \mathcal{FP}os(L)$. Then $T = L\theta$, and by the left-linearity assumption, $T' = L\theta'$ with $\theta \longrightarrow \theta'$. By the induction hypothesis (ii) applied to the smaller proof $\theta \longrightarrow \theta'$, θ' is admissible in the environment Γ.

   By the induction hypothesis (ii) again applied to the smaller proof $\overline{U}\theta' \longleftarrow^* \overline{U}\theta :^* \overline{s'}$, we get $\overline{U}\theta' :^* \overline{s'}$. Similarly, by induction hypothesis (i), we get $\overline{V}\theta' \downarrow \overline{W}\theta'$. Therefore, $T' = L\theta' : s$. By induction hypothesis (ii) aplied to the proof $S \longleftrightarrow^* T' : s$, we conclude that $S :^* s$, and we are done with this case.

   (b) $p \in \mathcal{FP}os(L)$. Then, $L|_p$ and G unify with most general unifier σ such that $\theta = \sigma\rho$. Since ρ must satisfy the conditions of the associated critical reduced membership, we have $T' :^* s$. Now, the proof $S \longleftrightarrow^* T' :^* s$ is smaller than the original one, since T has been replaced by straictly smaller terms. We therefore conclude by induction hypothesis (ii) that $S :^* s$.

2. If T' rewrites to T and the proof from T' to P' is empty, that is, $S \longrightarrow^* T'$ we are done with with (i) and (ii).

3. Assume now that $T' \longrightarrow^p P \longrightarrow^* P'$ and $T' \longrightarrow^q T$. We first prove that $P \downarrow T$, by case with respect to the respective positions p and q of the redexes in T'

   (a) *Disjoint redexes* commute as usual.

   (b) *Critical redexes.* The Critical Pair Lemma shows the existence of a confluent proof from $P|_p$ and $T|_p$ assuming p is above q. Hence $P \downarrow T$.

   (c) *Ancestor redexes.* We follow the same line of reasoning as above by showing first the existence of a proof $P|_p \downarrow T|_p$. This is done in the same way as case 1a. We then conclude that $P \downarrow T$.

   We can finally conclude property (i) by a last application of our induction hypothesis (i) to the smaller proof $T \downarrow P \longrightarrow^* P'$ yielding P'' such that $P' \longrightarrow^* P''$ and $T \longrightarrow^* P''$. Since the latter proof is smaller than the starting one, we can apply induction hypothesis (ii) and get $P'' :^* s$. As a result, $S :^* s$ and we are done.

□

We finally address regularity:

Theorem 51 *Let $\mathcal{R}$ be a confluent sort-decreasing CRMS. Then $\mathcal{R}$ is regular (resp. strongly regular) iff its irreducible (resp. all its) critical memberships are regular (resp. strongly regular).*

Proof: We prove the if case. Assume that M is an irreducible term such $M :_{\Gamma,\mathcal{R}} s$ and $M :_{\Gamma,\mathcal{R}} t$. By definition, $M = L\theta$, for some lefthand side L of a membership rule $L : s$ `if` $\ldots$, and $M = G\theta$, for some left hand side G of a membership rule $G : t$ `if` $\ldots$. such that $\mathcal{V}ar(L) \cap \mathcal{V}ar(G) = \emptyset$. Hence, $M = L\theta = G\theta$, and L and M unify with most general unifier σ such that $\theta = \sigma\tau$ and τ satisfies the conditions of the associated critical membership. Since $L\sigma\tau$ is irreducible, so is $L\sigma$. Hence, by assumption, there exists a sort $u \leq_{\mathcal{R}} s, t$ such that $L\sigma\tau : u$. It follows that a membership $P : u$ `if`... applies to $L\sigma$, hence to M by using Lemma 11, Proposition 15, and Proposition 18.

The case of strong regularity is of course very similar, but all critical memberships must be considered. $\square$

Note that there is a real difference between regularity and strong regularity in practice: if a critical membership is reducible, the corresponding pair has not to be considered for strong regularity. In particular, at least for ground terms, regularity of complete specifications reduces to regularity of their constructor part.

6.4 Decidable Criteria

As already pointed out, checking that critical pairs are confluent or that critical reduced memberships are sort-decreasing is undecidable in general in the context of conditionnal rules. We therefore need to elaborate decidable sufficient conditions allowing us to eliminate as many critical pairs/memberships as possible. The problem has been thoroughly investigated by Ganzinger in the usual case of conditional rewrite rules [35], see also [28].

A first possibility is to forget about the conditions of the critical inferences. We may guess that the confluence (resp. sort decreasing) test will then be negative in most cases. An alternative, complementary approach is to solve the conditions, that is, to compute a representation of the set of solutions of the condition of the critical pairs or rule instances. This is of course impossible in general. However, incomplete algorithms can be used for that purpose in order to partially solve these constraints, or, equivalently, have a simpler representation of the condition. Of course, some of the conditions may be really solvable, and some others may not be: solving the solvable ones while keeping the others is a particular incomplete algorithm. The case where the conditions of the rewrite and membership rules do not involve joinability is studied in [22], where confluence is shown to be decidable in this case. The solution, however, involves the use of a decidable fragment of second-order unification. A simpler, less powerful approach is taken in [37], where order-sorted unification is used. In contrast, [60] uses first-order unification only at the price of a complex sort inference system. Another possibility is to assume that the specification is modular, the conditions being evaluated at a lower level of a hierarchy of specifications. This lower level should of course have strong properties, such as a decidable entailment problem. This is not very likely to occur.

Let us explain in more details a solution inspired from [37], which can be easily implemented. The conditions in a rewrite rule (resp. a membership rule) can be split into two parts, the sort memberships for the variables of the pair on the one hand, and the other sort memberships and the joinability conditions on the other hand. Let therefore

$$L \rightarrow R \text{ if } \overline{x} : \overline{s} \wedge \overline{U} : \overline{t} \wedge \overline{V} \downarrow \overline{W} \text{ and } G \rightarrow D \text{ if } \overline{y} : \overline{s'} \wedge \overline{U'} : \overline{t'} \wedge \overline{V'} \downarrow \overline{W'}$$

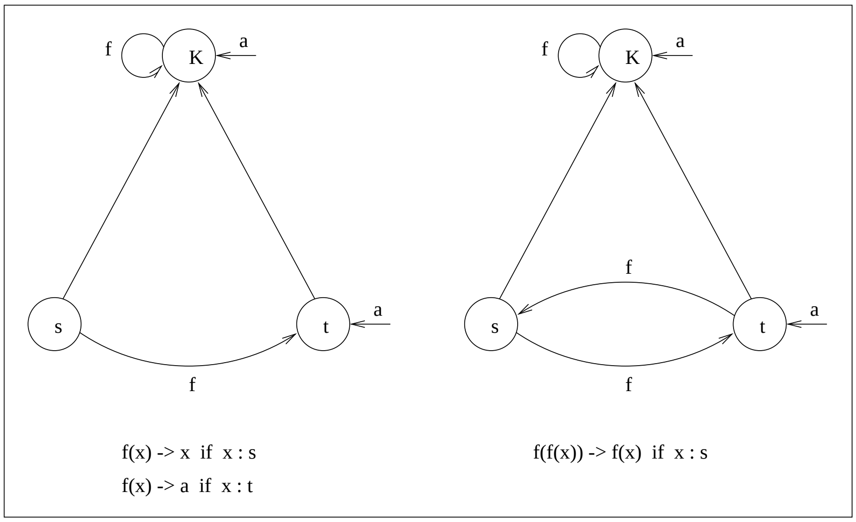


Figure 6.2: Using many-sorted unification versus order-sorted unification

be two reductive rewrite rules such that $\mathcal{V}ar(L) \cap \mathcal{V}ar(G) = \emptyset$, and $L|_p$ unifies G with most general unifier θ, for some $p \in \mathcal{FP}os(L)$. Then, any ground instance of θ which satisfies the conditions in $\overline{x} : \overline{s} \wedge \overline{U} : \overline{t} \wedge \overline{V} \downarrow \overline{W} \wedge \overline{y} : \overline{s'} \wedge \overline{U'} : \overline{t'} \wedge \overline{V'} \downarrow \overline{W'}$ is indeed a ground instance of a most general order-sorted unifier σ, whenever it exists, of the order-sorted terms $(L|_p, \overline{x} : \overline{s})$ and $(G, \overline{y} : \overline{s'})$ which satisfies the conditions in $\overline{U} : \overline{t} \wedge \overline{V} \downarrow \overline{W} \wedge \overline{U'} : \overline{t'} \wedge \overline{V'} \downarrow \overline{W'}$. Of course, there may be no such σ if the two order-sorted terms do not unify. This shows the usefulness of solving, at least partially, the condition in the critical pair, seen as a constraint operating on the variables in the pair. Perhaps it would be possible to integrate other membership conditions in the unification algorithm,but we have not investigated this problem yet.

The soundness of the above argument assumes implicitly that the subsort ordering as well as the sorted operator declarations are known beforehand. If some subsort membership can be deduced from the set of axioms, then, using order-sorted unification may be incomplete in some sense, as shown by the example shown on the left part of figure 6.2. In this case, a critical pair appears once the ordering has been updated as we will explain. A similar situation occurs with sorted operator declarations, as shown by example 6.2. In this case, a critical pair appears after a sorted operator declarations has been generated. A similar problem arises in [47], where order-sorted completion is also used.

Example 52 *For the lefthand side specification, the two rules have no order-sorted overlap, since s and t have no subsort in common. Therefore, the only proof obligation left is the subsort membership $x : t$* `if` *$x : s$ obtained by overlapping the first rule with the signature declaration $f(x) : t$* `if` *$x : s$.*

For the righthand side specification, the single rule has no order-sorted overlap onto itself, since $x : s$ implies $f(x) : t$ and $f(f(x)) : s$. Therefore, the only proof obligation left is the sort declaration $f(x) : s$ `if` *$x : s$ obtained by overlapping the rule with the signature declaration $f(x) : s$* `if` *$x : t$.*

In both cases, there are of course many-kinded unifiers which generate the appropriate critical pairs.

The solution implemented in the Maude Church-Rosser checker for order-sorted specifications [66] uses order-sorted unification as just explained. Signature declarations are kept as meta-level entities as well. The critical memberships are generated via the computation, for each order-sorted rewrite rule, of its *specializations*, obtained by decreasing the sorts of its lefthand side variables, before imposing that, for each specialized rule, the sort of its righthand side is smaller than or equal to the sort of its lefthand side. Then, two different kinds of checks can be performed. A standard check of joinability (resp. membership) for the critical pairs (resp. the critical memberships). This technique may generate redundant critical pairs or memberships which are eliminated in the Maude Church-Rosser checker by a subsumption test. Once these pairs or memberships are generated, their conditions may be removed before the checking, therefore approximating the property to be proved. In case a new subsort membership or a new sort declaration is generated, requiring a change of the signature, then the whole computation must be reinitialized with the new signature. The Maude checker may also generate inductive obligations, in which case only the ground instances of a critical pair (resp. a critical membership) need to be checked. In general, this will require using an inductive prover. The issue of testing for ground confluence and ground sort-decreasingness should merit more attention. Not much has yet been done on this question, despite its importance.

In conclusion, checking conditional critical pairs for joinability is a quite difficult problem, for which many techniques have been described in the literature which allow to solve the problem in various practical cases. But no really satisfactory automatic solution is available. In practice, the use of an inductive prover may be found useful.

On the other hand, regularity seems much easier to check, since the existence of a minimum for a given set of sorts does not depend on any computation. This allows inferring non-regularity in many cases. To infer regularity is as difficult as to infer confluence and sort-decreasingness, since the substitutions satisfying a given condition must be considered.

To conclude this section, let us analyze why critical pairs need superpositions at subterms, while critical inferences involving a membership rule do not. This is directly related to the deduction rules. The rules for rewriting (or equality) involve the congruence rule, while the rules for membership do not. Hence, deductions in normal form do not have the same shape. In the rewriting case, a normal form deduction may end up using the congruence rule, while it is of course not possible with the deductions for memberships.

Chapter 7

Completion

Due to the Horn Clause format, general techniques apply. However, we are not only interested in confluence, but also in reductivity, type decreasingness and, to a lesser extent, regularity. Our goal is to design a completion procedure that achieves (some of) these different goals simultaneously.

7.1 Confluence and Sort-Decreasingness

Since sort decreasingness and confluence can be obtained as usual by adding valid consequences, we start with these two properties alone. The completion procedure displayed in Figure 7.1 deals with four categories of objects: a set $\mathcal{R}$ of reductive conditional rules; a set $\mathcal{E}$ of conditional equations or conditional memberships; a set $\mathcal{M}$ of reductive conditional memberships; and the subsort ordering $\leq_{\mathcal{R}}$. We could have added the subsort membership axioms to $\mathcal{M}$, but incorporating all such axioms in the metalevel entity $\leq_{\mathcal{R}}$ allows for a better clarity of the exposition. The ordering $\succ$ on the term structure used to check reductivity of the conditional rules and memberships must be provided by the user. Given the ordering $\succ$, an ordering $>$ on rules is derived, which is used to prevent simplification of a lefthand side of rule by a bigger rule, see [27]. We have packed all simplification rules in one for matter of space saving. **Simplify E** (resp. **Simplify M**; **Simplify R**) is obtained by taking $C = C'$, $R = R'$ and $D = D'$ (resp. $e = e', R = R'$ and $D = D'$; $e = e', C = C'$ and ($R = R'$ or $D = D'$)).

Critical Pair adds the critical pairs to the set $\mathcal{E}$ of equational consequences, while **Critical Reduced Membership** adds membership equations obtained from the critical reduced memberships. As usual, the role of **Orient** is to transform equations of $\mathcal{E}$ into reductive rules, while **Check Memberships** does the same verification for memberships in $\mathcal{E}$. The **Delete** rule is as usual. **Subsort** moves a new subsort membership to the subsort ordering, while **Subsort Simplification** discards it if it is already in the transitive closure of the subsort ordering declarations. We will come back later on the key role played by these rules. We packed all simplification rules into one, by adding the extra assumption to make sure that some simplification happened. Note that simplifying a righthand side can only decrease its sort, allowing us to keep the reduced formula as a rule, exactly as in the standard completion procedure.

This version of completion is very close to the standard completion procedure, except that memberships need to be taken care of.

Theorem 53 *Given a specification $\mathcal{E}$ in membership equational logic, the completion rules displayed at Figure 7.1 return, assuming a successful fair execution, a (possibly infinite)*

$$\textbf{Critical Pair: } \frac{\mathcal{E}, \mathcal{M}, \mathcal{R}, \leq_{\mathcal{R}}}{\mathcal{E} \cup \{L = R \texttt{ if } C\}, \mathcal{M}, \mathcal{R}, \leq_{\mathcal{R}}}$$
if $L = R \texttt{ if } C$ is a critical pair of $\mathcal{R}$

$$\textbf{Reduced Membership: } \frac{\mathcal{E}, \mathcal{M}, \mathcal{R}, \leq_{\mathcal{R}}}{\mathcal{E}, \{R : t \texttt{ if } C\} \cup \mathcal{M}, \mathcal{R}, \leq_{\mathcal{R}}}$$
if $L : t \to R \texttt{ if } C$ is a critical reduced membership of $\mathcal{R} \cup \mathcal{M}$

$$\textbf{Orient Equations: } \frac{\mathcal{E} \cup \{P = Q \texttt{ if } \overline{U} : \overline{u} \wedge \overline{V} = \overline{W}\}, \mathcal{M}, \mathcal{R}, \leq_{\mathcal{R}}}{\mathcal{E}, \mathcal{M}, \mathcal{R} \cup \{P \to Q \texttt{ if } \overline{U} : \overline{u} \wedge \overline{V} \downarrow \overline{W}\}, \leq_{\mathcal{R}}}$$
if $P \succ Q$ and $\forall R \in \overline{U} \cup \overline{V} \cup \overline{W} \quad P(\succ \cup \rhd) R$

$$\textbf{Check Memberships: } \frac{\mathcal{E} \cup \{P : s \texttt{ if } \overline{U} : \overline{u} \wedge \overline{V} = \overline{W}\}, \mathcal{M}, \mathcal{R}, \leq_{\mathcal{R}}}{\mathcal{E}, \mathcal{M} \cup \{P : s \texttt{ if } \overline{U} : \overline{u} \wedge \overline{V} \downarrow \overline{W}\}, \mathcal{R}, \leq_{\mathcal{R}}}$$
if $P(\succ \cup \rhd) R \quad \forall R \in \overline{U} \cup \overline{V} \cup \overline{W}$

$$\textbf{Delete: } \frac{\mathcal{E} \cup \{S = T \texttt{ if } C\}, \mathcal{M}, \mathcal{R}, \leq_{\mathcal{R}}}{\mathcal{E}, \mathcal{M}, \mathcal{R}, \leq_{\mathcal{R}}}$$
if S and T are identical or C is unsatisfiable

$$\textbf{Subsort: } \frac{\mathcal{E}, \mathcal{M} \cup \{x : s \texttt{ if } x : t\}, \mathcal{R}, \leq_{\mathcal{R}}}{\mathcal{E}, \mathcal{M}, \mathcal{R}, \leq_{\mathcal{R}} \cup \{t \leq_{\mathcal{R}} s\}}$$
if x is a variable and $t \not\leq s$

$$\textbf{Subsort Simplification: } \frac{\mathcal{E}, \mathcal{M} \cup \{x : s \texttt{ if } x : t\}, \mathcal{R}, \leq_{\mathcal{R}}}{\mathcal{E}, \mathcal{M}, \mathcal{R}, \leq_{\mathcal{R}}}$$
if x is a variable and $t \leq_{\mathcal{R}} s$

$$\textbf{Simplify E/M/R: } \frac{\mathcal{E} \cup \{e\}, \mathcal{M} \cup \{M : s \texttt{ if } C\}, \mathcal{R} \cup \{L \to R \texttt{ if } D\}, \leq_{\mathcal{R}}}{\mathcal{E} \cup \{e'\}, \mathcal{M} \cup \{M : s \texttt{ if } C'\}, \mathcal{R} \cup \{L \to R' \texttt{ if } D'\}, \leq_{\mathcal{R}}}$$
$$\textbf{if } \begin{cases} e \longrightarrow^* e', C \longrightarrow^* C', \ R \longrightarrow^*_{\mathcal{R}} R', \ D \longrightarrow^*_{\mathcal{R}} D' \\ (e, C, R, D) \neq (e', C', R', D') \end{cases}$$

$$\textbf{Simplify Head M: } \frac{\mathcal{E}, \mathcal{M} \cup \{M : s \texttt{ if } C\}, \mathcal{R}, \leq_{\mathcal{R}}}{\mathcal{E} \cup \{M' : s \texttt{ if } C\}, \mathcal{M}, \mathcal{R}, \leq_{\mathcal{R}}}$$
if $M \longrightarrow_{\mathcal{R}} M'$

$$\textbf{Simplify Head R: } \frac{\mathcal{E}, \mathcal{M}, \mathcal{R} \cup \{L \to R \texttt{ if } C\}, \leq_{\mathcal{R}}}{\mathcal{E} \cup \{L' = R \texttt{ if } C\}, \mathcal{M}, \mathcal{R}, \leq_{\mathcal{R}}}$$
$$\textbf{if } \begin{cases} L \longrightarrow_{P \to Q \texttt{ if } D \,\in\, \mathcal{R}} L' \\ (L \to R \texttt{ if } C) > (P \to Q \texttt{ if } D) \end{cases}$$

Figure 7.1: Completion Rules

reductive, sort-decreasing, confluent CRMS $\mathcal{R} \cup \mathcal{M}$ equivalent to $\mathcal{E}$.

Proof: Since this kind of proof has by now become routine, we only sketch it but give in detail the needed ingredients. The idea is to view the inference rules as rewriting proofs until a normal form proof is obtained. Our goals will be fulfilled if any proof is transformed in finite time into a rewrite proof. This implies proving that: (a) the rewrite system on proofs terminates; and (b) the normal form proofs are indeed rewrite proofs, that is, of the form $S \longrightarrow^*_{\mathcal{R}} \longleftarrow^*_{\mathcal{R}} T$ and $T \longrightarrow^*_{\mathcal{R}} :_{\mathcal{R}} s$.

Our elementary proofs here are of the form $S \longleftrightarrow_{\mathcal{E}} T$, $S :_{\mathcal{E}} s$, $S \longrightarrow_{\mathcal{R}} T$ and $S :_{\mathcal{M}} s$.

To show (a), we exhibit a well-founded ordering on proofs, considered as multisets of elementary proofs. For this, we interpret elementary proof steps as triples as follows (replacing unnecessary information by $\perp$):

$S \longleftrightarrow_{\mathcal{E}} T$ with the equation $L = R \texttt{ if } P$ *by* $< \{S, T\}, L = R \texttt{ if } P, \perp >$
$S :_{\mathcal{E}} s$ with the membership $L : s \texttt{ if } P$ *by* $< \{S\}, L : s \texttt{ if } P, 0 >$
$S \longrightarrow_{\mathcal{R}} T$ with the rule $L \to R \texttt{ if } P$ *by* $< \{S\}, L = R \texttt{ if } P, \perp >$
$S :_{\mathcal{M}} s$ with the membership $L : s \texttt{ if } P$ *by* $< \{S\}, L : s \texttt{ if } P, 1 >$

In this analysis, we consider the subsort membership axioms as if they were in $\mathcal{M}$. As a consequence, our proof objects do not change when **Subsort** is applied. We can allow for this, since **Subsort** can be applied finitely times only.

Triples are compared in the ordering $(\succ_{mul}, >, >_{\mathbb{N}})_{lex}$, where $>$ is the ordering on rules (or memberships) defined as follows: $L \to R \texttt{ if } P > L' \to R' \texttt{ if } P'$ if and only if

(i) $L|_p = L'\sigma$ for some position p and substitution σ such that $p \neq \Lambda$ or σ is not a variable renaming, or else

(ii) $(L, R, P)(\longrightarrow_{\mathcal{R}})_{mul}(L', R', P')$

In the same way, $L : s \texttt{ if } P > L' : s \texttt{ if } P'$ if and only if (ii) is satisfied.

Verifying that the proof rewrite rules decrease in the ordering is now a routine check. For an example, **Reduced Membership** transforms a proof of the form $L\gamma[D\gamma]_p \underset{G \to D \texttt{ if } Q}{\overset{p}{\longleftarrow}} L\gamma :_{L:s \texttt{ if } P} s$ into the smaller proof $L\gamma[D\gamma]_p :_{L\sigma[D\sigma]_p:s \texttt{ if } P \wedge Q} s$.

To show (b) amounts to checking that all undesirable subproofs are eliminated by the rewrite rules on proofs. The case of equational proofs is as usual. For membership proofs, we need to show that subproofs of the form $S \longleftarrow_{\mathcal{R}} T :_{L:s \texttt{ if } P \in \mathcal{M}} s$ are reducible. We do it by induction on T using the ordering $\succ \cup \rhd$, the base case being void. Let therefore $T = L\gamma$ and $\vdash_{\mathcal{R} \cup \mathcal{M}} P\gamma$. There are two cases:

- If the rewrite from S to T takes place in the substitution part γ of the membership rule, that is $\gamma \longrightarrow_{\mathcal{R}} \gamma'$, it is then enough to show that $T :_{L:s \texttt{ if } P \in \mathcal{M}} s$, which reduces to $\vdash_{\mathcal{R} \cup \mathcal{M}} P\gamma'$. Now, for each $V \downarrow W \in P$, $V\gamma \downarrow W\gamma \in P\gamma$, therefore $V\gamma \downarrow_{\mathcal{R} \cup \mathcal{M}} W\gamma$, and hence $V\gamma' \downarrow_{\mathcal{R} \cup \mathcal{M}} W\gamma'$. On the other hand, for each $U : s \in P$, we have $U\gamma :^*_{\mathcal{R} \cup \mathcal{M}} s$ and $L\gamma \succ \cup \rhd U\gamma$, hence we conclude by induction that $U\gamma' :^*_{\mathcal{R} \cup \mathcal{M}} s$.

- If the rewrite from S to T takes place at a non-variable position of L, then the proof is reduced according to the rewrite rule on proofs associated with **Reduced Memberships**.

The rest of the argument uses fairness as usual to show that the limit system satisfies (a) and (b) even in case the computation diverges. $\square$

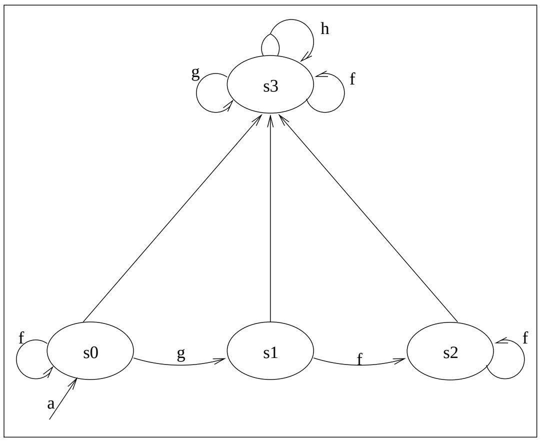
A specification resulting from completion will not be regular in general, even when the starting specification is regular, as witnessed by the simple example in which s and t are of the same kind, a has sort s and b sort t, and $a \to b$ is the only rule. This specification is of course regular (and strongly regular as well), but completion will add the membership $b : s$ in order to ensure sort-decreasingness, and this will destroy regularity.

In practice, we can expect completion to diverge often, as does conditional completion. Besides, our completion procedure is rather simple-minded, since it does not implement any sophisticated method to get rid of useless equations or memberships. Such methods are described at length in [35]. They can of course be safely added, with the necessary adaptations for memberships. One problem, in particular, is to get rid of an equation or a membership whose condition is unsatisfiable.

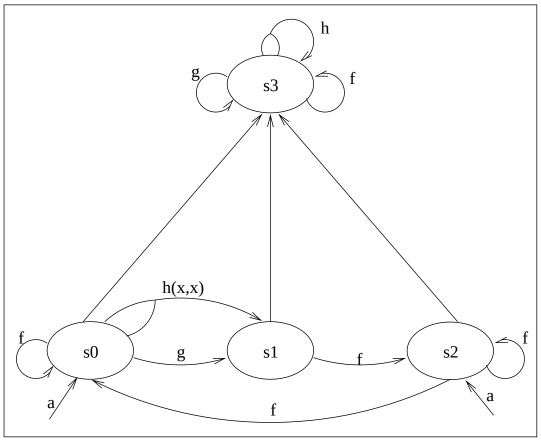
Compared to [47], our method is based on the same idea that the proof of a membership statement needs in general equality subproofs, as expressed by the subject reduction rule. The difference is that our approach is much simpler as well as more general, and this is due to the use of a more powerful logic. For example, the possibility of adding new membership axioms in the completion process solves the sort-decreasingness problem in a very general and satisfactory way, analogous to what is done for ensuring confluence. This solution cannot be used with order-sorted logic, and alternatives had to be chosen, such as the computation of *specializations* of rules as done in [41, 61].

In [47], there is no need of sort-decreasingness, which is not even used for speeding up computations. As a consequence, more deductions are needed for computing, while (part of) these deductions are in some sense factored-out in our approach and are enforced by the completion process.

Comon uses another logic extending order-sorted logic with second-order context variables. Again, our method is much simpler both technically and conceptually, as shown by an example taken from [22] and shown in Figure 7.2, in which the inferred memberships are added to the automaton. In this example, only memberships are added, the input and output set of rules being the same. This will actually be the case each time the starting example has no critical pairs, hence we are guaranteed to terminate when this is the case, which is not true of other methods.



$f(x) \to a$ `if` $x : s2$



$g(x) \to h(x, x)$ `if` $x : s3$

Figure 7.2: Completion of Comon's Specification

Property 54 *Completion terminates for any specification in membership equational logic which does not have critical pairs.*

$$\textbf{Critical Membership: } \frac{\mathcal{E}, \mathcal{M}, \mathcal{R}}{\mathcal{E}, \mathcal{M} \cup \{L : inf_{\leq_{\mathcal{R}}}(s,t) \texttt{ if } C\}, \mathcal{R}}$$
$$\textbf{if } L : s,t \texttt{ if } C \text{ is a critical membership of } \mathcal{M}$$

Figure 7.3: Additional Completion Rule for Regularity

7.2 Completion with Regularity

Achieving regularity will require adding new sorts, but this can be done by need: each time a term may inhabit different sorts in $\overline{s}$, we will introduce a new sort $\bigcap_{s\in\overline{s}} s$, which will be a subsort of each sort in $\overline{s}$ and a supersort of each of their common subsorts. Of course, we will have the properties $\bigcap_{s\in\{s\}} = s$, and $\bigcap_{s\in\overline{s}} \cap \bigcap_{s\in\overline{t}} = \bigcap_{s\in\overline{s}\cup\overline{t}}$, implying in particular that $s \cap s = s$, and making the set of intersection sorts ever needed finite. Adding these new sorts and subsort relationships (at once or by need) results in a conservative extension of the starting specification.

This first step does not imply regularity alone, new memberships at these new sorts have to be added in order to achieve this property, which is done by the rule given in Figure 7.3

The above rule do not preserve the theory, since new sorts are added. But it indeed constructs a conservative extension:

Theorem 55 *Given a specification $\mathcal{E}$ in membership equational logic, the completion rules displayed in Figures 7.1 and 7.3 return, assuming a successful fair execution, a (possibly infinite) reductive, sort-decreasing, confluent CRMS $\mathcal{R} \cup \mathcal{M}$ which is a conservative extension of $\mathcal{E}$.*

Proof: Since new sorts and new memberships at these sorts are added, we need first to prove that we cannot prove more memberships in the original signature after adding the new sorts and subsort relationships. For this, it is enough to reason semantically, and it is then clear, that the added sorts and subsorts relationship respect the existing inclusions between the sets interpreting the original sorts in the models.

The rest of the proof is similar to the previous one. □

Notice finally that building some axioms, e.g., subsort memberships and sort declarations, in the unification algorithm raises the same kind of problems as those pointed out in the previous chapter : generating new subsort memberships, or new sort declarations (in the order-sorted sense) results in changing the signature, hence the unification algorithm, and therefore to reinitialize the whole computation. It turns out that the same phenomenon surfaces in [47], where completion needs sort-inheritance for the order-sorted unification algorithm to behave correctly, and therefore it has to be reinitialized each time a subsort-membership is generated. In our case, this reinitialization is only needed when order-sorted unification is used instead of many-sorted unification.

Chapter 8

Protecting Extensions

Among the several constructions that allow building larger specifications from smaller components, *protecting extensions* play a central role. Adding new kinds, sorts, or rules to a specification should be modular, that is, it should not mess up the original specification, explaining the name of *protecting.* A frequent, particular case of a protecting extension is obtained when no new kinds or sorts are added, and is termed a *constructor specification*, the constructors being the function symbols in the protected subspecification. The requirement for constructor specifications is that any sortable term, when evaluated, should return a result expressed by means of constructors together with its sort. Terms whose result is not expressible by means of constructors will be considered to be *error terms.* In our framework, such error terms will inhabit a kind, but not a sort. In our specification of numbers given at figure 3.1, the substraction operation is not defined over complex numbers, hence the term `<S(0),0> - <0,0>` does not inhabit a sort, but the kind `Number`. This term evaluates to itself, and is considered an error term. Were integer division $\_/\_$ be defined in that specification, $0/0$ would be another error term. Error terms are very useful, since they allow for partial functions. On the other hand, our framework is in the spirit of total functions. This means that the user has the possibility to fully specify the appropriate input sorts for which a function is completely defined, and therefore always evaluates to a constructor term of the appropriate sort. This is usually done by giving membership rules for the defined symbols. If this is not the case, our upcoming procedure for testing completeness of a function f will exhibit the sorts on which f is only partially defined. More precisely, our procedure will output a set of patterns on whose ground instances the function f does not evaluate to a constructor term. For our example of numbers, a membership rule states that substraction is defined over integers always resulting in an integer, while another membership rule states under which condition it may result in a natural. Substraction is not defined when at least one of its arguments is not an integer. This can be indeed inferred by our proof procedure for testing completeness, regardless of the membership declarations, since they are not useful for rewriting a ground term to its normal form.

Since a constructor specification is a particular case of a protecting extension, we will mainly consider the latter case, and spell out when necessary how results specialize to the particular case. Of course, this assumes that the set of constructors of a specification is known. In Maude, the user will have to specify them by using different keywords for the constructors and the defined symbols. This will be spelled out when giving an example.

Before starting with the formal definitions, we introduce an important notation. Let $\mathcal{R} \subseteq \mathcal{R}'$ be two specifications, and let $\mathcal{T}_{\mathcal{R}'}|_{\mathcal{R}}$ be the restriction to the signature of the specification $\mathcal{R}$ of the initial algebra $\mathcal{T}_{\mathcal{R}'}$ associated with the specification $\mathcal{R}'$. Since $\mathcal{R} \subseteq$

$\mathcal{R}'$, $\mathcal{T}_{\mathcal{R}'}|_{\mathcal{R}}$ is an $\mathcal{R}$-algebra, and therefore there is a unique $\mathcal{R}$-homomorphism $h : \mathcal{T}_{\mathcal{R}} \to \mathcal{T}_{\mathcal{R}'}|_{\mathcal{R}}$.

Definition 56 *Let $\mathcal{R} = (\mathcal{K}, \Sigma, \mathcal{S}, \mathcal{R})$ and $\mathcal{R}' = (\mathcal{K}', \Sigma', \mathcal{S}', \mathcal{R}')$ be specifications such that $\mathcal{R} \subseteq \mathcal{R}'$ componentwise. We say that $\mathcal{R} \subseteq \mathcal{R}'$ is a* protecting extension *when $\forall s \in \mathcal{S}$, the function $h_s : (\mathcal{T}_{\mathcal{R}})_s \to (\mathcal{T}_{\mathcal{R}'})_s$ induced by the unique homomorphism $h : \mathcal{T}_{\mathcal{R}} \to \mathcal{T}_{\mathcal{R}'}|_{\mathcal{R}}$ is bijective.*

Using Goguen's vocabulary, injectivity and surjectivity correspond respectively to the so-called 'no junk' and 'no confusion' properties, which are here relativized to the sorts in the protected subspecification. This relativization hides a subtle problem in the case where the specification R contains two copies of a given sort, say, two copies `Nat` and `Nat'` of the natural numbers. Then, these two sorts can be identified in the initial model by adding a supersort of both in $\mathcal{S}'$, say `SuperNat` and by giving the appropriate equations to identify the corresponding elements, say `0=0'` and `s(x)=s'(x) if x:SuperNat`. With our definition, such an extension is protecting, because there will be no way to observe that the sort structure has indeed changed by identifying `Nat` and `Nat'` by checking properties of the functions h_s. This pathological case may be ruled out by simply forbidding adding a new sort above two sorts of $\mathcal{S}$ that did not have a common supersort. This extra hypothesis could be carried along this chapter without changing the existing results. We decided to warn the reader instead, and allow this extra generality. In the next chapter, we will take the opposite decision, because we lack the adequate understanding of the influence of this problem upon arbitrary models.

The idea captured by the definition is that each ground term in the whole specification $\mathcal{R}'$ having a sort s belonging to the subspecification $\mathcal{R}$ can be proved equal to a ground term of the subspecification $\mathcal{R}$ having the sort s, and also that $\mathcal{R}'$ does not impose new equalities on ground terms of the subspecification $\mathcal{R}$. However, ground terms of the specification $\mathcal{R}'$ not having a sort in the subspecification $\mathcal{R}$ need not have an equivalent term in $\mathcal{R}$. They may have a new sort, or simply be error terms inhabiting a kind but no sort.

The literature has stressed the particular case where $\mathcal{K}' = \mathcal{K}$, $\mathcal{S}' = \mathcal{S}$ and the symbols in Σ are free, in which case they are called constructors. We will adopt a more liberal definition of constructors, by considering that the symbols in the subspecification Σ are constructors, and the symbols in $\Sigma' \setminus \Sigma$ are defined.

The notion of protecting extension is very useful to express properties of *structured specifications*. Besides stating a crucial preservation property about the initial model of a subspecification, a protecting declaration allows expressing a variety of useful properties such as constructor subspecification, or getting the implicit equivalent of second-order axiomatizations in specifications with a "loose" (as opposed to initial) semantics, in which subspecifications with an initial semantics are imported. For example, a loose specification of partially ordered sets can import BOOL in a protecting mode, to define the equality predicate as a Boolean-valued function, or another theory can import NAT in a protecting mode to fix the interpretation of the NAT submodule to be the standard natural numbers, while leaving other parts vary in a loose way. Such protecting assertions can then be exploited when proving properties of structured specifications. In order to show the additional expressive power of protecting extensions obtained by the distinction between kinds and sorts, we give in Figure 8.1 yet another specification of numbers, by constructing first negative and positive numbers, before building the integers as a protecting extension of the first specification. In this simple-minded specification of the module INT, we give two equations, between constructor terms of the specification NAT-NEG. Protection is however ensured, since the lefthand side constructor term of each rule does not have a sortable ground instance in the subspecification NAT-NEG. Note that the specification of INT is not a constructor specification, since we have added a new sort. Of course, the difference is only in the vocabulary, we could adopt the same name for both cases, and, of course, the function symbols S and P of the module NAT-NEG play the role of constructors. According to the definition of a reductive system, the subsort inclusion memberships are given here via the Maude keyword `subsort`. We stick even closer to the Maude syntax, by using operator declarations via the keywords `cop` for constructors and `op` for non-constructors, both being syntactic sugar for particular membership rules.

```
fmod NAT-NEG
kind Number[Zero < Neg, Zero < Nat]
cop 0 : Zero
cop S_ : Nat → Nat
cop P_ : Neg → Neg
end fmod

fmod INT is protecting NAT-NEG
sorts Int
subsorts Neg < Int, Nat < Int
op S_ : Int → Int
op P_ : Int → Int
eq P(S(x)) = x if x : Int
eq S(P(x)) = x if x : Int
end fmod
```

Figure 8.1: A specification of integers

8.1 A Characterization of Protecting Extensions

Testing for protecting extensions is the motivation for the following fundamental theorem:

Theorem 57 *Let $(\mathcal{K},\Sigma,\mathcal{S},\mathcal{R}) \subseteq (\mathcal{K}',\Sigma',\mathcal{S}',\mathcal{R}')$ be both ground confluent, sort decreasing and reductive, and let $\mathcal{R}' = \mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}} \cup \mathcal{M}'_{\mathcal{S}'\setminus\mathcal{S}}$, with $\mathcal{E}'$ the rewrite rules, $\mathcal{M}'_{\mathcal{S}}$ the membership rules asserting sorts in $\mathcal{S}$, and $\mathcal{M}'_{\mathcal{S}'\setminus\mathcal{S}}$ the remaining memberships. Assume further that:*

(i) each membership rule in $\mathcal{R}$ is of the form $f(\overline{L}) : s \;\mathtt{if}\; \mathcal{V}ar(\overline{L}) : \overline{s} \wedge \overline{U} : \overline{t} \wedge \overline{V} \downarrow \overline{W}$ where $\mathcal{V}ar(\overline{U},\overline{V},\overline{W}) \subseteq \mathcal{V}ar(\overline{L})$ and for each $M \downarrow N \in \overline{V} \downarrow \overline{W}$, there exists some sort $t \in \mathcal{S}$ such that $M :^*_{\mathcal{V}ar(\overline{L}):\overline{s},\mathcal{R}} t$ and $N :^*_{\mathcal{V}ar(\overline{L}):\overline{s},\mathcal{R}} t$.

(ii) $\forall s' \in \mathcal{S}', s \in \mathcal{S}$ such that $s' \leq_{\mathcal{R}'} s$, then $s' \in \mathcal{S}$ and $s' \leq_{\mathcal{R}} s$,

(iii) $\forall S \in \mathcal{T}_\Sigma$ such that $S :^*_{\mathcal{R}} s$ with $s \in \mathcal{S}$ and $S = S\downarrow_{\mathcal{R}}$, then $S\downarrow_{\mathcal{R}'} = S$.

Then, $\mathcal{R} \subseteq \mathcal{R}'$ is a protecting extension iff

$$\forall \varphi \in \mathcal{M}'_{\mathcal{S}} \quad \mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}'\setminus\mathcal{S}} \models_{Ind} \varphi \quad (*)$$

in which case $\mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}'\setminus\mathcal{S}}$ is ground confluent, ground sort-decreasing and reductive.

Note that property (*) corresponds to surjectivity, while injectivity is hidden in assumption (iii), a property called *preservation of sortable ground normal forms.*

This result relates the property of being a protecting extension with induction, and provides a way to check it by using an inductive prover such as the one described in Section 9. Besides, it shows the interesting phenomenon that new memberships at the old sorts are not really useful for computing with ground terms. Indeed, it follows from the theorem that $\mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}' \setminus \mathcal{S}}$ is itself a ground-confluent, ground sort-decreasing CRMS with which we can do any computation we can do with R'. We will make an essential use of this property.

Proof: We prove in turn the if and only if directions of the statement.

1. Assuming (*), we need to prove that for each sort $s \in \mathcal{S}$, the function $h_s : (\mathcal{T}_{\mathcal{R}})_s \rightarrow (\mathcal{T}_{\mathcal{R}'})_s$ induced by the unique homomorphism $h : \mathcal{T}_{\mathcal{R}} \rightarrow \mathcal{T}_{\mathcal{R}'}|_{\mathcal{R}}$ is bijective. The proof splits again into injectivity followed by surjectivity.

   (a) Let U and V be ground terms in $\mathcal{T}_{\Sigma}$ such that $U :^*_{\mathcal{R}} s$ and $V :^*_{\mathcal{R}} s$ for some sort $s \in \mathcal{S}$, and $U \longleftrightarrow^*_{\mathcal{R}'} V$. Let now U' and V' be the $\mathcal{R}$-normal-forms of, respectively, U and V. By ground confluence of $\mathcal{R}'$, we have $U' \downarrow_{\mathcal{R}'} V'$, hence $U' = V'$ by assumption (iii). It follows that $U \longleftrightarrow^*_{\mathcal{R}} V$, showing injectivity of h_s for $s \in \mathcal{S}$.

   (b) Surjectivity is proved in two steps.

      i. Assume first that $\mathcal{M}'_{\mathcal{S}} = \emptyset$. In this case, note that (*) is trivially satisfied, hence we need to show surjectivity, that is, $\forall T \in \mathcal{T}_{\Sigma'}$ such that $\exists s \in \mathcal{S}$ with $T :^*_{\mathcal{R}'} s$, then $\exists T' \in \mathcal{T}_{\Sigma}$ with $T' :^*_{\mathcal{R}} s$ and $T \longleftrightarrow^*_{\mathcal{R}'} T'$, under our assumptions (i), (ii) and (iii). This follows directly from Lemma 58 below.

      ii. Assume now that $\mathcal{M}'_{\mathcal{S}} \neq \emptyset$. Then (*) implies that $\mathcal{R} \subseteq \mathcal{R}'$ is protecting because, considering $\mathcal{R} \subseteq \mathcal{R} \cup \mathcal{E} \cup \mathcal{M}_{\mathcal{S}' \setminus \mathcal{S}} \subseteq \mathcal{R}'$, the first inclusion is protecting by the previous proof, and the second inclusion is protecting by the fact that $\mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}' \setminus \mathcal{S}} \models_{Ind} \mathcal{M}'_{\mathcal{S}}$, which means that the two algebras $\mathcal{T}_{\mathcal{R}'}$ and $\mathcal{T}_{\mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}_{\mathcal{S}' \setminus \mathcal{S}}}$ are isomorphic. Hence, $\mathcal{R} \subseteq \mathcal{R}'$ is the composition of two protecting inclusions and therefore it can be easily seen that it is also protecting.

2. Assuming that $\mathcal{R} \subseteq \mathcal{R}'$ is a protecting extension, (*) follows from Lemma 59. □

We now state and prove the technical lemmas used in the proof of the theorem :

Lemma 58 *Let $\mathcal{R} \subseteq \mathcal{R}'$ be both ground confluent, ground sort-decreasing and reductive, and let, with the same notations as before, $\mathcal{R}' = \mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}' \setminus \mathcal{S}}$, assuming $\mathcal{M}'_{\mathcal{S}}$ is empty. Assume further that:*

(i) each membership in $\mathcal{R}$ is of the form $f(\overline{L}) : s$ $\mathtt{if}$ $\mathcal{V}ar(\overline{L}) : \overline{s} \wedge \overline{U} : \overline{t} \wedge \overline{V} \downarrow \overline{W}$ where $\mathcal{V}ar(\overline{U}, \overline{V}, \overline{W}) \subseteq \mathcal{V}ar(\overline{L})$ and for each $M \downarrow N \in \overline{V} \downarrow \overline{W}$, there exists some sort $t \in \mathcal{S}$ such that $M :^*_{\mathcal{V}ar(\overline{L}):\overline{s},\mathcal{R}} t$ and $N :^*_{\mathcal{V}ar(\overline{L}):\overline{s},\mathcal{R}} t$.

(ii) $\forall s' \in \mathcal{S}', s \in \mathcal{S}$ such that $s' \leq_{\mathcal{R}'} s$, then $s' \in \mathcal{S}$ and $s' \leq_{\mathcal{R}} s$,

(iii) $\forall S \in \mathcal{T}_{\Sigma}$ such that $S :^*_{\mathcal{R}} s$ with $s \in \mathcal{S}$ and $S = S \downarrow_{\mathcal{R}}$, then $S \downarrow_{\mathcal{R}'} = S$.

Then, $\forall T' \in \mathcal{T}_{\Sigma'}$ such that $T' :^*_{\mathcal{R}'} s \in \mathcal{S}$, there exist $T = T' \downarrow_{\mathcal{R}'}$ such that $T \in \mathcal{T}_{\Sigma}$ and $\vdash_{\mathcal{R}} T : s$, that is, $T :_{\mathcal{R}} t$ for some $t \leq_{\mathcal{R}} s$.

Proof: By induction on the well-founded ordering $\succ \cup \rhd$, where $\succ$ is the rewrite ordering used to prove reductivity of $\mathcal{R}'$.

Let $T' \in \mathcal{T}_{\Sigma'}$ such that $T' :^*_{\mathcal{R}'} s \in \mathcal{S}$, and $T = T' \downarrow_{\mathcal{R}'}$. By definition of $T' :^*_{\mathcal{R}'} s$, there exists some term $T'' \in \mathcal{T}_{\Sigma'}$ and sort $t' \leq_{\mathcal{R}'} s$ such that $T' \longrightarrow^*_{\mathcal{R}'} T''$ and $T'' :_{\mathcal{R}'} t'$. Since $\mathcal{R}'$ is ground-confluent, $T' \downarrow_{\mathcal{R}'} = T'' \downarrow_{\mathcal{R}'}$, and therefore $T'' \longrightarrow^*_{\mathcal{R}'} T$. Since $\mathcal{R}'$ is ground sort-decreasing, it follows that $T :^*_{\mathcal{R}'} t'$, and because T is in $\mathcal{R}'$-normal form by definition, there exists some sort $t \leq_{\mathcal{R}'} t'$ such that $T :_{\mathcal{R}'} t$. Hence, there exists a membership rule in $\mathcal{R}'$ of the form $f(\overline{L}) : t$ **if** P, such that $T = f(\overline{L})\gamma$, γ is a ground $\mathcal{R}'$-normalized substitution, and $\vdash_{\mathcal{R}'} P\gamma$. By transitivity, $t \leq_{\mathcal{R}'} s$, and by assumption (ii), $t \in \mathcal{S}$, and hence $t \leq_{\mathcal{R}} s$ by assumption (ii) again. Note now that this membership rule belongs indeed to $\mathcal{R}$ by our assumption that $\mathcal{M}'_{\mathcal{S}}$ is empty. Therefore, $f(\overline{L}) \in \mathcal{T}_\Sigma(\mathcal{X})$ and terms in P belong also to $\mathcal{T}_\Sigma(\mathcal{X})$.

By assumption (i), $\forall x \in \mathcal{V}ar(\overline{L})$, there exist a membership atom of the form $x : u$ in P with $u \in \mathcal{S}$, hence $\vdash_{\mathcal{R}'} x\gamma : u$. By the induction hypothesis, $\vdash_{\mathcal{R}} (x\gamma) \downarrow_{\mathcal{R}'} : u$ and $(x\gamma) \downarrow_{\mathcal{R}'} = x\gamma \in \mathcal{T}_\Sigma$. Hence, $\gamma \in \mathcal{T}_\Sigma$, therefore, $T = f(\overline{L})\gamma \in \mathcal{T}_\Sigma$.

To show that $\vdash_{\mathcal{R}} T : s$, it is enough to show that $T :_{\mathcal{R}} t$ by using the membership $f(\overline{L}) : t$ **if** P, that is, $\vdash_{\mathcal{R}} P\gamma$. By the induction hypothesis, this is true of the membership atoms $U\gamma : t \in P\gamma$, since $L(\succ \cup \rhd)U$, and therefore $L\gamma(\succ \cup \rhd)U\gamma$. We show now that this is true of the other atoms $V\gamma \downarrow W\gamma$. Note that $V\gamma$ and $W\gamma$ are both in $\mathcal{T}_\Sigma$, since $V, W \in \mathcal{T}_\Sigma(\mathcal{X})$ and $\gamma \in \mathcal{T}_\Sigma$. Besides, we already have $\vdash_{\mathcal{R}'} V\gamma \downarrow W\gamma$, hence, by confluence of $\mathcal{R}'$, $(V\gamma) \downarrow_{\mathcal{R}'} = (W\gamma) \downarrow_{\mathcal{R}'}$. By assumption (iii) and since $V\gamma$ and $W\gamma$ are in $\mathcal{T}_\Sigma$ and have the same sort in $\mathcal{S}$ by assumption (i), $(V\gamma) \downarrow_{\mathcal{R}'} = (V\gamma) \downarrow_{\mathcal{R}}$ and $(W\gamma) \downarrow_{\mathcal{R}'} = (W\gamma) \downarrow_{\mathcal{R}}$, therefore $(V\gamma) \downarrow_{\mathcal{R}} = (W\gamma) \downarrow_{\mathcal{R}}$, hence $\vdash_{\mathcal{R}} V\gamma \downarrow W\gamma$ and we are done. $\Box$

Lemma 59 *Let $\mathcal{R} \subseteq \mathcal{R}'$ be both ground confluent, ground sort-decreasing and reductive, and let $\mathcal{R}' = \mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}} \cup \mathcal{M}'_{\mathcal{S}'\setminus\mathcal{S}}$ as before. Assume further that:*

*(i) each membership in $\mathcal{R}$ is of the form $f(\overline{L}) : s$ **if** $\mathcal{V}ar(\overline{L}) : \overline{s} \wedge \overline{U} : \overline{t} \wedge \overline{V} \downarrow \overline{W}$ where $\mathcal{V}ar(\overline{U}, \overline{V}, \overline{W}) \subseteq \mathcal{V}ar(\overline{L})$ and for each $M \downarrow N \in \overline{V} \downarrow \overline{W}$, there exists some sort $t \in \mathcal{S}$ such that $M :^*_{\mathcal{V}ar(\overline{L}):\overline{s},\mathcal{R}} t$ and $N :^*_{\mathcal{V}ar(\overline{L}):\overline{s},\mathcal{R}} t$.*

(ii) $\forall s' \in \mathcal{S}', s \in \mathcal{S}$ such that $s' \leq_{\mathcal{R}'} s$, then $s' \in \mathcal{S}$ and $s' \leq_{\mathcal{R}} s$,

(iii) $\forall S \in \mathcal{T}_\Sigma$ such that $S :^*_{\mathcal{R}} s$ with $s \in \mathcal{S}$ and $S = S \downarrow_{\mathcal{R}}$, then $S \downarrow_{\mathcal{R}'} = S$.

Let T be a term in $\mathcal{T}_{\Sigma'}$ such that $T :^*_{\mathcal{R}'} s$ for some sort $s \in \mathcal{S}$. Then, $T :^*_{\mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}'\setminus\mathcal{S}}} s$.

Proof: We prove the stronger property that for all T and S in $\mathcal{T}_{\Sigma'}$ and $s \in \mathcal{S}$,

(a) if $T :^*_{\mathcal{R}'} s$, then $T :^*_{\mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}'\setminus\mathcal{S}}} s$,

(b) if $T \longrightarrow^*_{\mathcal{R}'} S$ then $T \longrightarrow^*_{\mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}'\setminus\mathcal{S}}} S$.

This is done by induction on T with respect to the ordering $(\succ \cup \rhd)$.

Base case: assume that T is in $\mathcal{R}'$-normal form. Then, $S = T$, settling part (b) of the assumption. For part (a), since h_s is surjective by assumption that $\mathcal{R} \subseteq \mathcal{R}'$ is a protecting extension, there must exist a term $T' \in \mathcal{T}_\Sigma$ with $T' :^*_{\mathcal{R}} s$, such that $T \longleftrightarrow^*_{\mathcal{R}'} T'$. Let now $T'' = T' \downarrow_{\mathcal{R}}$. Since $\mathcal{R}$ is sort-decreasing, $T'' \in \mathcal{T}_\Sigma$ and $T'' :_{\mathcal{R}} t$ for some $t \leq_{\mathcal{R}} s$. By assumption (iii), $T'' \downarrow_{\mathcal{R}'} T''$, hence $T = T''$ by assumption that $\mathcal{R}'$ is ground-confluent, and therefore $T \in \mathcal{T}_\Sigma$ with $T :_{\mathcal{R}} t$, and, a fortiori, $T :^*_{\mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}'\setminus\mathcal{S}}} s$, which concludes the base case.

Induction step: assume that $T \longrightarrow_{\mathcal{R}'} T'$ with a rule $L \to R$ **if** P and a substitution γ, and hence $\vdash_{\mathcal{R}'} P\gamma$. We will first show that $T \longrightarrow_{\mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}'\setminus\mathcal{S}}} T'$, and then conclude by induction on T'. Since the reasoning is the same for both parts of the statement, we do it for part (a) only. Since $\mathcal{R}'$ is reductive, the conditions justifying the rewrite step involve terms smaller than T. Let $U : u \in P\gamma$, hence $U :^*_{\mathcal{R}'} u$ by assumption that $\vdash_{\mathcal{R}'} P\gamma$. By the induction hypothesis, $U :^*_{\mathcal{R}\cup\mathcal{E}'\cup\mathcal{M}'_{\mathcal{S}'\setminus\mathcal{S}}} u$. Let now $V \downarrow W \in P\gamma$, and hence $V \downarrow_{\mathcal{R}'} W$ by assumption that $\vdash_{\mathcal{R}'} P\gamma$. Now, by the induction hypothesis applied successively to V and U, $V \downarrow_{\mathcal{R}\cup\mathcal{E}'\cup\mathcal{M}'_{\mathcal{S}'\setminus\mathcal{S}}} W$. We have therefore proved that $T \longrightarrow_{\mathcal{R}\cup\mathcal{E}'\cup\mathcal{M}'_{\mathcal{S}'\setminus\mathcal{S}}} T'$.

Now, since $\mathcal{R}'$ is ground sort-decreasing, $T' :^*_{\mathcal{R}'} s$ and we can conclude by the induction hypothesis applied to T'. □

We now need to turn our attention to the elaboration of techniques allowing the use of Theorem 57 in a language like Maude, where protecting extensions are declared by the user and must be checked with the help of the theorem proving environment. This implies checking the validity of applying the theorem, that is, conditions (i), (ii) and (iii), and the conditions ensuring that an extension is protecting, that is, property (*).

Assumptions (i) and (ii) can be checked statically in Maude, since they refer to syntactic properties of the module definitions.

In the following sections, we give methods for checking (iii) and (*). We will first show that both are decidable in the case of (unconditional) order-sorted specifications, by means of tree automata techniques. We will then consider the case of conditional rules, for which constructor specifications will play an important role.

8.2 Order-sorted Specifications

Recall that in an order-sorted specification, the conditions in a rule are simply memberships for its variables.

Proposition 60 *Let $\mathcal{R} = (\mathcal{K}, \Sigma, \mathcal{S}, \mathcal{R})$ and $\mathcal{R}' = (\mathcal{K}', \Sigma', \mathcal{S}', \mathcal{R}')$ be two specifications in membership equational logic, such that $\mathcal{R} \subseteq \mathcal{R}'$. Assume further that $\mathcal{R}$ is a ground confluent, terminating, order-sorted rewrite system, and $\mathcal{R}'$ is a ground confluent CRMS such that each rewrite (resp. membership) rule whose lefthand side is in $\mathcal{T}_\Sigma$ is an order-sorted rewrite (resp. membership) rule.*

Then, it is decidable whether for any sortable term T in $\mathcal{T}_\Sigma$, T is in $\mathcal{R}$-normal form iff T is in $\mathcal{R}'$-normal form.

Proof: Since the rules in $\mathcal{R}$ are order-sorted, we can construct a tree automaton $\mathcal{A}$ (possibly with membership/equality tests if the rules are not left-linear) recognizing the set of ground sortable terms in $\mathcal{R}$-normal form. This automaton can be obtained, for example, by intersecting the normal form automaton with respect to the rewrite rules in $\mathcal{R}$ with the automaton recognizing the set of sortable terms with respect to the membership rules in $\mathcal{R}$. Of course, a direct construction would be more efficient.

Now, the terms in $\mathcal{T}_\Sigma$ which are in $\mathcal{R}'$-normal form can be obtained in the same way under our assumption on the rules in $\mathcal{R}'$, resulting in an automaton $\mathcal{A}'$. Since inclusion of recognizable tree languages (possibly in the sense of automata with membership/equality tests) is decidable, we can decide whether the ground terms accepted by $\mathcal{A}$ are still accepted by $\mathcal{A}'$, which achieves the proof. □

We first give a sufficient condition for protection by replacing property (*) by a stronger one.

Proposition 61 *Let $(\mathcal{K},\Sigma,\mathcal{S},\mathcal{R}) \subseteq (\mathcal{K}',\Sigma',\mathcal{S}',\mathcal{R}')$ be specifications such that $\mathcal{K} \subseteq \mathcal{K}'$ and $\mathcal{S} \subseteq \mathcal{S}'$, and assume that $\mathcal{R}$ and $\mathcal{R}'$ are both ground confluent, sort decreasing and reductive systems satisfying properties (i) to (iii). Assume further the property*

(iv) for each membership $S : s$ `if` $\mathcal{V}ar(\overline{S}) : \overline{s} \wedge \overline{U} : \overline{t} \wedge \overline{V} \downarrow \overline{W}$ in $\mathcal{M}'_{\mathcal{S}}$, we have $S\gamma$ is reducible for every ground substitution γ such that $\mathcal{V}ar(\overline{S})\gamma :^* \overline{s}$, $\overline{U}\gamma :^* \overline{t}$, and $\overline{V}\gamma \downarrow \overline{W}\gamma$.

Then, $\mathcal{R} \subseteq \mathcal{R}'$ is a protecting extension.

Proof: Using the notations of Theorem 57, we show by induction on the ordering $\succ \cup \rhd$ used for showing reductivity of $\mathcal{R}'$ that for every ground term $T \in \mathcal{T}_{\Sigma'}$ and sort $s \in \mathcal{S}$, $T :^*_{\mathcal{R}'} s$ implies $T :^*_{\mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}' \setminus \mathcal{S}}} s$. This property implies property (*), and we conclude with Theorem 57.

Assume first that T is $\mathcal{R}'$-irreducible, hence T is a ground instance of a membership rule which must belong to $\mathcal{M}'_{\mathcal{S}' \setminus \mathcal{S}}$ by assumption (iv). We conclude by showing as in the proof of Lemma 59, by means of the induction hypothesis, that the conditions of the rules follow from the axioms in $\mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}' \setminus \mathcal{S}}$.

Assume now that T is $\mathcal{R}'$-reducible, and let $T\downarrow$ be the $\mathcal{R}'$-normal form of T. Since T is $\mathcal{R}'$-reducible, $T\downarrow$ is strictly smaller than T in our ordering. By definition that $T :_{\mathcal{R}'} s$, there exists a term T' and a sort $t \leq_{\mathcal{R}'} s$ such that $T \longrightarrow^*_{\mathcal{R}'} T'$ and $T' :_{\mathcal{R}'} t$. By assumption (ii), $t \in \mathcal{S}$ and $t \leq_{\mathcal{R}} s$. By confluence, $T' \longrightarrow^*_{\mathcal{R}'} T\downarrow$, and by sort-decreasingness, $T\downarrow :^*_{\mathcal{R}'} t$, hence $T\downarrow :^*_{\mathcal{R}'} s$. By induction hypothesis, $T\downarrow :^*_{\mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}' \setminus \mathcal{S}}} s$. To conclude, we show as in Lemma 59 that $T \longrightarrow^*_{\mathcal{R} \cup \mathcal{E}' \cup \mathcal{M}'_{\mathcal{S}' \setminus \mathcal{S}}} T\downarrow$. □

Considering now the case of order-sorted specifications, we know already that properties (i) to (iii) can be decided. We show easily that this is true of (iv) as well by using again tree automata techniques. In this case indeed, being order-sorted, memberships are of the simpler form $S : s$ `if` $\mathcal{V}ar(\overline{S}) : \overline{s}$. Therefore, the ground substitutions γ to be considered simply assert the membership to some sort for all their variables. We need then to check if $S\gamma$ is reducible for each such γ, that is, the ground reducibility of the order-sorted term (S, γ). We have seen already that this property is decidable in the context of Church-Rosser order-sorted specifications. As a conclusion:

Theorem 62 *Let $(\mathcal{K},\Sigma,\mathcal{S},\mathcal{R}) \subseteq (\mathcal{K}',\Sigma',\mathcal{S}',\mathcal{R}')$ be order-sorted specifications such that $\mathcal{K} \subseteq \mathcal{K}'$ and $\mathcal{S} \subseteq \mathcal{S}'$, and assume that $\mathcal{R}$ and $\mathcal{R}'$ are both ground confluent, sort decreasing and reductive systems. Then $\mathcal{R} \subseteq \mathcal{R}'$ is a protecting extension if the decidable properties (i) to (iv) are satisfied.*

Note that we can replace $\mathcal{R}'$ by the result of its completion in case $\mathcal{R}'$ is not confluent. Of course, we can check property (iv) at completion time, which is very much reminiscent of inductive completion.

8.3 Protection of Constructor Specifications

We now consider specifications which are not order-sorted, that is, their rules are conditional. We first consider how to prove property (iii).

We can give a simple natural condition under which assumption (iii) is satisfied when the rules are conditional:

For each rule $L \to R$ `if` C such that $L \in \mathcal{T}_\Sigma(\mathcal{X})$, and for each substitution $\sigma \in \mathcal{T}_\Sigma$ and sort $s \in \mathcal{S}$ such that $\vdash_{\mathcal{R}'} C\sigma$ and $L\sigma :^*_{\mathcal{R}'} s$, then $L\sigma$ is $\mathcal{R}$-reducible.

Of course, this reformulation is still undecidable, but sufficient conditions can now be found, by taking for example all possible substitutions σ replacing variables by terms of the appropriate sort of the form $f(\overline{x})$, and applying then narrowing (see Section 10.4) to solve the obtained constraint.

Note also that we can replace $L\sigma :^*_{\mathcal{R}'} s$ by $L\sigma :^*_{\mathcal{R}} s$ if property (*) is satisfied, because $L\sigma :^*_{\mathcal{R}'} s$ is equivalent to $L\sigma \longrightarrow^*_{\mathcal{R}'} :_{\mathcal{R}'} s$, which is itself equivalent to $L\sigma \longrightarrow^*_{\mathcal{R}} :_{\mathcal{R}'} s$ by the above property, ground confluence and sort-decreasingness of $\mathcal{R}'$, hence is finally equivalent to $L\sigma \longrightarrow^*_{\mathcal{R}} :_{\mathcal{R}} s$ by property (*).

We now consider proving property (*). It can of course be checked by using an inductive theorem proving facility, such as the one described in Section 9. We will improve this straightforward remark in the important subcase of so-called constructor specifications. Our hypothesis of a *complete specification* turns out to be much more general than the usual one, thanks to the additional expressive power of membership equational logic.

8.3.1 Complete Constructor Specifications

Let $\mathcal{C} = (\mathcal{K}, \mathcal{C}, \mathcal{S}, \mathcal{R}_{\mathcal{C}}) \subseteq \mathcal{D} = (\mathcal{K}', \Sigma = \mathcal{C} \cup \mathcal{D}, \mathcal{S}', \mathcal{R} = \mathcal{R}_{\mathcal{C}} \cup \mathcal{R}_{\mathcal{D}})$ be specifications such that $\mathcal{K} \subseteq \mathcal{K}'$ and $\mathcal{S} \subseteq \mathcal{S}'$. We assume that the properties (i),(ii) and (iii) of Theorem 57 are satisfied, and our goal is to prove property (*), that is, we want to show that $\mathcal{D}$ is a protecting extension of $\mathcal{C}$. We will call the function symbols in $\mathcal{C}$ the *constructors*, and the function symbols in $\mathcal{D}$ the *defined symbols*. Although we use the same familiar vocabulary, we want to stress that the notions of constructor and defined symbols that we are going to define now are far more general that what is usually found in the litterature.

Apart from [10], the existing algorithms for checking completeness of definitions in the case of constructor specifications assume either that constructors are free [8], or that rules for defined symbols are unconditional [54].

But constructor symbols may be free for some sorts, and completely defined in all other sorts. For example, the successor function S is free on $Zero$ ands Pos and completely defined on Neg in the specification given in 5.1. Formalizing this observation requires exploiting the expressive power of membership equational logic. Generalizing [10], we assume given a *complete specification*, which comes in two parts: a *complete specification of constructor symbols* and a *complete specification of defined symbols*.

Complete specification of constructor symbols

Definition 63 *Given a specification $\mathcal{C}$ whose function symbols are called constructors, a constructor $C : K_1 \times \ldots \times K_n \to K$ is* free *at sorts $s_1 \times \ldots \times s_n$ if $C(x_1, \ldots, x_n)$ is sortable and ground irreducible in the environement $\{x_1 : s_1, \ldots, x_n : s_n\}$.*

A constructor $C : K_1 \times \ldots \times K_n \to K$ is defined *at sorts $s_1 \times \ldots \times s_n$ if $C(x_1, \ldots, x_n)$ is sortable and ground reducible in the environment $\{x_1 : s_1, \ldots, x_n : s_n\}$.*

A constructor $C : K_1 \times \ldots \times K_n \to K$ is complete *at sort $s_1 \times \ldots \times s_n$ if there exists a cover sort $(s_1 \times \ldots \times s_n)_{\leq} = \{s^i_1 \times \ldots \times s^i_n\}_{i \in I}$ of $s_1 \times \ldots \times s_n$ such that C is free at all sorts in some subset $\mathcal{Q}$ of $(s_1 \times \ldots \times s_n)_{\leq}$ and is defined at all sorts in its complement $(s_1 \times \ldots \times s_n)_{\leq} \setminus \mathcal{Q}$.*

A constructor $C : K_1 \times \ldots \times K_n \to K$ is complete *if:*

(i) C is complete at all sorts $s_1 \times \ldots \times s_n \in K_1 \times \ldots \times K_n$ such that $(C(\overline{x}), \{x_i : s_i\}_{i \in [1..n]})$ is ground sortable,

(ii) $C(\overline{x})$ inhabits a minimal sort if the variables in $\overline{x}$ inhabit minimal sorts.

A specification of constructors is complete *if each constructor is complete.*

Theorem 64 ([20]) *It is decidable whether a specification of constructors in membership equational logic is complete.*

In practice, rather than checking, we would of course prefer transforming a given specification of constructors into a complete one. It turns out that this can be easily done by using the tree automata technique given in Section 5, under the natural assumption that the rewrite rules for constructors are unconditional order-sorted rules, therefore allowing us to compute the normal form automaton, whose final states are exactly the free sorts of the specification.

Complete specification of defined symbols

Definition 65 *Let $\mathcal{C} = (\mathcal{K}, \mathcal{C}, \mathcal{S}, \mathcal{R}_{\mathcal{C}}) \subseteq \mathcal{D} = (\mathcal{K}', \Sigma = \mathcal{C} \cup \mathcal{D}, \mathcal{S}', \mathcal{R} = \mathcal{R}_{\mathcal{C}} \cup \mathcal{R}_{\mathcal{D}})$ be specifications such that $\mathcal{K} \subseteq \mathcal{K}'$ and $\mathcal{S} \subseteq \mathcal{S}'$, and assume that $\mathcal{R}$ is a reductive CRMS. A function symbol $f \in \mathcal{D}$ is* completely defined *relative to the constructor subspecification $\mathcal{C}$ iff for each term T of the form $f(T_1, \ldots, T_n)$ where for all $1 \leq i \leq n$, $T_i \in \mathcal{T}_{\mathcal{C}}$ and $T :_{\mathcal{R}} s$ for some sort s, there exists $T' \in \mathcal{T}_{\mathcal{C}}$ such that $f(T_1, \ldots, T_n) \rightarrow^+_{\mathcal{R}} T'$ and $T' :_{\mathcal{R}_{\mathcal{C}}} t$ for some $t \leq_{\mathcal{R}_{\mathcal{C}}} s$. The specification $\mathcal{D}$ is* complete *relative to the constructor subspecification $\mathcal{C}$ iff each defined symbol $f \in \mathcal{D}$ is completely defined.*

Protecting extensions and definedness are two different notions, but the following proposition gives sufficient conditions under which they become equivalent:

Theorem 66 *Let $\mathcal{C} = (\mathcal{K}, \mathcal{C}, \mathcal{S}, \mathcal{R}_{\mathcal{C}}) \subseteq \mathcal{D} = (\mathcal{K}', \Sigma = \mathcal{C} \cup \mathcal{D}, \mathcal{S}', \mathcal{R} = \mathcal{R}_{\mathcal{C}} \cup \mathcal{R}_{\mathcal{D}})$ be specifications such that $\mathcal{K} \subseteq \mathcal{K}'$ and $\mathcal{S} \subseteq \mathcal{S}'$, and assume that $\mathcal{R}$ is a reductive, ground-confluent and ground sort-decreasing CRMS satisfying properties (i),(ii) and (iii). Assume further the property:*
(i') for each membership rule $f(\overline{L}) : s$ $\mathtt{if}$ $\mathcal{V}ar(\overline{L}) : \overline{s} \wedge C$ in $\mathcal{M}'_{\mathcal{S}}$, $\overline{s} \subseteq \mathcal{S}$.
Then $\mathcal{D}$ is a protecting extension of $\mathcal{C}$ iff it is completely defined relative to $\mathcal{C}$.

Proof: Assume first that $\mathcal{D}$ is completely defined relative to $\mathcal{C}$. We prove by induction on the size of terms, a property more general than (*), namely that $\forall T \in \mathcal{T}_{\Sigma}$ such that $T :^*_{\mathcal{R}} s$ for some $s \in \mathcal{S}$, $\exists T' \in \mathcal{T}_{\mathcal{C}}$ such that $T \longrightarrow^*_{\mathcal{R}} T' :_{\mathcal{R}_{\mathcal{C}}} t$ for some $t \leq_{\mathcal{R}_{\mathcal{C}}} s$. We can then conclude by using Theorem 57. The proof of the property itself is by induction on the ordering $\succ$ used to show reductivity of $\mathcal{R}$.

Base case: T is in normal form with respect to $\mathcal{R}$. First, an easy induction on size and use of assumptions (i) and (i') ensure that every subterm of T inhabits a sort in $\mathcal{S}$. We then prove that $T \in \mathcal{T}_{\mathcal{C}}$. Assume indeed that T has a subterm of the form $f(\overline{T})$, with $f \in \mathcal{D}$ and $T_i \in \mathcal{T}_{\mathcal{C}}$ for each $T_i \in \overline{T}$. Then, $f(\overline{T}) :_{\mathcal{R}} t$ for some $t \in \mathcal{S}$ by the above remark, hence it is reducible by the assumption that the specification is completely defined, resulting in a contradiction. Hence $T \in \mathcal{T}_{\mathcal{C}}$, and $T :_{\mathcal{R}_{\mathcal{C}}} s$.

Inductive step: T is $\mathcal{R}$-reducible. Then $T \longrightarrow_{\mathcal{R}} T'$ and $T' :^*_{\mathcal{R}} t$ for some $t \leq_{\mathcal{R}} s$. Hence, $t \in \mathcal{S}$ and $t \leq_{\mathcal{R}_{\mathcal{C}}} s$ by assumption (ii). We then conclude by the induction hypothesis applied to T'.

For the converse, we first apply Theorem 57 to obtain property (*). We then show as previously the more general property that $\forall T \in \mathcal{T}_{\Sigma}$ such that $T :^*_{\mathcal{R}} s$ for some $s \in \mathcal{S}$, $\exists T' \in \mathcal{T}_{\mathcal{C}}$ such that $T \longrightarrow^*_{\mathcal{R}} T' :_{\mathcal{R}_{\mathcal{C}}} t$ for some $t \leq_{\mathcal{R}_{\mathcal{C}}} s$. The proof is again by induction on the ordering $\succ \cup \rhd$.

Base case: T is in normal form with respect to $\mathcal{R}$. As previously remarked, all subterms of T are in normal form, and have a sort in $\mathcal{S}$. If $T \in \mathcal{T}_{\mathcal{C}}$, we are done. Otherwise, T has a subterm $f(\overline{T})$ of sort t which is in normal form with respect to $\mathcal{R}$. Then, $f(\overline{T}) :_{\mathcal{M}'_{\mathcal{S}}} t$. Using (*), we can replace this elementary membership proof by a membership proof using the axioms in $\mathcal{R} \setminus \mathcal{M}'_{\mathcal{S}}$. But, no membership can apply, since $f \in \mathcal{D}$ and $s \in \mathcal{S}$. Hence, $f(\overline{T}) \longrightarrow_{\mathcal{R}} T'$, contradicting the assumption that T was in normal form.

Inductive step: If $T \longrightarrow_{\mathcal{R}} T'$, then $T' :^*_{\mathcal{R}} t$ for some $t \leq_{\mathcal{R}} s$ by ground sort-decreasingness of $\mathcal{R}$, and indeed $t \in \mathcal{S}$ and $t \leq_{\mathcal{R}_{\mathcal{C}}} s$ by assumption (ii). We can now conclude by the induction hypothesis applied to T', which achieves the proof. □

Back to sufficient completeness It results from the previous theorems that sort declarations for the defined symbols are superfluous when computing with ground terms. For example, in the NUMBER specification, it is true that the product of a (ground) complex (term) by its conjugate is an integer: this property is an inductive consequence of the rest of the specification. More precisely, all sort declarations involving defined symbols are indeed true in the initial model of the specification obtained by removing such declarations: they are theorems for free, to follow a felicitous turn of phrase by Wadler.

Although general, these remarks apply to the particular case of a constructor specification, in its restrictive usual acceptation: no new kind, no new sort, no new constructor, but only defined symbols can be added together with their defining rules. With our notations, we write $\mathcal{C} = (\mathcal{K}, \mathcal{C}, \mathcal{S}, \mathcal{R}_{\mathcal{C}}) \subseteq \mathcal{D} = (\mathcal{K}, \Sigma = \mathcal{C} \cup \mathcal{D}, \mathcal{S}, \mathcal{R} = \mathcal{R}_{\mathcal{C}} \cup \mathcal{R}_{\mathcal{D}})$. Theorem 66 can now be restated for this particular case:

Theorem 67 *Let $\mathcal{C} = (\mathcal{K}, \mathcal{C}, \mathcal{S}, \mathcal{R}_{\mathcal{C}})$ be a specification of constructors, and $\mathcal{D} = (\mathcal{K}, \mathcal{C} \cup \mathcal{D}, \mathcal{S}, \mathcal{R} = \mathcal{R}_{\mathcal{C}} \cup \mathcal{R}_{\mathcal{D}})$ a complete specification relative to C. Assume that $\mathcal{R}$ is a confluent, ground sort-decreasing, reductive CRMS satisfying properties (i) and (iii). Let $\mathcal{M}_{\mathcal{D}}$ be the set of membership rules in $\mathcal{R}$ whose head contains a defined symbol.*

Then, all memberships in $\mathcal{M}_{\mathcal{D}}$ are inductive consequences of $\mathcal{R}' = \mathcal{R} - \mathcal{M}_{\mathcal{D}}$, and $\mathcal{R}'$ is a reductive system which is confluent, ground sort-decreasing, and has the same ground normal forms as $\mathcal{R}$.

Theorem 67 emphasizes the role of complete definitions, instead of protecting specifications, therefore suggesting new ways to prove protection without dealing with property (*) directly. Indeed, the literature about sufficient completeness is quite prolific.

8.3.2 A Completion Technique for Ground Confluence

An alternative approach to completion is suggested by Theorem 67: first, assume that the constructor specification (if constructors are not free) is reductive, confluent and sort decreasing. This assumption can of course be verified, and if necessary enforced by an appropriate completion process, which should be rather simple since the constructor rules are usually mostly non-conditional. Then, add the rewrite rules for defined symbols, and assume sufficient completeness. Sort-decreasingness is of course satisfied for the whole set of rules. Now, completion can be started, but this is a many-sorted algebra completion. We simply need to orient rules in such a way that lefthand sides are not constructor terms. When this process is completed, we can add the membership rules for the defined symbols, and we know that this does not destroy confluence. We can even start a new completion process operating on these memberships in order to ensure regularity. It is clear that this method should give better results than the previous one, since sort-decreasingness is to be checked for the constructor subspecification only.

Unfortunately, checking sufficient completeness assumes ground confluence. Luckily enough, the reasoning is still correct: assume sufficient completeness was proved without the memberships for the assumed defined symbols. Then adding them back may destroy confluence, because a critical pair can now have additional (non-confluent) instances. This means that the added membership $g(\overline{L}) : s$ `if` C had an instance $g(\overline{L}\gamma)$ such that $g(\overline{L}\gamma)$ does not rewrite to a constructor term of sort s, and hence, by sort-decreasingness, does not rewrite to a constructor term at all. Now, when testing $g(\overline{L})$ for completeness, this will eventually show up. The argument is that the completeness of $g(\overline{L})$ calls for the completeness of other terms like $h(\overline{L'})$, such that $g(\overline{L}\gamma)(\succ \cup \rhd)h(\overline{L'})$. This process must of course be finite by the reductivity property. Now, it may be that the inductive oracle is called to verify that all cases have been exhausted for $g(\overline{L})$. However, the condition sent to the inductive prover may only contain terms smaller than $G(\overline{L}\gamma)$ by reductivity property. These terms must in turn have been checked for sufficient completeness, but they are smaller, so we can argue by induction.

Note that if $f(\overline{L}) \to R$ `if` C, we need to consider for ground reducibility all occurrences of defined symbols in R *and* in C, since we need to evaluate recursively C.

Chapter 9

Proofs by Induction in Parameterized Specifications

9.1 Inductive Theory

Parameterization is very important for building up larger data types and software systems from generic specifications in a highly reusable way. The notion of initial algebra semantics generalizes in a natural way to that of *free extension* of an algebra of the parameter theory to an algebra of the body specification. Since inductive properties are properties of an initial algebra, they also generalize in a natural way to inductive properties of the free extension models of parameterized specifications.

We shall now define parameterized specifications.

Definition 68 *A* parameterized specification *is a pair of membership equational logic specifications $\mathcal{J} = (\mathcal{P}, \mathcal{B})$ with $\mathcal{P} \subseteq \mathcal{B}$. We call $\mathcal{P} = (\Omega_{\mathcal{P}}, \mathcal{E}_{\mathcal{P}})$ the* parameter *specification, and $\mathcal{B} = (\Omega_{\mathcal{B}}, \mathcal{E}_{\mathcal{B}})$ the* body *specification, where $\Omega_{\mathcal{P}} = (\mathcal{K}_{\mathcal{P}}, \mathcal{S}_{\mathcal{P}}, \Sigma_{\mathcal{P}})$, $\Omega_{\mathcal{B}} = (\mathcal{K}_{\mathcal{B}}, \mathcal{S}_{\mathcal{B}}, \Sigma_{\mathcal{B}})$, $\mathcal{E}_{\mathcal{P}}$ is the set of parameter* constraints *consisting of Horn clauses over $\Omega_{\mathcal{P}}$, and $\mathcal{E}_{\mathcal{B}}$ is the set of axioms of the parameterized specification. Similarly, we assume that the axioms in $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ are Horn clauses over $\Omega_{\mathcal{B}}$.*

In practice, for efficient deduction, we will always assume that the Horn clauses in $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ are indeed reductive rewrite or membership rules. All our examples will satisfy this additional assumption.

Example 69 *Figures 9.2 and 9.1 give an example of a parameterized specification* [1]*. To prove the termination of $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$, we can use the* lexicographic path ordering $\prec$ *(see for instance [27]) with the following precedence on functions:*

$$false \prec true \prec \leq \prec nil \prec Cons \prec sorted \prec insert$$

We can easily check that the specification NAT given in Figure 9.3 is an instance of the parameter specification TOSET, since all the parameter constraints are inductive consequence of the specification NAT.

[1] We use Maude-like notation for the parameter theory, with keyword `fth` for the parameter theory of totally ordered sets TOSET, which is of course interpreted loosely, and keyword `fmod` for the body specification SORTING which is interpreted freely relative to the parameter theory. Note that this example is somewhat more general than the parameterized specifications covered by Definition 68. The extra generality is given by the `protecting` BOOL declaration in TOSET, which requires the submodule BOOL specifying the standard Booleans to be interpreted *initially*. Our proof methods can handle the extra generality of this example, as we shall explain later.

```
fth TOSET is protecting BOOL
kind Elem?[Elem]
op _≤_ : Elem? × Elem? →Bool?
mb x ≤ y : Bool if x,y:Elem
eq x ≤ x = true if x:Elem
eq x ≤ z = true if x ≤ y = true and y≤ z = true and x,y,z:Elem
eq x = y if x ≤ y = true and y ≤ x = true and x,y:Elem
eq (x ≤ y) or (y ≤ x) = true if x,y:Elem
endfth
```

Figure 9.1: The parameter specification TOSET

As already mentioned in Section 3.1, an atom is either an equation $e = e'$ where $e,\ e' \in \mathcal{T}_\Sigma(\mathcal{X})_K$ for K a kind, or a membership assertion $e : s$ where $e \in \mathcal{T}_\Sigma(\mathcal{X})_K$ and s is a sort of kind K. A *clause* is an expression of the form $\neg\varphi_1 \vee \cdots \vee \neg\varphi_n \vee \varphi'_1 \cdots \vee \varphi'_m$, where the $\varphi_1, \ldots, \varphi_n,\ \varphi'_1, \ldots, \varphi'_m$ are atoms. A *positive* clause is a clause with only positive atoms.

In this chapter, goals to be proved are arbitrary clauses.

9.1.1 Semantics of Parameterized Specifications

Given a parameterized specification $\mathcal{J} = (\mathcal{P}, \mathcal{B})$, each membership algebra $B \in Alg_{\Omega_\mathcal{B},\mathcal{E}_\mathcal{B}}$ can be "restricted to the parameter $\mathcal{P}$" by forgetting about all the extra kinds, sorts, and operations in $\Sigma_\mathcal{B} - \Sigma_\mathcal{P}$. In this way we get a $\mathcal{P}$-algebra $\mathcal{B}|_\mathcal{P} \in Alg_{\Omega_\mathcal{P},\mathcal{E}_\mathcal{P}}$. Such a restriction is in fact a functor

$$-|_\mathcal{P} : Alg_{\Omega_\mathcal{B},\mathcal{E}_\mathcal{B}} \to Alg_{\Omega_\mathcal{P},\mathcal{E}_\mathcal{P}}$$

It follows from a more general theorem in [74] that this restriction functor has a left adjoint providing a free extension of each $\mathcal{P}$-algebra $A \in Alg_{\Omega_\mathcal{P},\mathcal{E}_\mathcal{P}}$ to a $\mathcal{B}$-algebra $F_\mathcal{J}(A) \in Alg_{\Omega_\mathcal{B},\mathcal{E}_\mathcal{B}}$ with a $\mathcal{P}$-homomorphism $\eta_A : A \to F_\mathcal{J}(A)|_\mathcal{P}$ such that for each $\mathcal{B}$-algebra $B \in Alg_{\Omega_\mathcal{B},\mathcal{E}_\mathcal{B}}$ and each $\mathcal{P}$-homomorphism $f : A \to B|_\mathcal{P}$ there exists a unique $\mathcal{B}$-homomorphism $\bar{f} : F_\mathcal{J}(A) \to B$ making the following diagram commute

$$\begin{tikzcd}
A \arrow[r, "\eta_A"] \arrow[dr, "f"'] & F_{\mathcal{J}}(A)|_{\mathcal{P}} \arrow[d, "\bar{f}|_{\mathcal{P}}"] \\
 & B|_{\mathcal{P}}
\end{tikzcd}$$

The explicit construction of $F_\mathcal{J}(A)$ specializing that in [74] is as follows. To each $\mathcal{P}$-algebra $A \in Alg_{\Omega_\mathcal{P},\mathcal{E}_\mathcal{P}}$, we associate a new signature $\Omega_\mathcal{B}(A) = (\mathcal{K}_\mathcal{B}, \mathcal{S}_\mathcal{B}, \Sigma_\mathcal{B}(A))$, where $\Sigma_\mathcal{B}(A)$ extends $\Sigma_\mathcal{B}$ by adding to it, for each kind $K \in \mathcal{K}_\mathcal{P}$, the elements of A_K as disjoint new constants of sort K. Similarly, $\Omega_\mathcal{P}(A)$ will denote the signature that extend $\Omega_\mathcal{P}$ by adding to $\Sigma_\mathcal{P}(A)$, for each kind $K \in \mathcal{K}_\mathcal{P}$, the elements of A_K as disjoint new constants of sort K. Then, we can associate to such a $\mathcal{P}$-algebra A the following set $\Delta(A)$ of $\Omega_\mathcal{P}(A)$-axioms:

```
fmod SORTING[T::TOSET]
kind List?[Nelist < List]
cop  nil      :                      → List?
cop  cons_ _  : Elem? × List?        → List?
op   insert_ _ : Elem? × List?       → List?
op   sorted_  : List?                → Bool?
op   last_    : List?                → Elem?
mb  nil : List
mb  cons(x,y):Nelist if x:Elem and y:List
mb  insert(x,y):Nelist if x:Elem and y:List
mb  sorted(x):Bool if x:List
mb  last(x):Elem if x:Nelist
eq  insert(x,nil) → cons(x,nil) if x:Elem
eq  insert(x,cons(y,z)) → cons(x,cons(y,z)) if x ≤ y=true and x,y: Elem and z:List
eq  insert(x,cons(y,z)) → cons(y,insert(x,z)) if x ≤ y=false and x,y: Elem and z:List
eq  sorted(nil) → true
eq  sorted(cons(x,nil)) → true if x:Elem
eq  sorted(cons(x,cons(y,z))) → false if x ≤ y=false and x,y: Elem and z:List
eq  sorted(cons(x,cons(y,z))) → sorted(cons(y,z)) if x ≤ y=true and x,y: Elem and z:List
eq  last(cons(x,nil)) → x if x:Elem
eq  last(cons(x,y)) → last(y) if x:Elem and y:Nelist
end fmod
```

Figure 9.2: The parameterized specification SORTING

```
fmod NAT protects BOOL
kind Nat?[Zero, Pos < Nat]
cop 0 : Zero
cop S_ : Nat → Nat
op _ ≤ _ : Nat? × Nat? → Bool?
mb x ≤ y : Bool if x,y: Nat
eq 0 ≤ x = true if x:Nat
eq S(x) ≤ 0 = false if x:Nat
eq S(x) ≤ S(y) = x ≤ y if x,y:Nat
end fmod
```

Figure 9.3: A specification of the predicate $\leq$ on natural numbers

(i) For each $f : K_1 \ldots K_n \to K$ in $\Sigma_{\mathcal{P}}$, $n \geq 0$, and for each $(a_1, \ldots, a_n) \in A_{K_1} \times \ldots \times A_{K_n}$ the equation

$$f(a_1, \ldots, a_n) = f_A(a_1, \ldots, a_n)$$

(ii) For each $s \in \mathcal{S}_{\mathcal{P}}$, and each $a \in A_s$ the membership axiom

$$a : s$$

It is then easy to check that the underlying $\mathcal{P}$-algebra of $\mathcal{T}_{\Omega_{\mathcal{P}(A)}, \Delta(A)}$ is isomorphic to A.

In the following, we denote by $\mathcal{E}_{\mathcal{B}}(A)$ the set $\Delta(A) \cup \mathcal{E}_{\mathcal{B}}$. $F_{\mathcal{J}}(A)$ is then the underlying $\mathcal{B}$-algebra of the initial algebra $\mathcal{T}_{\Omega_{\mathcal{B}}(A), \mathcal{E}_{\mathcal{B}}(A)}$. The following lemma is an easy consequence of this construction.

Lemma 70 *For $\mathcal{J} = (\mathcal{P}, \mathcal{B})$ a parameterized specification and Γ a context in $\mathcal{P}$, and $A = \mathcal{T}_{\Omega_{\mathcal{P}}, \mathcal{E}_{\mathcal{P}}}(\Gamma)$ we have $F_{\mathcal{J}}(\mathcal{T}_{\Omega_{\mathcal{P}}, \mathcal{E}_{\mathcal{P}}}(\Gamma)) = \mathcal{T}_{\Omega_{\mathcal{B}}, \mathcal{E}_{\mathcal{B}}}(\Gamma)$.*

We call the functor $F_{\mathcal{J}}$ sending each $\mathcal{P}$-algebra A to its free extension $F_{\mathcal{J}}(A)$ to a $\mathcal{B}$-algebra the *model-theoretic semantics* of the parameterized specification $\mathcal{J} = (\mathcal{P}, \mathcal{B})$.

We now define several useful properties about parameterized specifications.

Definition 71 *Let $\mathcal{J} = (\mathcal{P}, \mathcal{B})$ be a parameterized specification such that for each kind $K \in \mathcal{K}_{\mathcal{P}}$ there is a sort $\top_K \in \mathcal{S}_{\mathcal{P},K}$ such that for any $s \in \mathcal{S}_{\mathcal{P},K}$, $s \leq_{\mathcal{P}} \top_K$. Then, if for each $s \in \mathcal{S}_{\mathcal{P}}$ and $A \in Alg_{\Omega_{\mathcal{P}}, \mathcal{E}_{\mathcal{P}}}$ the function*

$$\eta_{A,s} : A_s \to F_{\mathcal{J}}(A)|_{\mathcal{P},s}$$

is injective, resp. surjective, resp. bijective, then $\mathcal{J}$ is called an extending*, resp.* generating*, resp.* persistent *parameterized specification.*

Lemma 72 *Let $\mathcal{J} = (\mathcal{P}, \mathcal{B})$ be a parameterized specification such that for each kind $K \in \mathcal{K}_{\mathcal{P}}$ there is a sort $\top_K \in \mathcal{S}_{\mathcal{P},K}$ such that for any $s \in \mathcal{S}_{\mathcal{P},K}$, $s \leq_{\mathcal{P}} \top_K$. Then $\mathcal{J}$ is* persistent *iff for each algebra $A \in Alg_{\Omega_{\mathcal{P}}, \mathcal{E}_{\mathcal{P}}}$ we have:*

- *For each T, $T' \in \mathcal{T}_{\Omega_{\mathcal{P}}}(A)_K$ such that $\mathcal{E}_{\mathcal{B}}(A) \vdash T, T' : \top_K$ and $\mathcal{E}_{\mathcal{B}}(A) \vdash T = T'$ we have $\Delta(A) \vdash T = T'$.*
- *For each term $T \in \mathcal{T}_{\Omega_{\mathcal{B}}}(A)$ such that $\mathcal{E}_{\mathcal{B}}(A) \vdash T : s$ for some sort $s \in \mathcal{S}_{\mathcal{P}}$, there exists a constant $a \in A$ such that $\mathcal{E}_{\mathcal{B}}(A) \vdash T = a$ and $\Delta(A) \vdash a : s$.*

Let us analyze in more detail the construction of the free extension $F_{\mathcal{J}}(A)$.

Notice that the unit of the adjunction in the $\Omega_{\mathcal{P}}$-homomorphism

$$A \xrightarrow{\eta_A} F_{\mathcal{J}}(A)|_{\mathcal{P}}$$

maps each $a \in A_K$ to the constant $a \in \Sigma_{\mathcal{B}}(A)_{\lambda,K}$.

Notice that there is also an $\Omega_{\mathcal{P}}$-homomorphism

$$A \xrightarrow{\mu_A} \mathcal{T}_{\Omega_{\mathcal{B}}(A), (\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A)}|_{\mathcal{P}}$$

again, sending $a \in A_K$ to the equivalence class of $a \in \Sigma_{\mathcal{B}}(A)_{\lambda,K}$.

Since in general, $(\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A)$ has fewer equations that $\mathcal{E}_{\mathcal{B}}(A) = \mathcal{E}_{\mathcal{B}} \cup \Delta(A)$, we have a quotient map:

$$q : \mathcal{T}_{\Omega_{\mathcal{B}}(A),(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} \to \mathcal{T}_{\Omega_{\mathcal{B}}(A),\mathcal{E}_{\mathcal{B}}(A)}$$

and therefore we have

$$\begin{tikzcd} & \mathcal{T}_{\Omega_{\mathcal{B}}(A),(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)}|_{\mathcal{P}} \arrow[dd, "q|_{\mathcal{P}}"] \\ A \arrow[ur, "\mu_A"] \arrow[dr, "\eta_A"'] & \\ & \mathcal{T}_{\Omega_{\mathcal{B}}(A),\mathcal{E}_{\mathcal{B}}(A)}|_{\mathcal{P}} \end{tikzcd}$$

In order to further study this diagram we need the notion of strict specifications.

Definition 73 *[75] We call a specification $(\Omega, \mathcal{E})$* strict *iff:*

- *For each kind K the set of sorts $\mathcal{S}_K$ is a poset with top sort $\top_K$ under the sort ordering $\leq_{\mathcal{E}}$ defined by $\mathcal{E}$.*
- *All the Horn clauses in $\mathcal{E}$ have* sorted variables*, that is, are of the form*

$$\varphi = \forall \overline{x} \;\; T = T' \texttt{ if } \overline{U} = \overline{V} \;\wedge\; \overline{x} : \overline{s} \;\wedge\; W : \overline{t}$$

$$\textit{or } \varphi = \forall \overline{x} \;\; T : s \texttt{ if } \overline{U} = \overline{V} \;\wedge\; \overline{x} : \overline{s} \;\wedge\; W : \overline{t}$$

- *$(\Omega, \mathcal{E})$ and $(\Omega, \gamma(\mathcal{E}))$ are equivalent theories (that is, they prove the same sentences) where $\gamma(\mathcal{E})$ is defined as follows. Given a Horn clause $\varphi = A \texttt{ if } \bigwedge_i B_i$ in $\mathcal{E}$, $\gamma(\varphi) = \{A \texttt{ if } \bigwedge_i B_i \wedge \bigwedge_i def(B_i)\} \cup \{D \texttt{ if } \bigwedge_i B_i \wedge \bigwedge_i def(B_i) \mid D \in def(A)\}$, where for each atom A, $def(A) = \{U : \top_{K(U)} \mid U$ nonvariable subterm of $A\}$, where $K(U)$ denotes the kind of term U.*

The intuitive idea of $\gamma(\mathcal{E})$ is that its axioms are just like those in $\mathcal{E}$, except that in addition they require that all subterms in the condition and the conclusion should be sortable, so that the axioms in fact only affect data in the sorts, not in the kinds. Strictness is a property satisfied by many specifications. For example, standard order-sorted specifications in which all terms appearing in the axioms are sortable are strict specifications.

The key property of strict specifications is that their axioms only affect data in the sorts. This can be made precise in terms of the associated membership equational logic partial algebras. Such algebras have been defined in [75].

The idea is as follows. Given a strict specification $(\Omega, \mathcal{E})$, let $Alg_{\Omega,\leq_{\mathcal{E}}}$ denote the class of Ω-algebras satisfying the sort inclusions in the poset of sorts $(\mathcal{S}, \leq_{\mathcal{E}})$ defined by the specification $\mathcal{E}$, that is $\leq_{\mathcal{E}} = \{x : s \Leftarrow x : s' \mid s' \leq s \text{ in } \leq_{\mathcal{E}}\}$. Then, given an algebra $A \in Alg_{\Omega,\leq_{\mathcal{E}}}$, we can throw away the data elements that have a kind, but not a sort, and get a partial algebra A° with the same sort and subsort structure as A, that is, $A^\circ_s = A_s$, and such that each $f : K_1 \times \cdots \times K_n \to K$ in Σ is now interpreted as the partial operation

$$A^\circ_f : A_{\top_{K_1}} \times \cdots \times A_{\top_{K_n}} \to A_{\top_K}$$

obtained by restricting to $A_{\top_{K_1}} \times \cdots \times A_{\top_{K_n}} \times A_{\top_K}$ the graph of the function A_f. The key point about strict specifications can then be stated as follows.

Lemma 74 *Given a strict specification $(\Omega, \mathcal{E})$, and given $A \in Alg_{\Omega,\leq_{\mathcal{E}}}$ as above, we have*

$$A \models \mathcal{E} \Leftrightarrow A^\circ \models \mathcal{E}$$

where satisfaction in A° is defined in the usual way from satisfaction of atomic formulas, with satisfaction of atomic formulas defined by:

$$A^\circ, a \models T = T' \;\; iff \;\; \tilde{a}(T) \; and \; \tilde{a}(T') \; are \; both \; defined \; and \; \tilde{a}(T) = \tilde{a}(T')$$

$$A^\circ, a \models T : s \;\; iff \;\; \tilde{a}(T) \; is \; defined, \; and \; \tilde{a}(T) \in A_s$$

where $\tilde{a}$ is the partial function extending the assignment a to terms inductively.

Proposition 75 *Assume that each connected component in $\mathcal{K}_{\mathcal{P}}$ has a top sort $\top_K$, and assume that the axioms of $\mathcal{E}_{\mathcal{P}}$ "only affect sorts" in the sense that $\mathcal{E}_{\mathcal{P}}$ is a strict specification. Then, if $\mu_{A,s}$ is bijective for each $s \in \mathcal{S}_{\mathcal{P}}$, then $\eta_{A,s}$ is also bijective $\forall s \in \mathcal{S}_{\mathcal{P}}$ and in fact we have that q is an isomorphism.*

Proof: If $\mu_{A,s}$ is bijective for each $s \in \mathcal{S}_{\mathcal{P}}$, then

$$A^\circ \simeq (\mathcal{T}_{\Omega_{\mathcal{B}}(A),(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)}|_{\mathcal{P}})^\circ$$

and, since $\mathcal{E}_{\mathcal{P}}$ is strict we have,

$$A \models \mathcal{E}_{\mathcal{P}} \Leftrightarrow A^\circ \models \mathcal{E}_{\mathcal{P}} \Leftrightarrow (\mathcal{T}_{\Omega_{\mathcal{B}}(A),(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)}|_{\mathcal{P}})^\circ \models \mathcal{E}_{\mathcal{P}} \Leftrightarrow \mathcal{T}_{\Omega_{\mathcal{B}}(A),(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} \models \mathcal{E}_{\mathcal{P}}$$

Therefore, for each $A \in Alg_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}$ such that the $\mu_{A,s}$ are bijective we have $\mathcal{T}_{\Omega_{\mathcal{B}}(A),(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} \models \mathcal{E}_{\mathcal{P}}$ and therefore $\mathcal{T}_{\Omega_{\mathcal{B}}(A),(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} \simeq \mathcal{T}_{\Omega_{\mathcal{B}}(A),\mathcal{E}_{\mathcal{B}}(A)}$, that is, q is an isomorphism, so that $\eta_{A,s}$ is bijective for each $s \in \mathcal{S}_{\mathcal{P}}$. $\square$

From now on, we assume that $\Sigma_{\mathcal{B}} - \Sigma_{\mathcal{P}}$ is partitioned into two disjoint sets $\Sigma_{\mathcal{D}} \cup \Sigma_{\mathcal{C}}$, with $\Sigma_{\mathcal{C}}$ the set of constructor symbols, and $\Sigma_{\mathcal{D}}$ the set of defined symbols. We also assume that $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}} = \mathcal{E}_{\mathcal{D}} \cup \mathcal{E}_{\mathcal{C}}$ where $\mathcal{E}_{\mathcal{C}}$ are Horn clauses over $\Sigma_{\mathcal{C}}$ and all lefthand sides of rules in $\mathcal{E}_{\mathcal{D}}$ are of the form $f(\overline{L})$ where $f \in \Sigma_{\mathcal{D}}$.

Definition 76 *Consider specifications $\mathcal{P} \subseteq \mathcal{C} \subseteq \mathcal{B}$ such that $\mathcal{K}_{\mathcal{C}} = \mathcal{K}_{\mathcal{B}}$, $\mathcal{S}_{\mathcal{C}} = \mathcal{S}_{\mathcal{B}}$, and for each $K \in \mathcal{K}_{\mathcal{C}}$ and $s, s' \in \mathcal{S}_{\mathcal{C}}$ we have $s \leq_{\mathcal{C}} s'$ iff $s \leq_{\mathcal{B}} s'$, and with a top sort $\top_K$ for each $(\mathcal{S}_{\mathcal{C}})_K$ in both (identical) orderings. Then, we say that $\mathcal{J}_{\mathcal{C}} = (\mathcal{P}, \mathcal{C})$ is a generating (resp. persistent) parameterized constructor subspecification of $\mathcal{J} = (\mathcal{P}, \mathcal{C})$ iff for each $s \in \mathcal{S}_{\mathcal{C}}$ and each $A \in Alg_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}$ the function*

$$\delta_s : F_{\mathcal{J}_{\mathcal{C}}}(A)_s \to F_{\mathcal{J}}(A)_s$$

is surjective (resp. bijective), where $\delta : F_{\mathcal{J}_{\mathcal{C}}}(A) \to F_{\mathcal{J}}(A)$ is the instance for $C = F_{\mathcal{J}_{\mathcal{C}}}(A)$ of the natural map $\delta_C : C \to F_{\mathcal{L}}(C)_{\mathcal{C}}$, $C \in Alg_{\Omega_{\mathcal{C}},\mathcal{E}_{\mathcal{C}}}$ associated to the parameterized module $\mathcal{L} = (\mathcal{C}, \mathcal{B})$.

Lemma 77 *Consider specifications $\mathcal{P} \subseteq \mathcal{C} \subseteq \mathcal{B}$ such that $\mathcal{K}_{\mathcal{C}} = \mathcal{K}_{\mathcal{B}}$, $\mathcal{S}_{\mathcal{C}} = \mathcal{S}_{\mathcal{B}}$, and for each $K \in \mathcal{K}_{\mathcal{C}}$ and $s, s' \in \mathcal{S}_{\mathcal{C}}$ we have $s \leq_{\mathcal{C}} s'$ iff $s \leq_{\mathcal{B}} s'$, and with a top sort $\top_K$ for each $(\mathcal{S}_{\mathcal{C}})_K$ in both (identical) orderings. Then $\mathcal{J}_{\mathcal{C}}$ is a persistent parameterized constructor subspecification of $\mathcal{J} = (\mathcal{P}, \mathcal{C})$ iff for each algebra $A \in Alg_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}$,*

- *For each $T,\ T' \in \mathcal{T}_{\mathcal{C}(A)_K}$ such that $\mathcal{E}_{\mathcal{B}}(A) \vdash T, T' : \top_K$ and $\mathcal{E}_{\mathcal{B}}(A) \vdash T = T'$ we have $\mathcal{E}_{\mathcal{C}}(A) \vdash T = T'$.*

- *For each term $T \in \mathcal{T}_{\mathcal{B}(A)_K}$ such that $\mathcal{E}_{\mathcal{B}}(A) \vdash T : s$ for some sort s, there exists $T' \in \mathcal{T}_{\mathcal{C}(A)_K}$ such that $\mathcal{E}_{\mathcal{B}}(A) \vdash T = T'$ and $\mathcal{E}_{\mathcal{C}}(A) \vdash T' : s$.*

where $\mathcal{E}_{\mathcal{C}}(A) = \Delta(A) \cup \mathcal{E}_{\mathcal{C}}$.

Note that we can decompose proving that a parameterized module is generating (resp. persistent) into proving the same result for a generating (resp. persistent) constructor subspecification.

Lemma 78 *Consider specifications $\mathcal{P} \subseteq \mathcal{C} \subseteq \mathcal{B}$ such that $\mathcal{K}_{\mathcal{C}} = \mathcal{K}_{\mathcal{B}}$, $\mathcal{S}_{\mathcal{C}} = \mathcal{S}_{\mathcal{B}}$, and for each $K \in \mathcal{K}_{\mathcal{C}}$ and $s, s' \in \mathcal{S}_{\mathcal{C}}$ we have $s \leq_{\mathcal{C}} s'$ iff $s \leq_{\mathcal{B}} s'$, and with a top sort $\top_K$ for each $(\mathcal{S}_{\mathcal{C}})_K$ in both (identical) orderings. Assume also that for each $K' \in \mathcal{K}_{\mathcal{P}}$ there is a top sort $\top'_{K'}$ with $\top'_{K'} \geq_{\mathcal{P}} s'$ for each $s' \in \mathcal{S}_{\mathcal{P},K'}$. Then if $\mathcal{J}_{\mathcal{C}} = (\mathcal{P}, \mathcal{C})$ is a generating (resp. persistent) parameterized specification, and $\mathcal{J}_{\mathcal{C}}$ is a generating (resp. persistent) parameterized constructor subspecification of $\mathcal{J} = (\mathcal{P}, \mathcal{B})$, then $\mathcal{J}$ is a generating (resp. persistent) parameterized specification.*

Now we will introduce sufficient conditions to guarantee that a parameter constructor specification is persistent.

Let us first introduce the following notions. To distinguish between the rewrite steps that result from the systems $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ and those resulting from $\Delta(A)$ we write $\xrightarrow{\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}}}_{(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)}$ (resp. $\xrightarrow{\Delta(A)}_{(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)}$) to indicate that the rule inducing the rewrite step is an element from $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ (resp. $\Delta(A)$). To each non-left-linear rule $L \to R \ \texttt{if}\ P$, we can associate its *linearized* version $L' \to R' \ \texttt{if}\ P' \wedge P''$, such that L' is linear, $L = L'\sigma$ for some renaming σ, $R = R'\sigma$, $P = P'\sigma$, and $x = y \in P''$ for each pair (x, y) of variables of $\mathcal{V}ar(L')$ such that $x\sigma = y\sigma$. This shows that left-linearity is not a real restriction when dealing with conditional rules.

The following lemma is fundamental in order to prove that the rewrite relation $\to_{(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)}$ is terminating for every algebra A in $Alg_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}$.

Lemma 79 *Assume that $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ is left-linear [2] and reductive and let $T \in \mathcal{T}_{\mathcal{B}(A)}$. If there exists T' and T'' such that*

$$T \xrightarrow{\Delta(A)}_{(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} T' \xrightarrow{\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}}}_{(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} T''$$

then there exists T''' such that

$$T \xrightarrow{\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}}}_{(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} T''' \xrightarrow{\Delta(A)}{}^{*}_{(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} T''$$

As a consequence of the interchangeability lemma we get the well-foundedness of $\to_{(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)}$.

Theorem 80 *If $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ is reductive, then the rewrite relation $\to_{(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)}$ is terminating for every algebra A in $Alg_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}$.*

The following proposition presents sufficient conditions to guarantee that $(\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A)$ is ground confluent and ground sort decreasing for every algebra A in $Alg_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}$.

[2] If $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ is not left-linear, then we can consider the linearized version of $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$.

Proposition 81 *If $\mathcal{E}_\mathcal{B}-\mathcal{E}_\mathcal{P}$ is confluent and sort-decreasing and no symbol from $\Sigma_\mathcal{P}$ occurs on the lefthand side of any rewrite/membership rule from $\mathcal{E}_\mathcal{B}-\mathcal{E}_\mathcal{P}$, then $(\mathcal{E}_\mathcal{B}-\mathcal{E}_\mathcal{P})\cup\Delta(A)$ is ground confluent and ground sort decreasing for every algebra A in $Alg_{\Omega_\mathcal{P},\mathcal{E}_\mathcal{P}}$.*

Proof: Since no symbol from $\Sigma_\mathcal{P}$ occurs on the lefthand side of any rule from $\mathcal{E}_\mathcal{B}-\mathcal{E}_\mathcal{P}$, then no critical pair nor critical membership overlap between a rewrite/membership rule from $\mathcal{E}_\mathcal{B}-\mathcal{E}_\mathcal{P}$ and a rewrite/membership rule from $\Delta(A)$ can exist for any algebra A in $Alg_{\Omega_\mathcal{P},\mathcal{E}_\mathcal{P}}$. So no new critical pairs nor critical memberships can be derived and therefore we can easily show that $(\mathcal{E}_\mathcal{B}-\mathcal{E}_\mathcal{P})\cup\Delta(A)$ is ground confluent and ground sort decreasing for every algebra A in $Alg_{\Omega_\mathcal{P},\mathcal{E}_\mathcal{P}}$. □

Theorem 82 (Persistence of Parameterized Constructor Specifications) *Let $\mathcal{J}_\mathcal{C}=(\mathcal{P},\mathcal{C})$ be a parameterized specification in which $\mathcal{P}$ is strict, for each kind K in $\mathcal{K}_\mathcal{C}$ there is a top sort $\top_K$ in $\leq_\mathcal{C}$, $\leq_\mathcal{C}\cap\,\mathcal{S}_\mathcal{P}^2=\leq_\mathcal{P}$, and $\forall s\in\mathcal{S}_\mathcal{C}\ \forall s'\in\mathcal{S}_\mathcal{P}:\ s\leq_\mathcal{C} s'\Rightarrow s\in\mathcal{S}_\mathcal{P}$ (that is, all sorts in $\mathcal{S}_\mathcal{C}$ are above sorts in $\mathcal{S}_\mathcal{P}$ or are incomparable with them). Assume further that for each operator $c\in\Sigma_\mathcal{C}-\Sigma_\mathcal{P}$ all the memberships are of the form*

$$c(\overline{T}) : s \ \texttt{if}\ \overline{U}=\overline{V}\ \wedge\ \overline{W}:\overline{s}$$

with $s\in\mathcal{S}_\mathcal{C}-\mathcal{S}_\mathcal{P}$, all memberships in $\mathcal{E}_\mathcal{C}-\mathcal{E}_\mathcal{P}$ are of this form, and that all equations in $\mathcal{E}_\mathcal{C}-\mathcal{E}_\mathcal{P}$:

(i) have all their symbols in $\Sigma_\mathcal{C}-\Sigma_\mathcal{P}$.
(ii) are confluent and reductive

then $\mathcal{J}_\mathcal{C}$ is a persistent parameterized constructor specification.

Proof: To prove that $\mathcal{J}_\mathcal{C}$ is a persistent parameterized constructor specification, we need to prove that for each $A\in Alg_{\Omega_\mathcal{P},\mathcal{E}_\mathcal{P}}$ the map $\eta_{A_s}: A_s\to F_{\mathcal{J}_\mathcal{C}}(A)_s$ is bijective for each $s\in\mathcal{S}_\mathcal{P}$. Since $\mathcal{P}$ is strict, by Proposition 75 it is enough to show that $\mu_{A,s}: A_s\to \mathcal{T}_{\Omega_\mathcal{C}(A),(\mathcal{E}_\mathcal{C}-\mathcal{E}_\mathcal{P})\cup\Delta(A),s}$ is bijective for each $s\in\mathcal{S}_\mathcal{P}$.

(i) **injectivity:** Let $T,T'\in\mathcal{T}_{\Omega_\mathcal{P}(A)}$ be such that $(\mathcal{E}_\mathcal{C}-\mathcal{E}_\mathcal{P})\cup\Delta(A)\vdash T,\ T':s$ for some sort $s\in\mathcal{S}_\mathcal{P}$ and $(\mathcal{E}_\mathcal{C}-\mathcal{E}_\mathcal{P})\cup\Delta(A)\vdash T=T'$. By following the same reasoning used in the proofs of Proposition 81 and Theorem 80, we can easily conclude that $(\mathcal{E}_\mathcal{C}-\mathcal{E}_\mathcal{P})\cup\Delta(A)$ is ground confluent and the rewrite relation $\to_{(\mathcal{E}_\mathcal{C}-\mathcal{E}_\mathcal{P})\cup\Delta(A)}$ is terminating for every algebra $A\in Alg_{\Omega_\mathcal{P},\mathcal{E}_\mathcal{P}}$, then there exists T'' such that $T\to^*_{(\mathcal{E}_\mathcal{C}-\mathcal{E}_\mathcal{P})\cup\Delta(A)} T''$ and $T'\to^*_{(\mathcal{E}_\mathcal{C}-\mathcal{E}_\mathcal{P})\cup\Delta(A)} T''$. No rule from $\mathcal{E}_\mathcal{C}-\mathcal{E}_\mathcal{P}$ can be used in the rewriting steps since no symbol from $\Sigma_\mathcal{P}$ occurs on the lefthand side of any rule from $\mathcal{E}_\mathcal{C}-\mathcal{E}_\mathcal{P}$. So, we have $T\to^*_{\Delta(A)} T''$ and $T'\to^*_{\Delta(A)} T''$, this implies that $\Delta(A)\vdash T=T'$.

(ii) **surjectivity:** Let $T\in\mathcal{T}_{\Omega_\mathcal{C}(A)}$ be such that $(\mathcal{E}_\mathcal{C}-\mathcal{E}_\mathcal{P})\cup\Delta(A)\vdash T:s$ for some sort $s\in\mathcal{S}_\mathcal{P}$. By assumption on the memberships and equations of $\mathcal{E}_\mathcal{C}-\mathcal{E}_\mathcal{P}$, no terms with symbols in $\Sigma_\mathcal{C}-\Sigma_\mathcal{P}$ can have a sort $s\in\mathcal{S}_\mathcal{P}$. So, we have $T\in\mathcal{T}_{\Omega_\mathcal{P}(A)}$ and $\Delta(A)\vdash T:s$ and therefore there exists a constant $a\in A$ such that $\Delta(A)\vdash T=a$ and $\Delta(A)\vdash a:s$ by the sort-decreasingness of $\Delta(A)$. □

Completeness of Defined Symbols

Guttag showed that sufficient completeness is undecidable for many-sorted equational specifications [46], but becomes decidable when the specification is given in the form of a confluent (non-conditional) rewrite system [54]. Although this property is in general undecidable, some syntactic criteria can be given. Most of them consider (non-conditional) rewrite systems and they are based on rewriting methods [45, 48, 26, 62, 19]. In this section, we give an effective method for testing this property in the framework of membership equational logic. This method is inspired by [8] and it is based on the notion of *Pattern* trees.

Let us first define our notion of completeness of parameterized specifications. From now on, we assume that the axioms in $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ are partitioned into two sets of axioms: $\mathcal{E}_{\mathcal{C}}$ involving only constructors and the axioms $\mathcal{E}_{\mathcal{D}}$ involving defined symbols, we obtain a subspecification $\mathcal{C} \subseteq \mathcal{B}$ with signature $\Sigma_{\mathcal{P}} \cup \Sigma_{\mathcal{C}}$, and with axioms $\mathcal{E}_{\mathcal{P}} \cup \mathcal{E}_{\mathcal{C}}$.

Definition 83 *Let $\mathcal{J} = (\mathcal{P}, \mathcal{B})$ be a* parameterized specification*, and assume that $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ is a reductive CRMS. We furthermore assume that $\Sigma_{\mathcal{B}} - \Sigma_{\mathcal{P}}$ is partitioned into two disjoint sets $\Sigma_{\mathcal{D}} \cup \Sigma_{\mathcal{C}}$, with $\Sigma_{\mathcal{C}}$ the set of constructor symbols, and $\Sigma_{\mathcal{D}}$ the set of defined symbols. We say the $\mathcal{J}$ is* complete *iff each defined symbol $f \in \Sigma_{\mathcal{B}} - \Sigma_{\mathcal{C}}$ is completely defined.*

A function symbol $f \in \Sigma_{\mathcal{B}} - \Sigma_{\mathcal{C}}$ is completely defined *iff for each $\mathcal{P}$-algebra A and each term T of the form $f(T_1, \ldots, T_n)$ where for all $1 \leq i \leq n$, $T_i \in \mathcal{T}_{\Omega_{\mathcal{C}}(A)}$ and $(\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A) \vdash T : s$ for some sort s, there exists $T' \in \mathcal{T}_{\Omega_{\mathcal{C}}(A)}$ such that $f(T_1, \ldots, T_n) \rightarrow^{+}_{(\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A)} T'$ and $(\mathcal{E}_{\mathcal{C}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A) \vdash T' : s$.*

Completeness is then a sufficient condition to ensure that $\mathcal{J}_{\mathcal{C}} = (\mathcal{P}, \mathcal{C})$ is a *generating* constructor subspecification of $\mathcal{J} = (\mathcal{P}, \mathcal{B})$.

The following Theorem gives sufficient conditions to guarantee that a parameterized specification is persistent.

Theorem 84 *Let $\mathcal{P} \overset{\mathcal{J}_{\mathcal{C}}}{\hookrightarrow} \mathcal{C} \hookrightarrow \mathcal{B}$ and $\mathcal{P} \overset{\mathcal{J}}{\hookrightarrow} \mathcal{B}$ be a couple of parameterized specifications such that:*

1. *$\mathcal{J}_{\mathcal{C}}$ satisfies the conditions in Theorem 82*
2. *$(\mathcal{S}_{\mathcal{B}}, \leq_{\mathcal{B}}) = (\mathcal{S}_{\mathcal{C}}, \leq_{\mathcal{C}})$ (same sort structure for $\mathcal{C}$ and $\mathcal{B}$)*
3. *$\mathcal{J}$ is* complete
4. *Denoting by $\Sigma_{\mathcal{D}} = \Sigma_{\mathcal{B}} - \Sigma_{\mathcal{C}}$ the set of defined symbols, we have:*
   - *(i) All equations in $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{C}}$ are such that their lefthand sides have a top symbol in $\Sigma_{\mathcal{D}}$ and no symbol in $\Sigma_{\mathcal{P}}$.*
   - *(ii) The equations in $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ are confluent, sort-decreasing and reductive.*
   - *(iii) Each membership rule in $\mathcal{E}_{\mathcal{B}}$ is of the form $f(\overline{L}) : s$ `if` $\mathcal{V}ar(\overline{L}) : \overline{s} \wedge \overline{U} : \overline{t} \wedge \overline{V} \downarrow \overline{W}$ where $\mathcal{V}ar(\overline{U}, \overline{V}, \overline{W}) \subseteq \mathcal{V}ar(\overline{L})$ and for each $M \downarrow N \in \overline{V} \downarrow \overline{W}$, there exists some sort $t \in \mathcal{S}_{\mathcal{B}}$ such that $\mathcal{V}ar(\overline{L}) : \overline{s} \vdash_{\mathcal{E}_{\mathcal{B}}} M : t$ and $\mathcal{V}ar(\overline{L}) : \overline{s} \vdash_{\mathcal{E}_{\mathcal{B}}} N : t$.*

Then $\mathcal{J}_{\mathcal{C}} \subset \mathcal{J}$ is a persistent constructor subspecification and $\mathcal{J}$ is persistent.

Proof: Note that, if we show that $\mathcal{J}_{\mathcal{C}} \subset \mathcal{J}$ is a persistent constructor subspecification, then using Lemma 78 and Theorem 82, we automatically get that $\mathcal{J}$ is persistent. Indeed, by Theorem 82, we have for each $A \in Alg_{\Omega_{\mathcal{P}}, \mathcal{E}_{\mathcal{P}}}$ that $\eta_{A_s} : A_s \rightarrow F_{\mathcal{J}_{\mathcal{C}}}(A)_s$ is bijective for each $s \in \mathcal{S}_{\mathcal{P}}$. Showing that $\mathcal{J}_{\mathcal{C}} \subset \mathcal{J}$ is a persistent constructor subspecification, by definition, is equivalent to showing that for each $s \in \mathcal{S}_{\mathcal{C}}$ and for each such $A \in Alg_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}$, the function

$$\delta_s : F_{\mathcal{J}_{\mathcal{C}}}(A)_s \to F_{\mathcal{J}}(A)_s$$

associated to the natural map $\delta : F_{\mathcal{J}_{\mathcal{C}}}(A) \to F_{\mathcal{J}}(A)$ is bijective. But since, denoting by $\eta' : A \to F_{\mathcal{J}}(A)$ the unit map, we have the equation $\eta' = (\delta|_{\mathcal{P}}) \circ \eta$. So, if we show that the δ_s are bijective, then we will have that the η'_s are bijective too and therefore $\mathcal{J}$ is persistent. Therefore, all we need to show is that the maps δ_s above are bijective.

To show that δ_s is bijective for each $s \in \mathcal{S}_{\mathcal{C}}$, it is enough to show that the natural map

$$\rho_s : \mathcal{T}_{\Omega_{\mathcal{C}}(A),(\mathcal{E}_{\mathcal{C}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A),s} \to \mathcal{T}_{\Omega_{\mathcal{B}}(A),(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A),s}$$

is bijective for each $s \in \mathcal{S}_{\mathcal{C}}$. Indeed,

(1) By the proof of Theorem 82, we know that the quotient map

$$q : \mathcal{T}_{\Omega_{\mathcal{C}}(A),(\mathcal{E}_{\mathcal{C}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} \to F_{\mathcal{J}_{\mathcal{C}}}(A)$$

is an isomorphism.

(2) We have a commutative diagram of natural maps

$$\begin{tikzcd}
\mathcal{T}_{\Omega_{\mathcal{C}}(A),(\mathcal{E}_{\mathcal{C}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} \arrow[r, "\rho"] \arrow[d, "q"'] & \mathcal{T}_{\Omega_{\mathcal{B}}(A),(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} \arrow[d, "q'"] \\
F_{\mathcal{J}_{\mathcal{C}}}(A) \arrow[r, "\delta"] & F_{\mathcal{J}}(A)
\end{tikzcd}$$

(3) We can use Proposition 75, again, but now applied to $\mathcal{J}$, because the map $\mu'_A : A \to \mathcal{T}_{\Omega_{\mathcal{B}}(A),(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)}$ factors as the map $\mu'_A = \rho \circ \mu_A$ and μ_A is bijective by the proof of Theorem 82, so if ρ_s is bijective, $\mu'_{A,s}$ is bijective, and by Proposition 75, then q' is an isomorphism.

Therefore, we have reduced showing δ_s bijective for each $s \in \mathcal{S}_{\mathcal{C}}$ to showing ρ_s bijective.

(i) **injectivity:** Let $T, T' \in \mathcal{T}_{\Omega_{\mathcal{C}}(A)_K}$ be such that $(\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A) \vdash T,\ T' : \top_K$ and $(\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A) \vdash T = T'$. By Proposition 81 and Theorem 80, we conclude that $(\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A)$ is ground confluent and the rewrite relation $\to_{(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)}$ is terminating for every algebra $A \in Alg_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}$, then there exists T'' such that $T \to^*_{(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} T''$ and $T' \to^*_{(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} T''$. The only rules in $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ that can rewrite T and T' are those in $\mathcal{E}_{\mathcal{C}} - \mathcal{E}_{\mathcal{P}}$, since all equations in $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{C}}$ are such that their lefthand sides have a top symbol in $\Sigma_{\mathcal{D}}$. So, we have $T \to^*_{(\mathcal{E}_{\mathcal{C}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} T''$ and $T' \to^*_{(\mathcal{E}_{\mathcal{C}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)} T''$, this implies that $(\mathcal{E}_{\mathcal{C}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A) \vdash T = T'$.

(ii) **surjectivity:** Let $T \in \mathcal{T}_{\Omega_{\mathcal{B}}(A)_K}$ be such that $(\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A) \vdash T : s$ for some sort s. By Proposition 81 and Theorem 80, we conclude that $(\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A)$ is ground confluent and ground sort decreasing and the rewrite relation $\to_{(\mathcal{E}_{\mathcal{B}}-\mathcal{E}_{\mathcal{P}})\cup\Delta(A)}$ is terminating for every algebra $A \in Alg_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}$. Let T' be the normal form of T with respect to $(\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A)$. If T' is a constructor term, using the ground sort-decreasingness of $(\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A)$ we conclude that $(\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A) \vdash T' : s$.

Now, since T' is a constructor term and no symbol from $\Sigma_{\mathcal{D}}$ occurs on the axioms from $\mathcal{E}_{\mathcal{C}}$, we have $(\mathcal{E}_{\mathcal{C}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A) \vdash T' : s$.

Otherwise, T' must contain a subterm of the form $g(\overline{T})$ where $g \in \Sigma_{\mathcal{D}}$ and for all $i \in [1..n]$, $T_i \in \mathcal{T}_{\Omega_{\mathcal{C}}(A)_K}$. Using assumption 4.(iii), we can show that $(\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}) \cup \Delta(A) \vdash g(\overline{T}) : s'$ for some sort s'. This contradicts the fact that $\mathcal{J}$ is complete. □

Checking Completeness

From now on we assume that all memberships for defined functions are of the form:

$$f(\overline{x}) : s \texttt{ if } \overline{x} : \overline{s}$$

In order to check that a specification $\mathcal{J} = (\mathcal{P}, \mathcal{B})$ is complete, we use the following notions:

Definition 85 *A* pattern *is an order-sorted term* $(f(\overline{T}), \{\mathcal{V}ar(\overline{T}) : \overline{s}\})$ *such that* $f \in \Sigma_{\mathcal{D}}$ *and for each* $T_i \in \overline{T}$, $T_i \in \mathcal{T}_{\Sigma_{\mathcal{C}}}(\mathcal{X})$.

Definition 86 *Given a parameterized specification* $\mathcal{J} = (\mathcal{P}, \mathcal{B})$. *A term* (T, Γ) *is said to be* ground reducible *(resp.* ground irreducible, ground sortable*) if* $T\gamma$ *is reducible (resp. irreducible, sortable) for each* $\mathcal{P}$*-algebra* A *and each irreducible admissible ground substitution* γ *over* $F_{\mathcal{J}}(A)$. *We also say that* T *is ground reducible (resp. irreducible, sortable) in the environment* Γ.

Definition 87 *Given a parameterized specification* $\mathcal{J} = (\mathcal{P}, \mathcal{B})$. *A non-parameter sort* s *is* free *if for each* $\mathcal{P}$*-algebra* A*, every ground term inhabiting* $F_{\mathcal{J}_{\mathcal{C}}}(A)_s$ *is irreducible. A Cartesian product of sorts is free if so are its components.*

Given a sort s*, a set* $\mathcal{S}$ *of free subsorts of* s *is a* cover sort *of* s *if for each* $\mathcal{P}$*-algebra* A*, every irreducible ground term* T *inhabiting* $F_{\mathcal{J}_{\mathcal{C}}}(A)_s$ *inhabits a unique sort in* $\mathcal{S}$.

Definition 88 *A finite set* $\mathcal{T}$ *of order-sorted terms inhabiting a free sort* s *is a* cover set *of* s *iff for each* $\mathcal{P}$*-algebra* A*, every ground term inhabiting* $F_{\mathcal{J}_{\mathcal{C}}}(A)_s$ *is an instance of a term in* $\mathcal{T}$.

Example 89 (Example 69 Cont.) *A cover set of the sort* List *contains* $\{(nil, \emptyset);\ (cons(x, y),\ \{x : Elem,\ y : List\})\}$.

If $\mathcal{J}$ is a complete parameterized specification over free constructors, then each non-parameter sort is free, each constructor term is ground irreducible and each sortable non-constructor term is ground reducible.

The following definition characterizes induction variables:

Definition 90 *Given a parameterized specification* $\mathcal{J} = (\mathcal{P}, \mathcal{B})$*, the set* $\mathcal{I}nd\mathcal{P}os(f, \mathcal{J})$*, of* induction positions *of* $f \in \Sigma_{\mathcal{B}}$ *is the set of positions* p *such that there exists in* $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ *a rewrite rule or a membership rule of left-hand side* $f(\overline{L})$*, and* p *is the position in* $f(\overline{L})$ *of a non-variable subterm of a non-parameter sort.*

The set of induction variables *of a pattern* (T, Γ)*, written* $\mathcal{I}nd\mathcal{V}ar(T, \mathcal{J})$*, is the subset of* $\mathcal{V}ar(T)$ *whose elements inhabit a free sort* s *and occur in a subterm of* T *of the form* $f(\overline{S})$*, such that* $S_i \in \mathcal{T}_{\Sigma_{\mathcal{C}}}(\mathcal{X})$ *for each* $S_i \in \overline{S}$*, at an induction position of* f *w.r.t. Definition 90. The notion of induction variables is extended to clauses as expected.*

Example 91 (Example 69 Cont.)

- $\mathcal{I}nd\mathcal{P}os(insert, \mathcal{J}) = \{2\}$
- $\mathcal{I}nd\mathcal{P}os(sorted, \mathcal{J}) = \{1,\ 1.2\}$
- $\mathcal{I}nd\mathcal{P}os(last, \mathcal{J}) = \{1,\ 1.2\}$
- $\mathcal{I}nd\mathcal{V}ar((insert(x,y),\ \{x : Elem,\ y : List\}), \mathcal{J}) = \{y\}$
- $\mathcal{I}nd\mathcal{V}ar((insert(x, cons(y,z)),\ \{x, y : Elem,\ z : List\}), \mathcal{J}) = \emptyset$

The construction of the pattern tree is based on the notion of *strong ground reducibility.*

Definition 92 *A term* $(T, \{\mathcal{V}ar(T) : s\})$ *is* strongly ground reducible *if*
(i) T *is reducible in the environment* $\{\mathcal{V}ar(T) : s\}$*, or*
(ii) the formula $P_1\sigma_1 \vee \ldots \vee P_n\sigma_n$ *is an inductive theorem of* $\mathcal{J}$*, where* $\{L_i \to R_i \texttt{ if } P_i\}_{i\in[1..n]}$ *is the set of linearized rules in* $\mathcal{E_B} - \mathcal{E_P}$ *whose lefthand sides match a subterm of* T *with respective substitutions* $\sigma_1, \ldots, \sigma_n$*.*

Case (ii) of strong ground reducibility is undecidable, while case (i) is a particular decidable case. On the other hand, case (ii) can be checked (and hopefully solved) by using an inductive theorem prover as the one described in Section 9.1.2.

From now on, we assume that $\mathcal{E_D}$ is partitioned into two disjoint sets $\mathcal{M_D}$, the set of membership rules over $\Sigma_\mathcal{D}$ and $\mathcal{R_D}$, the set of rewrite rules over $\Sigma_\mathcal{D}$. Our algorithm for completeness is presented in Figure 9.4 as a set of inference rules operating on quadruples $(\mathcal{P}, \mathcal{RL}, \mathcal{IL}, \mathcal{CP})$, where $\mathcal{P}$ is a set of patterns labeling the leaves of the tree constructed so far, $\mathcal{RL}$ is the set of strongly ground reducible leaves of the result tree, $\mathcal{IL}$ is its set of strongly ground irreducible leaves, and $\mathcal{CP}$ is the set of patterns for which we still must compute their pattern trees.

Success applies when the sets $\mathcal{P}$, $\mathcal{IL}$ and $\mathcal{CP}$ are empty. We can then conclude that the specification of defined symbols is complete. **Missing Patterns** applies when the sets $\mathcal{P}$ and $\mathcal{CP}$ are empty, but $\mathcal{IL}$ is not. In this case, the user is prompted to complete the specification of f at the patterns in $\mathcal{IL}$. When a leaf (T, Γ) is found such that T is strongly ground reducible in Γ, then the **Reducible Leaf** rule adds it to the current set $\mathcal{RL}$. **Decompose Sort** applies when a pattern $(T, \{x : s\} \cup \Gamma)$ has a variable ranging over a non-free sort. **Decompose Variable** instantiates a variable x ranging over a free sort s by terms in a cover set of s. **Irreducible Leaf** applies when a leaf (T, Γ) is found such that no other rule applies.

Lemma 93 *Let* $\mathcal{J}$ *be a parameterized specification which satisfies the conditions in Theorem 84 except condition 3 (*$\mathcal{J}$ *is complete). Assume further that:*

$$(*)\quad \mathcal{J}' \models_{\mathcal{I}nd} \mathcal{M}_s$$

where $\mathcal{M}_s$ *is the set of membership rules in* $\mathcal{E_B} - \mathcal{E_P}$ *of the form* $f(\overline{x}) : s \texttt{ if } \overline{x} : \overline{s}$ *where* $s \in \mathcal{S_P}$*, and* $\mathcal{J}'$ *is obtained from* $\mathcal{J}$ *by removing the membership rules in* $\mathcal{M}_s$ *from* $\mathcal{E_B} - \mathcal{E_P}$*. Then, we have:*

i. $\mathcal{E_B}(A) \vdash T : s$ *for some sort* $s \in \mathcal{S_P}$ *iff* $(\mathcal{E_B} - \mathcal{E_P}) \cup \Delta(A) \vdash T : s$*.*
ii. $\mathcal{E_B}(A) \vdash T, T' : s$ *for some sort* $s \in \mathcal{S_P}$ *and* $\mathcal{E_B}(A) \vdash T = T'$ *iff* $(\mathcal{E_B} - \mathcal{E_P}) \cup \Delta(A) \vdash T = T'$*.*

Init: $\overline{(\emptyset, \emptyset, \emptyset, \mathcal{CP}_{\mathcal{I}})}$

where $\left\{ \begin{array}{l} \mathcal{CP}_{\mathcal{I}} = \{(f(\overline{x}), \{\overline{x} : s\}) \mid \\ f \in \Sigma_{\mathcal{D}} \text{ and } f(\overline{x}) : s \texttt{ if } \overline{x} : \overline{s} \in \mathcal{M}_{\mathcal{D}}\} \end{array} \right.$

Success: $\dfrac{(\emptyset,\ \mathcal{RL},\ \emptyset,\ \emptyset)}{\mathit{success}}$

Check New Function: $\dfrac{(\emptyset,\ \mathcal{RL},\ \mathcal{IL},\ \mathcal{CP} \cup \{(T, \Gamma)\})}{(\{(T, \Gamma)\},\ \mathcal{RL},\ \mathcal{IL},\ \mathcal{CP})}$

Missing Patterns: $\dfrac{(\emptyset,\ \mathcal{RL},\ \mathcal{IL},\ \emptyset)}{\mathcal{IL}}$

if $\mathcal{IL} \neq \emptyset$

Reducible Leaf: $\dfrac{(\mathcal{P} \cup \{(T, \Gamma)\},\ \mathcal{RL},\ \mathcal{IL},\ \mathcal{CP})}{(\mathcal{P},\ \mathcal{RL} \cup \{(T, \Gamma)\},\ \mathcal{IL},\ \mathcal{CP})}$

if T is strongly ground reducible in Γ

Decompose Sort: $\dfrac{(\mathcal{P} \cup \{(T, \{x : s\} \cup \Gamma)\},\ \mathcal{RL},\ \mathcal{IL},\ \mathcal{CP})}{(\mathcal{P} \cup \bigcup_{i \in I} \{(T, \{x : s_i\} \cup \Gamma)\},\ \mathcal{RL},\ \mathcal{IL},\ \mathcal{CP})}$

if $\left\{ \begin{array}{l} T \text{ is not strongly ground reducible in } \Gamma \\ s \text{ is non-free} \end{array} \right.$

where $\{s_i\}_{i \in I}$ is a cover sort of s

Decompose Variable: $\dfrac{(\mathcal{P} \cup \{(f(\overline{T})[x]_p, \{x : s\} \cup \Gamma)\},\ \mathcal{RL},\ \mathcal{IL},\ \mathcal{CP})}{(\mathcal{P} \cup \{(f(\overline{T})[S_i]_p, \Gamma_i \cup \Gamma(x \mapsto S_i))\}_{i \in I},\ \mathcal{RL},\ \mathcal{IL},\ \mathcal{CP})}$

if $\left\{ \begin{array}{l} T \text{ is not strongly ground reducible in } \Gamma \\ s \text{ is free} \\ x \text{ is an induction variable} \end{array} \right.$

where $\{S_i, \Gamma_i\}_{i \in I}$ is a cover set of s using fresh variables

Irreducible Leaf: $\dfrac{(\mathcal{P} \cup \{(T, \Gamma)\},\ \mathcal{RL},\ \mathcal{IL},\ \mathcal{CP})}{(\mathcal{P},\ \mathcal{RL},\ \mathcal{IL} \cup \{(T, \Gamma)\},\ \mathcal{CP})}$

if no other rule applies to the term (T, Γ)

Figure 9.4: Inference System $\mathcal{SC}$ for checking completeness of defined symbols

Proof: The proof of this Lemma uses the same arguments of the proof of Theorem 84.

Let us show that assumption $(*)$ guarantees the surjectivity of $\delta_s : F_{\mathcal{J}_\mathcal{C}}(A)_s \to F_\mathcal{J}(A)_s$ for each $s \in \mathcal{S}_\mathcal{P}$. Note first that $\mathcal{J}' \models_{\mathcal{I}nd} \mathcal{M}_s$ implies that all the membership rules in $\mathcal{M}_s$ are not really useful for computing with ground terms over $\mathcal{T}_{\mathcal{B}(A)_K}$.

Let $T \in \mathcal{T}_{\mathcal{B}(A)_K}$ be such that $\mathcal{E}_\mathcal{B}(A) \vdash T : s$ for some sort $s \in \mathcal{S}_\mathcal{P}$. If T is a constructor term, we are done. Otherwise, if T contains a defined symbol, since $\mathcal{J}' \models_{\mathcal{I}nd} \mathcal{M}_s$, then there exists $T' \in \mathcal{T}_{\mathcal{C}(A)_K}$ such that $\mathcal{E}_\mathcal{B}(A) \vdash T = T'$ and $\mathcal{E}_\mathcal{C}(A) \vdash T' : s$. □

Theorem 94 *Let $\mathcal{J}$ be a parameterized specification which satisfies the conditions in Lemma 93. Assume that all literals occuring in the conditions of the rewriting rules from $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$ are literals over a parameter sort, and assume an oracle for deciding inductive consequences. Then, $\mathcal{J}$ is* complete *iff the inference system $\mathcal{SC}$ returns* success *for each $f \in \Sigma_\mathcal{D}$.*

Proof:

- $\Leftarrow$ Let $T \in \mathcal{T}_{\Omega_\mathcal{B}(A)_K}$ be such that $(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \Delta(A) \vdash T : s$ for some sort s. By Proposition 81 and Theorem 80, we conclude that $(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \Delta(A)$ is ground confluent and ground sort decreasing and the rewrite relation $\to_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \Delta(A)}$ is terminating for every algebra $A \in Alg_{\Omega_\mathcal{P}, \mathcal{E}_\mathcal{P}}$. Let T' be the normal form of T with respect to $(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \Delta(A)$. If T' is a constructor term, using the ground sort-decreasingness of $(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \Delta(A)$ we conclude that $(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \Delta(A) \vdash T' : s$. Now, since T' is a constructor term and no symbol from $\Sigma_\mathcal{D}$ occurs on the axioms from $\mathcal{E}_\mathcal{C}$, we have $(\mathcal{E}_\mathcal{C} - \mathcal{E}_\mathcal{P}) \cup \Delta(A) \vdash T' : s$.

  Otherwise, T' must contain a subterm T'' of the form $g(\overline{T})$ where $g \in \Sigma_\mathcal{D}$ and for all $i \in [1..n]$, $T_i \in \mathcal{T}_{\Omega_\mathcal{C}(A)_K}$. Using assumption 4.(iii), we can show that $(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \Delta(A) \vdash g(\overline{T}) : s'$ for some sort s'. Since the leaves of the pattern tree exhaust all cases by construction, this subterm must be an instance of a leaf S: $S\sigma = T''$, where σ is a ground substitution over $\mathcal{T}_{\Omega_\mathcal{C}(A)}$.

  Since S must be strongly ground reducible, there exists a non-empty sequence of conditional rules in $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$: $L_1 \to R_1$ `if` $P_1, \ldots, L_n \to R_n$ `if` P_n and a sequence of positions $u_1, u_2, \ \ldots\ , u_n$ in S such that $S/u_1 = T_1\sigma_1,\ S/u_2 = T_2\sigma_2,\ \ldots\ , S/u_n = T_n\sigma_n$ and $P_1\sigma_1 \ \vee\ P_2\sigma_2 \ \vee\ \ldots\ \vee\ P_n\sigma_n$ is an inductive theorem of $\mathcal{J}$.

  Then, there exists k such that $\mathcal{E}_\mathcal{B}(A) \vdash P_k\sigma_k$. By assumption, all literals occuring in $P_k\sigma_k$ are literals over a parameter sort, then Lemma 93 can be applied to conclude that $(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \Delta(A) \vdash P_k\sigma_k$. Since $(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \Delta(A)$ is ground confluent and the rewrite relation $\to_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \Delta(A)}$ is terminating for every algebra $A \in Alg_{\Omega_\mathcal{P}, \mathcal{E}_\mathcal{P}}$, all equalities and memberships in $P_k\sigma_k$ can be proved by normalization, and therefore the rule $L_k \to R_k$ `if` P_k can be applied to simplify T''. This contradicts the fact that T' is in normal form.

- $\Rightarrow$ Let $(f(\overline{T}), \Gamma)$ be a pattern. Since $f \in \mathcal{D}$, the pattern is ground reducible. Assume that **Decompose Sort** does not apply, hence all variables in the pattern inhabit free sorts. Assume further that **Decompose Variable** does not apply. As a consequence, the pattern does not contain an induction variable and therefore it is strongly ground reducible, hence **Reducible Leaf** applies, resulting eventually in **Success**. □

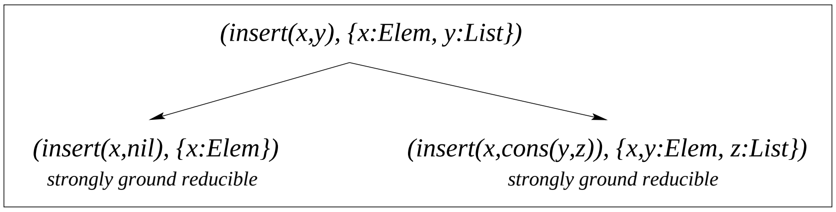


Figure 9.5: The pattern tree of *insert*

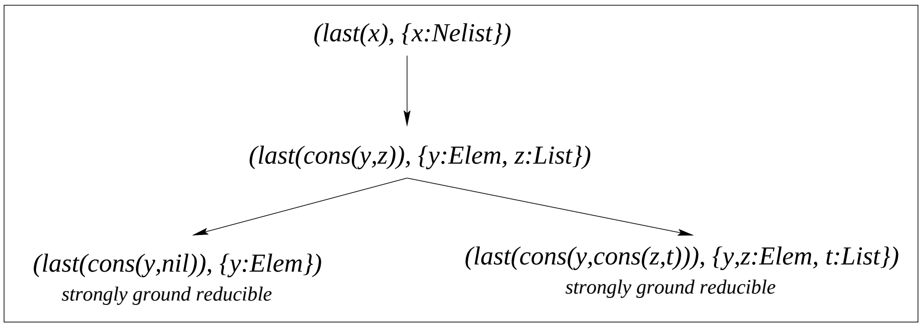


Figure 9.6: The pattern tree of *last*

Ground confluence is not necessary to check completeness if we use a *weak notion* of reducibility: Let $\mathcal{R}$ be a conditional rewriting/membership system and let $T[L\sigma]$ be a term. We say that T is *weakly reducible* by $\mathcal{R}$ if there exists a rule $L \to R$ `if` $P \in \mathcal{R}$ such that $\mathcal{R} \models_{\mathcal{I}nd} P\sigma$. Of course the notions of reducibility and weak reducibility are equivalent if $\mathcal{R}$ is ground confluent.

Example 95 (Example 69 Cont.) *Let us prove that the parameterized specification $\mathcal{J} = (\mathcal{P}, \mathcal{B})$ is complete. In Figure 9.5, we prove that the function* insert *is completely defined. Note that the pattern $(insert(x, nil),\ \{x : Elem\})$ is reducible and the pattern $(insert(x, cons(y, z)),\ \{x, y : Elem,\ z : List\})$ is strongly ground reducible since $x \leq y = true \vee x \leq y = false$ `if` $x, y : Elem$ is an inductive theorem w.r.t. the parameterized specification* [3]*. In Figure 9.6, we prove that the function* last *is completely defined. The patterns $(last(cons(y, nil)),\ \{y : Elem\})$ and $(last(cons(y, z)),\ \{y : Elem,\ z : Nelist\})$ are strongly ground reducible since they are reducible. In the same way, we can also prove that the function* sorted *is completely defined.*

9.1.2 Proving Inductive Theorems w.r.t. Parameterized Specifications

We shall now define what an inductive theorem in a parameterized specification is.

Definition 96 *Let $\mathcal{J} = (\mathcal{P}, \mathcal{B})$ be a parameterized specification. A* clause *C is an inductive theorem of $\mathcal{J}$ (or inductively valid), iff $F_{\mathcal{J}}(A)$ satisfies C for each model A of $\mathcal{P}$. This will be denoted by $\mathcal{J} \models_{\mathcal{I}nd} C$.*

[3]This follows from the membership axiom $x \leq y : Bool$ `if` $x, y : Elem$ in TOSET, and from the inductive lemmas $\neg(true = false)$ and $x = true$ or $x = false$ or $\neg(x : Bool)$ that can be inductively proved in TOSET because of the *protecting* BOOL declaration.

The next lemma gives us a useful characterization of inductive theorems.

Lemma 97 *A clause* $\neg\varphi_1 \vee \cdots \vee \neg\varphi_n \vee \varphi'_1 \vee \cdots \vee \varphi'_m$ *is an inductive consequence of* $\mathcal{J} = (\mathcal{P}, \mathcal{B})$ *iff for each* $\mathcal{P}$*-algebra* A *and for each* $\mathcal{K}_{\mathcal{B}}$*-kinded ground substitution* σ *assigning elements of* $\mathcal{T}_{\Omega_{\mathcal{B}}}(A)$ *to the variables* [4] *in the clause*

$$(\forall i \in [1..n] : \mathcal{E}_{\mathcal{B}}(A) \models \varphi_i\sigma) \;\; implies \;\; (\exists j \in [1..m] \; such \; that \; \mathcal{E}_{\mathcal{B}}(A) \models \varphi'_j\sigma)$$

Proof: By definition, $\mathcal{J} \models_{\mathcal{I}nd} C$ iff for each $\mathcal{P}$-algebra A we have $F_{\mathcal{J}}(A) \models C$ iff (by the construction of $F_{\mathcal{J}}(A)$) $\mathcal{T}_{\Omega_{\mathcal{B}}(A), \mathcal{E}_{\mathcal{B}}(A)} \models C$ iff for each ground substitution σ we have $\mathcal{E}_{\mathcal{B}}(A) \models C\sigma$. □

Example 98 *The following conjectures are inductive theorems w.r.t. the parameterized specification given in Figure 9.1 and 9.2:*

$$sorted(insert(x, y)) = sorted(y) \; \texttt{if} \; x : elem \; and \; y : List$$

$$x \leq last(y) = true \; \texttt{if} \; sorted(Cons(x, y)) = true \; and \; x : elem \; and \; y : Nelist$$

Inductive consequence has better properties when the parameterized specification is persistent. Essentially, satisfaction of clauses involving only symbols in $\mathcal{P}$ can, under reasonable assumptions, be reduced to ordinary logical consequence using the axioms in $\mathcal{P}$. The reasonable assumptions have to make sure that the clauses in question only involve the sorts, and are not affected by error elements in the kinds. This is formalized by the notion of a defined clause.

Definition 99 *Given a specification* $\mathcal{E} = ((\mathcal{K}, \Sigma), \mathcal{S}, \mathcal{E})$*, where for each kind* K *there is a top sort* $\top_K$ *of kind* K*, we call an atomic* $((\mathcal{K}, \Sigma), \mathcal{S})$*-clause* $(\forall\Gamma)\; T = T'$ $\mathcal{E}$*-defined if and only if:*

(i) the environment Γ *is a* total *function* $\Gamma : \overline{x} \to \mathcal{S}$*, and*

(ii) for each subterm U *of* T *or* T' *there exists a sort* $s \in \mathcal{S}$ *such that* $\Gamma \vdash_{\mathcal{E}} U : s$*.*

Similarly, an atomic $((\mathcal{K}, \Sigma), \mathcal{S})$*-clause* $(\forall\Gamma)\; T : s$ *is called* $\mathcal{E}$*-defined if and only if:*

(i) the environment Γ *is a* total *function* $\Gamma : \overline{x} \to \mathcal{S}$*, and*

(ii) for each subterm U *of* T *there exists a sort* $t \in \mathcal{S}$ *such that* $\Gamma \vdash_{\mathcal{E}} U : t$*.*

More generally, a $((\mathcal{K}, \Sigma), \mathcal{S})$*-clause* $(\forall\Gamma)\; \neg\varphi_1 \vee \ldots \neg\varphi_n \vee \varphi'_1 \vee \ldots \vee \varphi'_m$ $\mathcal{E}$*-defined iff* $(\forall\Gamma)\; \varphi_i$ *and* $(\forall\Gamma)\; \varphi'_i$ *are all* $\mathcal{E}$*-defined.*

The interest of considering $\mathcal{E}$-defined clauses is that their satisfaction in an $\mathcal{E}$-algebra A only depends on the restriction of A to its sorts. As already explained when discussing strict specifications, such a restriction takes the form of a partial Ω-algebra A° [75], where for each sort $s \in \mathcal{S}$, we have $A^\circ_s = A_s$, and where for each $f : K_1 \ldots K_n \to K$ in Σ, A°_f is the partial function $A_f \cap (A^\circ_{\top_{K_1}} \times \ldots \times A^\circ_{\top_{K_n}} \times A^\circ_{\top_K})$. The key point about $\mathcal{E}$-defined clauses is the following.

[4] For simplicity, we assume that the variables are many-kinded, that is, that an environment Γ has been added to the clauses as a collection of additional atoms. Of course, if Γ is an environment having sort assignments $x_1 : s_1, \ldots, x_n : s_n$, and assigning only kinds to the remaining variables $x_{n+1} : K_1, \ldots, x_{n+m} : K_m$, we then define $\mathcal{J} \models_{\mathcal{I}nd} (\forall\Gamma)C$ iff $\mathcal{J} \models_{\mathcal{I}nd} (\forall\overline{x})C \vee \neg x_1 : s_1 \vee \ldots \vee \neg x_n : s_n$.

Lemma 100 *If* $(\forall\Gamma)\ C$ *is an* $\mathcal{E}$*-defined clause, then given an* $\mathcal{E}$*-algebra* A*, we have*

$$A \models (\forall\Gamma)\ C \quad \textit{iff} \quad A^{\circ} \models (\forall\Gamma)\ C,$$

The key observation is that in an $\mathcal{E}$-defined clause C, for any assignment a and any term T in C, $\tilde{a}(T)$ is always defined in A°, and therefore both notions of satisfaction coincide.

Theorem 101 *Let* $\mathcal{J} = (\mathcal{P}, \mathcal{B})$ *be a persistent parameterized specification, and let* $(\forall\Gamma)\ C$ *be a* $\mathcal{P}$*-defined-clause. Then*

$$\mathcal{J} \models_{\mathcal{I}nd} (\forall\Gamma)\ C \quad \textit{iff} \quad \Gamma \vdash_{\mathcal{P}} C$$

Proof: Since C only involves symbols in $\mathcal{P}$, we have $\mathcal{J} \models_{\mathcal{I}nd} (\forall\Gamma)C$ iff $\forall A \in Alg_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}$ $F_{\mathcal{J}}(A) \models (\forall\Gamma)\ C$. This implies that for the $\mathcal{P}$-context Γ, $\mathcal{T}_{\Omega_{\mathcal{B}},\mathcal{E}_{\mathcal{B}}}(\Gamma) \models (\forall\Gamma)\ C$. But notice that, by the persistence hypothesis, for each kind $K \in K_{\mathcal{P}}$ we have bijections

$$\mathcal{T}_{\Omega_{\mathcal{B}},\mathcal{E}_{\mathcal{B}}}(\Gamma)|^{\circ}_{\mathcal{P},\top_K} = \mathcal{T}_{\Omega_{\mathcal{B}},\mathcal{E}_{\mathcal{B}}}(\Gamma)_{\top_K} \overset{\eta_{\top_K}}{=} \mathcal{T}_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}(\Gamma)_{\top_K} = \mathcal{T}_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}(\Gamma)^{\circ}_{\top_K}$$

and by η being a $\mathcal{P}$-homomorphism we have therefore an isomorphism $\mathcal{T}_{\Omega_{\mathcal{B}},\mathcal{E}_{\mathcal{B}}}(\Gamma)|^{\circ}_{\mathcal{P}} \simeq \mathcal{T}_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}(\Gamma)^{\circ}$. Therefore, by $(\forall\Gamma)\ C$ being $\mathcal{P}$-defined, we have $\mathcal{T}_{\Omega_{\mathcal{B}},\mathcal{E}_{\mathcal{B}}}(\Gamma) \models (\forall\Gamma)\ C \Leftrightarrow \mathcal{T}_{\Omega_{\mathcal{B}},\mathcal{E}_{\mathcal{B}}}(\Gamma)|_{\mathcal{P}} \models (\forall\Gamma)\ C \Leftrightarrow \mathcal{T}_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}(\Gamma) \models (\forall\Gamma)\ C$, which by the construction of $\mathcal{T}_{\Omega_{\mathcal{P}},\mathcal{E}_{\mathcal{P}}}(\Gamma)$ out of the proof theory is equivalent to $\Gamma \vdash_{\mathcal{P}} C$. Conversely, if $\Gamma \vdash_{\mathcal{P}} C$, by completeness $F_{\mathcal{J}}(A)|_{\mathcal{P}} \models (\forall\Gamma)\ C$ for each $A \in Alg_{\Omega\mathcal{P},\mathcal{E}_{\mathcal{P}}}$, which by C being a $\mathcal{P}$-clause is equivalent to $F_{\mathcal{J}}(A) \models (\forall\Gamma)\ C$, as desired. $\Box$

In this chapter, we develop a goal-directed inductive proof system $\vdash_{\mathcal{I}(\mathcal{J})}$ along the lines of [12, 9]. Given a set of clauses $\mathcal{E}$, called conjectures, this proof system, when it does not diverge, returns a success or a disproof. The proof system $\vdash_{\mathcal{I}(\mathcal{J})}$ is *sound*, that is, the input clauses are inductive consequences of the axioms iff it diverges or returns a success.

More formally, one step in our proof system can be formalized as follows:

$$C \vdash_{\mathcal{I}(\mathcal{J})} \{C_1,\ \ldots, C_k\} \text{ implies } \mathcal{J} \models_{\mathcal{I}nd} C \Leftrightarrow C_1 \wedge \ldots \wedge C_k$$

where C is the conjecture to be proved, and $C_1, \ldots C_k$ are the new subgoals. Soundness of $\vdash_{\mathcal{I}(\mathcal{J})}$ implies that: if $C \vdash^{+}_{\mathcal{I}(\mathcal{J})} = \mathit{true}$, then $\mathcal{J} \models_{\mathcal{I}nd} C$.

9.2 Computation of Test Sets

The following (mutually recursive) definition introduces our notions of test sets and test substitutions.

Definition 102 *(Test Set) A* test term *is a ground-reducible order-sorted term whose variables inhabit free sorts.*
A test set $\mathcal{TS}(s, \mathcal{J})$ *at a free sort* s *is a cover set of* s *such that each instance of a test term by a test substitution is strongly ground reducible by* $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$*.*
A test substitution *for a test term* (T, Γ) *instantiates the induction variables of* (T, Γ) *by terms taken from a given test set whose variables are fresh. We denote by* $\mathcal{TS}\Sigma((T, \Gamma))$ *the set of all possible test substitutions for* (T, Γ)*.*

The next proposition is analogous to one from [10] and permits us to compute a test set in parameterized specifications. Let us first introduce the following notion: Given a non-empty set of rules $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$, we denote by $\mathcal{D}epth(\mathcal{J})$ the maximum non-variable depth of the left-hand sides of rules in $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$.

Proposition 103 *Let $\mathcal{J} = (\mathcal{P}, \mathcal{B})$ be a parameterized specification. Let $\mathcal{TS}' = \{(T, \Gamma) \mid$ T is a linear irreducible ground term of sort s such that $\mathcal{D}epth(T) \leq \mathcal{D}epth(\mathcal{J})$, and $\forall p \in Pos(T)$, $T|_p$ is a non-parameter variable iff $|p| = \mathcal{D}epth(\mathcal{J})$, and Γ is the associated environment$\}$. Then, the maximal, irredundant with respect to subsumption, subset $\mathcal{TS}$ of $\mathcal{TS}'$ is a test set.*

Proof: The proof is similar to the one given in [10]. □

Example 104 (Example 69 Cont.) *A test set of the sort* List *contains* $\{(nil, \emptyset);\ (cons(x, nil),\ \{x : Elem\})\ (cons(x, cons(y, z)),\ \{x, y : Elem,\ z : List\})\}$.

9.3 An inductive Procedure for Parameterized Specifications

Proof by induction for parameterized specifications allows us to have shorter and more structured proofs. A generic proof must be given only once and can be reused for each instantiation of the parameter. Many tools for proof by induction have been developed for non-parameterized specifications: The first type applies explicit induction arguments on the term structure [1, 13]. The second type involves a proof by consistency [82, 48, 54, 31, 2]. More recently, new methods were developed that do not rely on the completion framework [12, 9].

For the case of parameterized specifications, most of the work considers only the technique of proof by consistency [59, 4]. It is generally accepted that such techniques may be very inefficient since the completion procedure often diverges. For that reason, we adopt here a method along the lines of [8, 9] that does not require completion.

In the following, we adapt our technique for proof by induction for non-parameterized and order-sorted specfication [10] for the case of parameterized membership equational specifications. Our inference system builds inductive proofs by instantiating induction variables of a goal (or subgoal) with terms in a test set, and then simplifying the obtained instances by appropriate simplification rules, therefore producing new subgoals. We start describing the simplification rules that we use.

Inductive simplification and **Membership Simplification** simplify goals with axioms as well as instances of the induction hypotheses, provided they are smaller than the goal to be proved. Our well-founded ordering $\prec_c$ on clauses is generated from the ordering $\succ$ on terms and $\leq_{\mathcal{K}}$ on sorts (see [10]).

When simplifying a clause C, an environment $\mathcal{MC}ond(C)$ needs to be constructed by collecting all membership conditions of the form $x : s$ appearing negatively in C.

Rewrite Splitting simplifies a clause a subterm of which matches some rules of $\mathcal{E}_{\mathcal{B}}$. Each rewritten clause is valid in the context of the condition of the rule used for simplification; hence the need for: (i) accumulating the condition of the rule in the clause, and (ii) checking that the disjunction of the conditions is an inductive theorem of the specification in order to show that all cases are covered.

The inference system is displayed in Figure 9.8. Its rules apply to pairs $(\mathcal{E}, \mathcal{H})$, where $\mathcal{E}$ is the set of current conjectures and $\mathcal{H}$ is the set of inductive hypotheses. The **Decompose Sort** rules replace non-deterministically each non-free sort s in a membership $x : s$

Inductive Simplification:	$(\{C[S]\}, \mathcal{H}) \ \Rightarrow\ \{C[T]\}$
	if $(S, \mathcal{MC}ond(C)) \underset{\mathcal{E_D} \cup \mathcal{H}_{\prec_c C}}{\longrightarrow} (T, \mathcal{MC}ond(C))$
Membership Simplification:	$(\{S : s \vee \Gamma\}, \mathcal{H}) \ \Rightarrow\ \emptyset$
	if $(S, \mathcal{MC}ond(\Gamma)) \vdash^{\succ,0}_{\mathcal{E_D} \cup \mathcal{H}_{\prec_c S:s\vee\Gamma}} t$ and $s \geq_{\mathcal{K}} t$
Rewrite Splitting:	$C[f(\overline{L})]_p \ \Rightarrow\ \{C[R_i\sigma_i]_p \ \mathtt{if}\ P_i\sigma_i\}_{i\in[1..n]}$
	if $\begin{cases} f \in \mathcal{D}_{BODY} \\ \text{Variables in } f(\overline{L}) \text{ inhabit free sorts} \\ \mathcal{J} \models_{\mathcal{I}nd} P_1\sigma_1 \vee \ldots \vee P_n\sigma_n \end{cases}$
	where $\begin{cases} \{L_i \rightarrow R_i \ \mathtt{if}\ P_i\}_{i\in[1..n]} \text{ is the set of} \\ \text{linearized rules in } \mathcal{E_D} \\ \text{such that } f(\overline{L}) = L_i\sigma_i \end{cases}$
Tautology:	$(\{C\}, \mathcal{H}) \ \Rightarrow\ \emptyset$
	if C is a tautology
Valid:	$(\{C\}, \mathcal{H}) \ \Rightarrow\ \emptyset$
	if C is a parameter clause over $\mathcal{T}_{\Sigma_\mathcal{P}}(\mathcal{X})$ and $\mathcal{E_P} \models C$

Figure 9.7: Simplification Rules

by sorts in a cover sort of s. The **Narrowing** rules eliminate ground reducible terms in a clause by simplifying their instances by test substitutions, while deriving new conjectures considered as subgoals. **Simplify** reduces a conjecture according to the rules shown in Figure 9.7. **Subsume** is an additional simplification rule, which cannot be used as the other rules in **Simplify** [5].

Soundness of the above inference system is not straightforward. The main difficulty is to make sure that the rules preserve a counterexample when one exists. Indeed, we will more precisely show that the *minimal* such counterexample is preserved along a *fair* derivation. The idea already appeared in [12, 9].

Definition 105 *A* derivation *is a sequence of inference steps starting with a pair of the form* $(\mathcal{E}_0, \emptyset)$*, using the inference rules in* $\mathcal{I}(\mathcal{J})$*, written*

$$(\mathcal{E}_0, \emptyset) \vdash_{\mathcal{I}(\mathcal{J})} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{\mathcal{I}(\mathcal{J})} \cdots (\mathcal{E}_n, \mathcal{H}_n) \vdash_{\mathcal{I}(\mathcal{J})} \cdots$$

We say that a derivation is fair *if the set of persistent clauses* $(\cup_i \cap_{j\geq i} \mathcal{E}_j)$ *is empty or* $\{\perp\}$*. The derivation is said to be a* failure *in the latter case, and a* success *in the former.*

Finite success is obtained when the set of conjectures to be proved is exhausted. Infinite success is obtained when the procedure diverges, assuming fairness.

Theorem 106 *(Soundness) Let* $\mathcal{J}$ *be a persistent parameterized specification and let* $\mathcal{J}_\mathcal{C} \subset \mathcal{J}$ *be a persistent constructor subspecification. Let* $(\mathcal{E}_0, \emptyset) \vdash_{\mathcal{I}(\mathcal{J})} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{\mathcal{I}(\mathcal{J})} \ldots$ *be a successful derivation such that the conjectures in* $\mathcal{E}_0$ *are* $\mathcal{E}$*-defined. Then* $\mathcal{J} \models_{\mathcal{I}nd} \mathcal{E}_0$*.*

[5] Otherwise, the application of this rule may be incorrect.

Decompose Sort Pos: $\dfrac{(\mathcal{E} \cup \{C \vee x : s\}, \mathcal{H})}{(\mathcal{E} \cup \{C \vee \bigvee_{i \in I} x : s_i\}, \mathcal{H})}$

where $\{s_i\}_{i \in I}$ is a cover sort of s

Decompose Sort Neg: $\dfrac{(\mathcal{E} \cup \{C \vee \neg x : s\}, \mathcal{H})}{(\mathcal{E} \cup \{C \vee \neg x : s_i\}_{i \in I}, \mathcal{H})}$

where $\{s_i\}_{i \in I}$ is a cover sort of s

Inductive Narrowing: $\dfrac{(\mathcal{E} \cup \{C[S]\}, \mathcal{H})}{(\mathcal{E} \cup \{\mathcal{E}_\sigma\}_{\sigma \in \mathcal{TS}\Sigma(C)}, \mathcal{H} \cup \{C\})}$

if $(S, \mathcal{MC}ond(C))$ is a ground reducible non-constructor term

where $\forall \sigma \in \mathcal{TS}\Sigma(C),\ (\{C\sigma\}, \mathcal{E} \cup \mathcal{H} \cup \{C\}) \ \Rightarrow_S \ \mathcal{E}_\sigma$

Narrowing: $\dfrac{(\mathcal{E} \cup \{C[S]\}, \mathcal{H})}{(\mathcal{E} \cup \{(C\sigma){\downarrow_{\mathcal{E}_\mathcal{C}}}\}_{\sigma \in \mathcal{TS}\Sigma(S)}, \mathcal{H})}$

if $(S, \mathcal{MC}ond(C))$ is a ground reducible constructor term

Simplify: $\dfrac{(\mathcal{E} \cup \{C\}, \mathcal{H})}{(\mathcal{E} \cup \mathcal{E}', \mathcal{H})}$

if $(\{C\}, \mathcal{E} \cup \mathcal{H}) \ \Rightarrow \ \mathcal{E}'$

Subsume: $\dfrac{(\mathcal{E} \cup \{C\}, \mathcal{H})}{(\mathcal{E}, \mathcal{H})}$

if C is subsumed by another clause of $\mathcal{E}_\mathcal{B} \cup \mathcal{E} \cup \mathcal{H}$

Fail: $\dfrac{(\mathcal{E} \cup \{C\}, \mathcal{H})}{(\bot, \mathcal{H})}$

if no other rule applies to the clause C

Figure 9.8: $\mathcal{I}(\mathcal{J})$: Induction Inference Rules

Proof: The proof is by contradiction. Assume that $\mathcal{J} \not\models_{\mathcal{I}nd} \mathcal{E}_0$. Let now $\mathcal{S}$ be the set of minimal counterexamples with respect to the clause ordering. Note that $\mathcal{S} \neq \emptyset$ since $\mathcal{J} \not\models_{\mathcal{I}nd} \mathcal{E}_0$ and $\prec_c$ is well founded. Let C' be a counterexample in $\mathcal{S}$. Then, there exists an irreducible ground substitution γ, and a clause $C \in \mathcal{E}_k$ such that $C' = C\gamma$, and C is minimal in the following sense: (i) some ground instance of C is a minimal counterexample w.r.t. the ordering on clauses; and (ii) C is minimal w.r.t. the subsumption ordering.

By the assumption that the derivation is successful, **Fail** does not apply to C. We can easily show (again by contradiction) that no other rule applies to C, resulting in a contradiction by the fairness assumption. □

Example 107 *In the following we give the proof of the conjecture*

$$sorted(insert(x_1, x_2)) = sorted(x_2) \texttt{ if } x_1 : Elem \textit{ and } x_2 : List$$

As in Example 91, we assume that the lemma $x \leq y = true$ *or* $x \leq y = false$ if $x, y : Elem$ *is inductively valid. This is justified by the protecting importation of BOOL in TOSET, which ensures that true and false are the only Boolean values, and true* $\neq$ *false.*

We will use the following cover set: $\{(nil, \emptyset);\ (cons(x, y),\ \{x : Elem,\ y : List\})\}$ *for* List.

$$(\{sorted(insert(x_1, x_2)) = sorted(x_2) \texttt{ if } x_1 : Elem \textit{ and } x_2 : List\}, \emptyset)$$

$$\vdash_{\textbf{Inductive Narrowing}}$$

$$\begin{array}{c}
(\{true = sorted(nil); \\
sorted(Cons(x_1, cons(x_2, x_3))) = sorted(cons(x_2, x_3)) \texttt{ if} \\
x_1 \leq x_2 = true \textit{ and } x_1, x_2 : Elem \textit{ and } x_3 : List; \\
sorted(cons(x_2, insert(x_1, x_3))) = sorted(cons(x_2, x_3)) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1, x_2 : Elem \textit{ and } x_3 : List\}, \\
\{sorted(insert(x_1, x_2)) = sorted(x_1) \texttt{ if } x_1 : Elem \textit{ and } x_2 : List\})
\end{array}$$

$$\vdash^*_{\textbf{Subsume}}$$

$$\begin{array}{c}
(\{sorted(cons(x_2, insert(x_1, x_3))) = sorted(cons(x_2, x_3)) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1, x_2 : Elem \textit{ and } x_3 : List\}, \\
\{sorted(insert(x_1, x_2)) = sorted(x_1) \texttt{ if } x_1 : Elem \textit{ and } x_2 : List\})
\end{array}$$

$$\vdash_{\textbf{Inductive Narrowing}} \vdash^*_{\textbf{Simplify}}$$

$$\begin{array}{c}
(\{sorted(cons(x_2, cons(x_1, nil))) = sorted(cons(x_2, nil)) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1, x_2 : Elem; \\
sorted(cons(x_2, cons(x_3, insert(x_1, x_4)))) = sorted(cons(x_2, cons(x_3, x_4))) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1 \leq x_3 = false \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List; \\
sorted(cons(x_2, cons(x_1, cons(x_3, x_4)))) = sorted(cons(x_2, cons(x_3, x_4))) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1 \leq x_3 = true \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List\}, \\
\{sorted(cons(x_2, insert(x_1, x_3))) = sorted(cons(x_2, x_3)) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1, x_2 : Elem \textit{ and } x_3 : List; \\
sorted(insert(x_1, x_2)) = sorted(x_1) \texttt{ if } x_1 : Elem \textit{ and } x_2 : List\})
\end{array}$$

$$\vdash_{\textbf{Rewrite Splitting}}$$

$$\begin{array}{c}
(\{false = true \texttt{ if } x_1 \leq x_2 = false \textit{ and } x_2 \leq x_1 = false; \\
true = true \texttt{ if } x_1 \leq x_2 = false \textit{ and } x_2 \leq x_1 = true; \\
sorted(cons(x_2, cons(x_3, insert(x_1, x_4)))) = sorted(cons(x_2, cons(x_3, x_4))) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1 \leq x_3 = false \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List;
\end{array}$$

$$
\begin{gathered}
sorted(cons(x_2, cons(x_1, cons(x_3, x_4)))) = sorted(cons(x_2, cons(x_3, x_4))) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1 \leq x_3 = true \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List\}, \\
\{sorted(cons(x_2, insert(x_1, x_3))) = sorted(cons(x_2, x_3)) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1, x_2 : Elem \textit{ and } x_3 : List; \\
sorted(insert(x_1, x_2)) = sorted(x_1) \texttt{ if } x_1 : Elem \textit{ and } x_2 : List\})
\end{gathered}
$$

$\vdash$**Valid** $\vdash$**Tautology**

$$
\begin{gathered}
(\{sorted(cons(x_2, cons(x_3, insert(x_1, x_4)))) = sorted(cons(x_2, cons(x_3, x_4))) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1 \leq x_3 = false \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List; \\
sorted(cons(x_2, cons(x_1, cons(x_3, x_4)))) = sorted(cons(x_2, cons(x_3, x_4))) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1 \leq x_3 = true \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List\}, \\
\{sorted(cons(x_2, insert(x_1, x_3))) = sorted(cons(x_2, x_3)) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1, x_2 : Elem \textit{ and } x_3 : List; \\
sorted(insert(x_1, x_2)) = sorted(x_1) \texttt{ if } x_1 : Elem \textit{ and } x_2 : List\})
\end{gathered}
$$

$\vdash$**Rewrite Splitting**

$$
\begin{gathered}
(\{false = false \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1 \leq x_3 = false \textit{ and } x_2 \leq x_3 = false \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List; \\
sorted(cons(x_3, insert(x_1, x_4))) = sorted(cons(x_3, x_4)) \texttt{ if } x_1 \leq x_2 = false \textit{ and} \\
x_1 \leq x_3 = false \textit{ and } x_2 \leq x_3 = false \textit{ and } x_2 \leq x_3 = true \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List; \\
sorted(cons(x_2, cons(x_1, cons(x_3, x_4)))) = sorted(cons(x_2, cons(x_3, x_4))) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1 \leq x_3 = true \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List\}, \\
\{sorted(cons(x_2, insert(x_1, x_3))) = sorted(cons(x_2, x_3)) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1, x_2 : Elem \textit{ and } x_3 : List; \\
sorted(insert(x_1, x_2)) = sorted(x_1) \texttt{ if } x_1 : Elem \textit{ and } x_2 : List\})
\end{gathered}
$$

$\vdash$**Subsume** $\vdash$**Tautology**

$$
\begin{gathered}
(\{sorted(cons(x_2, cons(x_1, cons(x_3, x_4)))) = sorted(cons(x_2, cons(x_3, x_4))) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1 \leq x_3 = true \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List\}, \\
\{sorted(cons(x_2, insert(x_1, x_3))) = sorted(cons(x_2, x_3)) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1, x_2 : Elem \textit{ and } x_3 : List; \\
sorted(insert(x_1, x_2)) = sorted(x_1) \texttt{ if } x_1 : Elem \textit{ and } x_2 : List\})
\end{gathered}
$$

$\vdash$**Rewrite Splitting**

$$
\begin{gathered}
(\{false = sorted(cons(x_2, cons(x_3, x_4))) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1 \leq x_3 = true \textit{ and } x_2 \leq x_1 = false \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List\}, \\
sorted(cons(cons(x_3, x_4))) = sorted(cons(x_2, cons(x_3, x_4))) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1 \leq x_3 = true \textit{ and } x_2 \leq x_1 = true \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List\}, \\
\{sorted(cons(x_2, insert(x_1, x_3))) = sorted(cons(x_2, x_3)) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1, x_2 : Elem \textit{ and } x_3 : List; \\
sorted(insert(x_1, x_2)) = sorted(x_1) \texttt{ if } x_1 : Elem \textit{ and } x_2 : List\})
\end{gathered}
$$

$\vdash$**Valid**

$$
\begin{gathered}
(\{sorted(cons(cons(x_3, x_4))) = sorted(cons(x_2, cons(x_3, x_4))) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1 \leq x_3 = true \textit{ and } x_2 \leq x_1 = true \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List\}, \\
\{sorted(cons(x_2, insert(x_1, x_3))) = sorted(cons(x_2, x_3)) \texttt{ if} \\
x_1 \leq x_2 = false \textit{ and } x_1, x_2 : Elem \textit{ and } x_3 : List; \\
sorted(insert(x_1, x_2)) = sorted(x_1) \texttt{ if } x_1 : Elem \textit{ and } x_2 : List\})
\end{gathered}
$$

$\vdash$**Rewrite Splitting**

$$
\begin{gathered}
(\{sorted(cons(x_3, x_4)) = sorted(cons(x_3, x_4)) \texttt{ if } x_1 \leq x_2 = false \textit{ and} \\
x_1 \leq x_3 = true \textit{ and } x_2 \leq x_1 = true \textit{ and } x_2 \leq x_3 = true \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List; \\
sorted(cons(cons(x_3, x_4))) = sorted(cons(x_2, cons(x_3, x_4))) \texttt{ if } x_1 \leq x_2 = false \textit{ and} \\
x_1 \leq x_3 = true \textit{ and } x_2 \leq x_1 = true \textit{ and } x_2 \leq x_3 = false \textit{ and } x_1, x_2, x_3 : Elem \textit{ and } x_4 : List\}, \\
\{sorted(cons(x_2, insert(x_1, x_3))) = sorted(cons(x_2, x_3)) \texttt{ if}
\end{gathered}
$$

$$x_1 \leq x_2 = false \text{ and } x_1, x_2 : Elem \text{ and } x_3 : List;$$
$$sorted(insert(x_1, x_2)) = sorted(x_1) \texttt{ if } x_1 : Elem \text{ and } x_2 : List\})$$
$$\vdash_{\textbf{Tautology}} \quad \vdash_{\textbf{Valid}}$$

$$(\emptyset, \{sorted(cons(x_2, insert(x_1, x_3))) = sorted(cons(x_2, x_3)) \texttt{ if}$$
$$x_1 \leq x_2 = false \text{ and } x_1, x_2 : Elem \text{ and } x_3 : List;$$
$$sorted(insert(x_1, x_2)) = sorted(x_1) \texttt{ if } x_1 : Elem \text{ and } x_2 : List\})$$

Comon Delor: $\dfrac{(\mathcal{E} \cup \{C\}, \mathcal{H})}{(\mathcal{E}, \mathcal{H})}$

if C is ground irreducible, and $C \not\Rightarrow_{CD} \perp$

Disproof: $\dfrac{(\mathcal{E} \cup \{C\}, \mathcal{H})}{(\perp, \mathcal{H})}$

if no other rule applies to the clause C

Figure 9.9: **New rules for non-parameterized specifications**

9.4 Case of non-Parameterized Specifications

Now assume that we have a non-parameterized specification. We can define a new inference system from $\mathcal{I}(\mathcal{J})$ by removing the rules **Valid** and **Fail** and adding the new rules **Comon Delor** and **Disproof** (see Figure 9.9). These rules permit us to check whether a constructor clause, whose subterms are all ground irreducible, is valid in the initial model of the constructors. They transform a first-order formula into $\perp$ iff this formula is not valid in the initial algebra associated with the free sorts [23], by using a rewrite relation $\Rightarrow_{CD}$. In our method, a false conjecture will always be rejected by **Disproof**, the above rules operating in non-deterministic polynomial time for the considered fragment. The obtained inference system is sound, and refutationally complete (see [10]).

Chapter 10

Further Extensions

In this section, we discuss practical situations for which our framework needs enhancements.

10.1 Extra Variables in Conditions

In practice, it is often desirable to have new variables in the condition of a rewrite or a membership rule. This is however forbidden in our previous definition of a reductive system. We now allow extra variables in the *equality* conditions of a rule. We therefore consider a CRMS $\mathcal{R}$ whose rules satisfy the following format:

$$(\mathcal{V}ar(R) \cup \mathcal{V}ar(U)) \subseteq (\mathcal{V}ar(L) \cup \mathcal{V}ar(V) \cup \mathcal{V}ar(W)) \;\; \forall L \to R \;\texttt{if}\; \overline{U} : \overline{t'} \wedge \overline{V} \downarrow \overline{W} \in \mathcal{R}$$

$$(\mathcal{V}ar(U)) \subseteq (\mathcal{V}ar(L) \cup \mathcal{V}ar(V) \cup \mathcal{V}ar(W)) \;\; \forall L : s \;\texttt{if}\; \overline{U} : \overline{t'} \wedge \overline{V} \downarrow \overline{W} \in \mathcal{R}$$

Variables in $(\mathcal{V}ar(V) \cup \mathcal{V}ar(W)) - \mathcal{V}ar(L)$ are called *extra variables* of the rule.

The use of these rules needs a new definition of rewriting and reductivity, both borrowed from [28]:

Definition 108 *$S \longrightarrow^p T$ if $S|_p = L\sigma$ and $T = S[R\sigma]_p$ for some substitution σ and rule $L \to R$ $\texttt{if}$ $\overline{U} : \overline{t'} \wedge \overline{V} \downarrow \overline{W}$ such that $z\sigma$ is in normal form for each extra variable z, $\overline{U}\sigma : \overline{t'}$, and $\overline{V}\sigma \downarrow \overline{W}\sigma$.*

A CRMS $\mathcal{R}$, in this more general sense, is reductive *if there exists a reduction ordering $\succ$ such that*

(i) for each rewrite rule $L \to R$ $\texttt{if}$ $\overline{U} : \overline{t'} \wedge \overline{V} \downarrow \overline{W}$ in $\mathcal{R}$, $L\sigma \succ R\sigma$ for each substitution σ satisfying $U\sigma : t$, $V\sigma \downarrow W\sigma$, with the condition that $y\sigma$ is in normal form for each extra variable y,

(ii) for each rewrite/membership rule $L \ldots$ $\texttt{if}$ $\overline{U} : \overline{t'} \wedge \overline{V} \downarrow \overline{W}$ in $\mathcal{R}$, $L\sigma \succ U\sigma, V\sigma, W\sigma$ for each substitution σ satisfying $U\sigma : t$, $V\sigma \downarrow W\sigma$, with the condition that $y\sigma$ is in normal form for each extra variable y.

It turns out that all previous results, including confluence checks, completeness for solving goals and completeness of completion, are still true with this new definition of rewriting and reductive system. However, reductivity becomes harder to check, since we cannot rely anymore on the closure of the ordering by instantiation. The same is true for type decreasingness.

We conclude by discussing a specification of paths on a graph given at figure 10.1, which involves an extra variable in the condition of its last (membership) equation. This

```
fth GRAPH[N:TRIV,L:TRIV]*(Elt.N to Node, Elt.L to Label, Elt?.L to Label?)
kind Edge?[Edge]
op _—_⟶_ : Node × Label × Node → Edge?
endfth

fmod PATH[G:GRAPH[X,Y]*(rename Edge? by Path?)
subsort Edge < Path
subsort Label < LabelExp
cop _—_⟶_ : Node×LabelExp×Node→Path
cop _;_ : LabelExp×LabelExp→LabelExp [assoc,id:nil]
cop _;_ : Path×Path→Path?
cop id : Node→Path
eq id(N) = N—nil⟶N
eq (N—e⟶N');(N'—e'⟶N") = N—e;e'⟶N"
mb N—e;e'⟶N" : Path if N—e⟶N':Path and N'—e'⟶N":Path
end fmod
```

Figure 10.1: A specification of paths in a graph

raises several questions. Firstly, the termination proof cannot be done by the standard tools, as the recursive path ordering, but needs a specialized variant accounting for the sort information [56]. Secondly, the construction of the automaton: again, this can almost be done since the class of automata used to encode shallow conditional specifications is closed under projection, allowing for the handling of existential variables. The slight problem is that the condition in the last membership rule is not quite shallow: it checks whether two non-variable terms have the sort *Path*. We believe, however, that the notion of shallow specification can be extended so as to include this case, since the considered terms are indeed "subterms" (in some sense) of the left-hand side, hence they can be accessed (the encoding of the runs of the automaton makes it clear that such "subterms" should be allowed). Here, they are accessed by means of an existential quantification in the condition, but we could have written the specification in a different way, allowing accessing the same terms by using *selectors* instead (using them to obtain the source and target of a path). This shows that some work is still needed to better understand the most general kind of automata that are needed for membership equational logic.

10.2 Rewriting Modulo

As usual, everything works when the equational axioms have no condition (it is even possible to have overloading, see [7]), provided that (i) axioms $S = T$ are *sort preserving*, that is, $\Gamma \vdash^{\succ}_{\mathcal{R}} S : s$ iff $\Gamma \vdash^{\succ}_{\mathcal{R}} T : s$, and (ii) the set of variables is the same on both sides of each equation.

The modulo part may come in the form of a confluent and terminating rewrite system, see [68].

10.3 Hierarchical Rewriting and Built-In's

The idea here is to evaluate the conditions of a rule by using a subset of the whole set of rules which is already known as confluent. We could as well generalize our notion of rule by including built-in predicates, and considering that there is a built-in check for equality in an imported module. We can of course mix conditions evaluated recursively or in a built-in. So, this section should rather be seen as an extension of the previous formalism.

10.4 Solving Goals

We terminate this list of important future extensions of the framework by looking quickly at narrowing.

Given a specification $(\Omega, \mathcal{E})$ in membership equational logic, we turn our attention to solving *goals*, that is, existential sentences of the form

$$G = (\exists \Gamma)\ \overline{U} : \overline{s} \wedge \overline{V} = \overline{W}$$

in which Γ is an environment. Satisfaction of such sentences in an $(\Omega, \mathcal{E})$-algebra $\mathcal{A}$ is defined in the obvious way, that is

$$\mathcal{A} \models_\Omega G$$

iff there exists an assignment $a : \Gamma \longrightarrow \mathcal{A}$, called a solution of G, such that $\overline{a}(U_i) \in \mathcal{A}_{s_i}$ for $i \in [1..n]$, and $\overline{a}(V_j) = \overline{a}(W_j)$ for $j \in [1..m]$. Note that the membership declarations $x : s$ present in Γ can always be moved to the goal part without changing the set of solutions, hence allowing us to consider Γ as empty without loss of generality. If the goal is satisfied by all $(\Omega, \mathcal{E})$-algebras, we say that it is *solvable*, and write

$$\mathcal{E} \models_\Omega G$$

By results of [43], this is always equivalent to solvability in the initial algebra $\mathcal{T}_{\Omega,\mathcal{E}}(\emptyset)$, that is

$$\mathcal{E} \models_\Omega G \text{ iff } \mathcal{T}_{\Omega,\mathcal{E}}(\emptyset) \models G$$

Therefore, it is enough to find an admissible ground substitution $\sigma : \Gamma \longrightarrow \mathcal{T}_\Omega(\emptyset)$ that is a solution of G in $\mathcal{T}_{\Omega,\mathcal{E}}(\emptyset)$.

If all the sorts of $\mathcal{T}_{\Omega,\mathcal{E}}(\emptyset)$ are non-empty, this is always equivalent to finding a solution $\sigma : \Gamma \longrightarrow \mathcal{T}_\Omega(\emptyset)(\Gamma')$ in a free algebra generated by a set of variables, since we can then always compose with a substitution $\tau : \Gamma' \longrightarrow \mathcal{T}_\Omega(\emptyset)$ to get a solution in the initial algebra (see also [51]).

Let us assume that $\mathcal{R}$ is a confluent, sort decreasing CRMS, and that all sorts of $\mathcal{T}_{\Omega,\mathcal{E}_\mathcal{R}}$ are non-empty. Then, we can state a goal in the equivalent form:

$$G = (\exists \Gamma)\ \overline{U} : \overline{s} \wedge \overline{V} = \overline{w}$$

and use narrowing as a complete procedure to solve such goals.

We actually change the notation of our goals, by distinguishing membership goals of the form $U : s$, and equality goals of the form $V \downarrow W$, therefore exploiting implicitly the confluence hypothesis. It is part of the goal-solving process to infer sorts for the variables in the goals so as to satisfy these goals. As said before, we will consider that the environment Γ is empty, all sort information known about the variables being contained in the current goal. We will sometimes need to extract this information as an environment: an arbitrary goal G can be written as $\Gamma \wedge G'$, where Γ is an environment, that is a conjunction of memberships of the form $x : s$, in which a variable may not appear more than once.

We assume that $\mathcal{R}$ is a confluent, sort-decreasing, regular CRMS. Because of regularity, a goal like $x : s \wedge x : t$ is satisfiable iff s and t have common subsorts, and is always equivalent to the (possibly empty) disjunction of goals $\bigvee_{u \leq_{\mathcal{R}} s, u \leq_{\mathcal{R}} t} x : u$. In order to avoid disjunctions, we will use a don't-know non-deterministic rule in that case.

Definition 109 *A goal $\overline{x} : \overline{s}$ is a solved goal if it is an environment.*

An unsolved goal $x : s \wedge x : t \wedge G$ narrows (non-deterministically) to $x : u \wedge G$, for u a subsort common to s and t if any, and to False otherwise.

An unsolved goal $\Gamma \wedge G \wedge S : s$ narrows to $\Gamma \wedge G$ if $\Gamma \ \vdash \ S : s$. Otherwise,

(i) let σ be a most general unifier of S and L, where L is the lefthand side of the membership rule $L : t$ `if` $\overline{U} : \overline{t'} \wedge \overline{V} \downarrow \overline{W}$ such that $t \leq_{\mathcal{R}} s$. Then, $G \wedge S : s$ narrows to $G\sigma \wedge \overline{U}\sigma : \overline{t'} \wedge \overline{V}\sigma \downarrow \overline{W}\sigma$.

(ii) let σ be a most general unifier of $S|_p$ for some p and L, where L is the lefthand side of the rewrite rule $L \rightarrow R$ `if` $\overline{U} : \overline{t'} \wedge \overline{V} \downarrow \overline{W}$ such that $t \leq_{\mathcal{R}} s$. Then, $G \wedge S : s$ narrows to $G\sigma \wedge S\sigma[R\sigma]_p : s \wedge \overline{U}\sigma : \overline{t'} \wedge \overline{V}\sigma \downarrow \overline{W}\sigma$, while $G \wedge S = T$ narrows to $\mathcal{G}\sigma \wedge S\sigma[R\sigma]_p = T \wedge \overline{U}\sigma : \overline{t'} \wedge \overline{V}\sigma \downarrow \overline{W}\sigma$.

An unsolved goal $G \wedge S \downarrow T$ narrows to G if S and T are unifiable. Otherwise, let σ be a most general unifier of $S|_p$ and L, lefthand side of the rewrite rule $L \rightarrow R$ `if` $\overline{U} : \overline{t'} \wedge \overline{V} \downarrow \overline{W}$. Then, $G \wedge S \downarrow T$ narrows to $G \wedge \overline{U}\sigma : \overline{t'} \wedge \overline{V}\sigma \downarrow \overline{W}\sigma \wedge S[R\sigma]_p \downarrow T$.

We see that narrowing acts very much like an SLD computation, using in particular many-sorted unification. Actually,

Theorem 110 *For $\mathcal{R}$ a confluent, sort decreasing, regular CRMS, such that all the sort in $\mathcal{T}_{\Omega,\mathcal{E}_{\mathcal{R}}}$ are non-empty, narrowing is complete for solving goals, that is, returning a goal in solved form.*

Proof: Routine technique again, considering goals as multisets and using an induction on $(\succ \cup \rhd)_{mul}$. □

Of course, narrowing contains unification in membership equational logic, hence order-sorted unification, as a particular case. Specifically, a goal G is called a *unification problem* iff it is of the more restricted form

$$(\exists \Gamma) \ \overline{V} = \overline{W}$$

that is, if we are trying to solve equations where the existential variable must satisfy some environment Γ. Of course, for a general $\mathcal{R}$, even if the starting goal is a unification problem, narrowing may create new goals involving subgoals of the form $\overline{U} : \overline{s}$.

It is also clear that the numerous complete restrictions of narrowing studied in the literature should remain complete in this extended framework, for example, basic narrowing [49]. This strategy is interesting, since it is known to terminate in some cases, when it terminates from the right-hand sides of rules [49], or when the variables in lefthand sides are shallow, that is at depth one [18]. When the lefthand sides of rules are *constructor based*, that is, when their strict subterms are built up from free constructors, then narrowing can be made non redundant, hence very efficient [63].

Our narrowing procedure assumes regularity for the correctness of the second rule, whose goal is to transform a conjunction of memberships for variables into an environment. We could actually work with non-regular specifications by adopting a more general notion of environment, in which the same variable can be bound to different sorts of a same kind. The second rule would not be needed anymore, and the result of the narrowing process would therefore be an environment in this more general sense. Since a variable may have several sort assignments in this more general setting, e.g. $x : s \wedge x : t$, it is not at all clear whether there are terms inhabiting both sorts, hence solutions to solved forms. This question is of course undecidable in the general case, but becomes decidable when the specification of constructors is shallow conditional, allowing to use tree automata techniques for deciding whether a given Boolean combination of sorts is empty or not. This alternative is probably worth being explored.

Chapter 11

Membership Equational Logic as a Logical Framework

We summarize here results proved in [75] about the generality of membership equational logic and its suitability as a logical framework for algebraic specification.

Although membership equational logic is a very simple logic, it can faithfully represent very nicely many other logics, even more complex ones, used in algebraic specification. In particular, denoting membership equational logic by *MEqtl*, we have a conservative map of logics (in the sense of [71])

$$\Phi : \mathit{OSEqtl}^R \longrightarrow \mathit{MEqtl}$$

from (an appropriate version of) order-sorted equational logic to membership equational logic, and also a conservative map

$$\Psi : \mathit{PMEqtl} \longrightarrow \mathit{MEqtl}$$

from a partial version of membership equational logic—in which many other partial equational logics can be embedded—to membership equational logic. In this way, both partial and order-sorted algebra are subsumed in membership algebra.

These extensions to membership algebra are both particularly nice, in that they are *bicompatible extensions*, so that for each order-sorted (resp. partial membership) theory T there is a full inclusion (called the *extension functor*) of the category of algebras of T into the category of membership algebras for $\Phi(T)$ (resp. $\Psi(T)$) that has a right adjoint in the other direction, called the *restriction functor*. It then follows that initial algebras, free algebras, and relatively free algebras—for example, in parameterized constructions—are all preserved by both extension and restriction. Therefore, for all practical purposes we can do our computation and our proof-theoretic and model-theoretic reasoning for order-sorted or partial algebra specifications in their corresponding translations into membership equational logic.

In addition, not only is membership equational logic a special case of Horn logic with equality, denoted $\mathit{Horn}^=$, so that we have an obvious inclusion of logics $\mathit{MEqtl} \hookrightarrow \mathit{Horn}^=$, but we can also define what at the model-theoretic level amounts to another, slightly more general "inclusion" called an *embedding*, $\mathit{Horn}^= \hookrightarrow \mathit{MEqtl}$ so that in fact both logics have exactly the same expressive power to specify classes of models.

It should be noted that, model-theoretically, we have a strict hierarchy of types of classes of models

$$\mathit{Varieties} \subset \mathit{Semivarieties} \subset \mathit{Horn} \subset \mathit{PartialSemivarieties}$$

the first classes are specifiable by many-sorted equations, the second by conditional many-sorted equations, the third by Horn clauses, and the last by partial equational logic with conditional existence equations, and in particular by partial membership equational logic [75].

The last family of model classes can be characterized more abstractly as *finitely locally presentable categories* [33]. Mossakowsky [80] has shown how a wide range of partial algebra specification formalisms, including partial algebras with conditional existence equations, are in fact equivalent at the model-theoretic level, in that in fact they all specify the same categories of models up to equivalence, and are all "sublogics" of each other in an appropriate model-theoretic sense. The upper part of the diagram below shows "Mossakowski's web" of embeddings and weak embeddings of logics for partial algebra specification showing the essential equivalence between such formalisms. We refer to [80] for the details of this web and briefly summarize below his notation for the different additional logics of partiality:

- *LESketch* is the logic of models specifiable as limit-preserving functors into **Set** for left-exact sketches [3].

- *Limit* is the logic of models for limit theories in the sense of Coste [25].

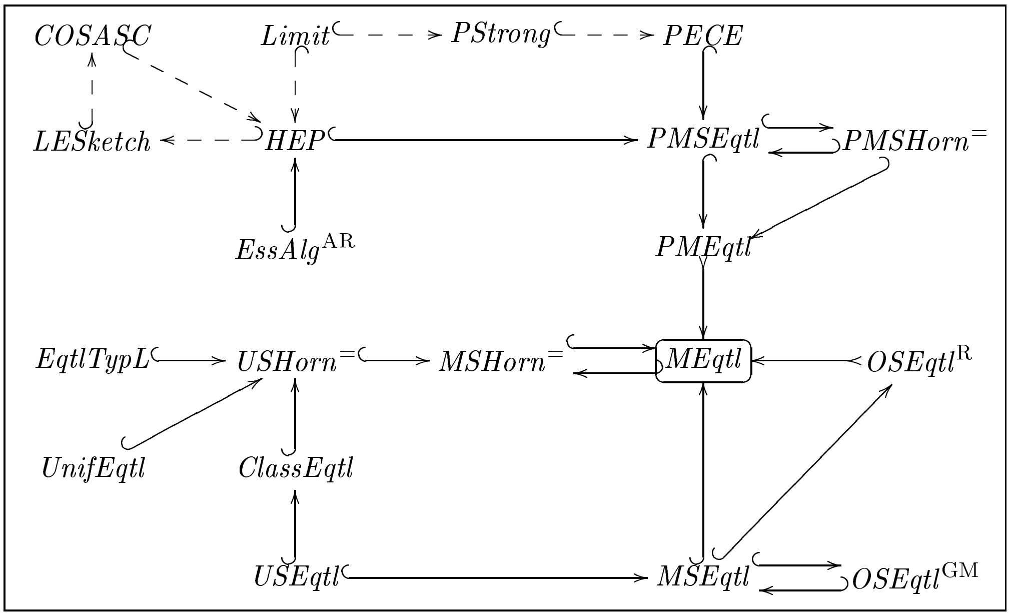
- *PStrong* is the logic of partial many-sorted equational logic with *strong* equations [14].

- *PECE* is the sublogic of *PMSEqtl* where all equations in a clause's conditions are "definedness" existence equations of the form $t = t$.

- *COSASC* is the logic of conditional order-sorted algebras with sort constraints in the sense of [41, 77].

In addition, we can consider the logic $PMSHorn^{=}$ of partial many-sorted Horn logic with equality, where the models are partial many-sorted algebras together with predicates, and the sentences are Horn clauses whose atomic formulas are either existence equations or predicates. In a way entirely analogous to the embedding $MSHorn^{=} \hookrightarrow MEqtl$, we also have an embedding $PMSHorn^{=} \hookrightarrow PMEqtl$. Furthermore, Mossakowski has also constructed mutual embeddings between *PMSEqtl* and a logic *RPCEL* essentially equivalent to $PMSHorn^{=}$.

The point is that, following such a web, all these logics of partiality can be embedded, or weakly embedded, in *PMEqtl*, which in turn, as mentioned above and shown in detail in [75], can be mapped in a bicompatible way into *MEqtl*.



Of course, such classes of partial models are intrinsically more complex than the classes of models that are Horn specifiable—or, equivalently, specifiable in membership equational logic—and require also more complex proof systems to reason about them. The attractive feature of membership equational logic is that, by using a bicompatible extension map, we can always embed those more complex logics into the simpler proof-theoretic and model-theoretic world of membership equational logic in a conservative way, and we can safely reason about free algebras, initial algebras, and parameterized data types in this simpler framework, being sure that the exact same results and constructions hold in the same way, via the extension adjunction, for their partial algebra counterparts. Furthermore, using a technique developed by Cerioli and Meseguer [17], we can borrow the simpler logic of membership algebra to endow a partial logic with sound and complete rules of deduction, and use an implementation such as that of Maude to mechanize such deduction.

The case of order-sorted equational logic with conditional equations has abstractly speaking the same power to specify classes of models as many-sorted equational logic with conditional equations [44], although it is of course much more expressive and convenient in practice. Therefore membership equational logic is strictly more expressive. This is so because membership axioms generalize sort constraints. However, this is done while remaining in a Horn framework, whereas the specification of sort constraints by equational partiality would push the models and the deduction to the more complex world of partial algebras. More generally, the lower part of the diagram above shows how different logics for *total* equational specification can be embedded in a conservative way that also preserve all the free and parameterized constructions into membership equational logic.

We have already discussed embeddings $\mathit{MSHorn}^{=} \hookrightarrow \mathit{MEqtl}$ and $\mathit{MEqtl} \hookrightarrow \mathit{MSHorn}^{=}$, demonstrating that both logics have essentially the same expressive power.

Since *unsorted* Horn logic with equality $\mathit{USHorn}^{=}$ is a special case sublogic of many-sorted Horn logic with equality, we obtain by composition an embedding

$$\mathit{USHorn}^{=} \hookrightarrow \mathit{MSHorn}^{=} \hookrightarrow \mathit{MEqtl}\ .$$

A number of important total equational specification formalisms are naturally embedded in *USHorn*$^=$, often as sublogics, including the following:

- Unsorted (conditional) equational logic *USEqtl*.
- The *equational type logic* (*EqtlTypL*) of Manca, Sallibra and Scollo [67], which is the special case of unsorted Horn logic with equality obtained by allowing only a binary "typing" predicate $\_ : \_$ besides equality.
- The *classified algebra* (*ClassEqtl*) of Wadge [87, 86] is the sublogic of membership equational logic obtained by allowing only a single kind. In spite of its very good properties, such a restriction to a single kind makes classified algebra strictly more limited in expressive power—it cannot for example represent the usual many-sorted formalisms by means of embeddings. Furthermore, the useful type-checking allowed by many-kinded specifications, by which we can throw away nonsensical expressions such as `true/7` that do not have a parse at the kind level, is not available in the one-kinded case.
- The *unified algebra* formalism (*UnifEqtl*) of Mosses [81] can also be regarded as essentially a sublogic of unsorted Horn logic with equality. Indeed, Mosses [81] defines in detail an embedding *UnifEqtl* $\hookrightarrow$ *USHorn*$^=$ showing that the models of a unified algebra specification are precisely the models of a corresponding theory in unsorted Horn logic with equality having a typing predicate $\_ : \_$ and a partial order predicate $\_ \leq \_$ in addition to the equality predicate, and satisfying suitable axioms that characterize the lattice-theoretic and typing properties of unified algebras.

Of course, the well-known case of (conditional) many-sorted equational logic (*MSEqtl*) is also a sublogic of membership equational logic, namely the sublogic obtained by making the set of sorts S empty. Finally, it is also well-known that (conditional) many-sorted equational logic is a sublogic of (conditional) order-sorted equational logic (*OSEqtl*) as formulated in [44], namely the case when the poset of sorts is discrete, and that, conversely, there is also an embedding of logics *OSEqtl* $\hookrightarrow$ *MSEqtl* [44], which composed with the embedding *MSEqtl* $\hookrightarrow$ *MEqtl* gives us an embedding *OSEqtl* $\hookrightarrow$ *MEqtl*. However, as already pointed out above, there is a more natural way of representing in a very direct way an appropriate variant of order-sorted equational logic in membership equational logic.

Since the logics just discussed constitute what might be called a representative collection of partial and total equational specification formalisms, in the sense that, although not being an exhaustive list of all the formalisms that have been proposed, seems to cover the most commonly used ones, the situation summarized in above diagram justifies our claim that membership equational logic is indeed a suitable and simple logical framework for equational specification.

Chapter 12

Conclusion

We have presented membership equational logic as a simple and general framework for algebraic specification that extends both order-sorted algebra and partial algebra approaches.

We have given precise conditions under which membership algebra specifications can be efficiently executed by rewriting. This provides an operational semantics for Maude's functional sublanguage, in which these rewriting techniques have been implemented [65]. The current Maude implementation can support efficient equational logic computation reaching up to 200K rewrites per second for typical examples on a 90 MHz Sun Hyper SPARC, which appears to be competitive with up-to-date implementations of PROLOG.

A number of directions for future research are suggested by the techniques and results that we have presented, including the following:

- Further generalization of the tree-automata techniques, to handle more complex membership tests that emerge naturally in membership equational specifications,
- Further development of the proving techniques for parameterized specifications,
- Weakening or removal of the sort-decreasingness conditions, as it was done in the order-sorted case by using tree automata [21],
- Elaboration of a higher-order membership equational logic along the lines of [69, 70] for the semantic part, and [55] for the operational part,
- Extension of membership equational logic with sort functions to achieve polymorphism in a more convenient way than via parameterization alone, as advocated first by Moses [81],
- More generally, the investigation of membership equational logic as a formalism for defining (dependent) inductive types from which more complex types could be generated as usual by means of function space construction and polymorphism.

Acknowledgments: We thank Benjamin Monate for reading our manuscript, as well as an anonymous referee who made numerous suggestions to improve the presentation.

Bibliography

[1] R. Aubin. Mechanizing structural induction. In *Theoretical Computer Science*, volume 9, pages 329–362, 1979.

[2] L. Bachmair. Proof by consistency in equational theories. In *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, pages 228–233, 1988.

[3] M. Barr and C. Wells. *Toposes, Triples and Theories.* Springer-Verlag, 1985.

[4] K. Becker. Inductive proofs in specifications parameterized by a built-in theory. SEKI-report SR-92-02, Universität Kaiserslautern, 1992.

[5] J.A. Bergstra and J.V. Tucker. Characterization of computable data types by means of a finite equational specification method. In J. W. de Bakker and J. van Leeuwen, editors, *7th Inter. Conference on Automata, Languages and Programming*, LNCS 81, pages 76–90. Springer-Verlag, 1980.

[6] B. Bogaert and S. Tison. Equality and disequality constraints on brother terms in tree automata. In A. Finkel, editor, *Proc. 9th Symp. on Theoretical Aspects of Computer Science, Paris*, LNCS 577, pages 161–172. Springer-Verlag, 1992.

[7] A. Boudet. Unification in order-sorted algebras with overloading. In D. Kapur, editor, *11th International Conference on Automated Deduction, Saratoga Springs, NY*, LNAI 607, pages 193–207. Springer-Verlag, June 1992.

[8] A. Bouhoula. Using Induction and Rewriting to Verify and Complete Parameterized Specifications. *Theoretical Computer Science*, 170(1 & 2):245–276, 1996.

[9] A. Bouhoula. Automated Theorem Proving by Test Set Induction. *Journal of Symbolic Computation*, 23(1):47–77, 1997.

[10] A. Bouhoula and J.-P. Jouannaud. Automata-driven automated induction. In *Proc. 12th IEEE Symp. Logic in Computer Science, Varsaw*, pages 14–25. IEEE Comp. Soc. Press, 1997.

[11] A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. In M. Bidoit and M. Dauchet, editors, *Theory and Practice of Software Development, Lille*, LNCS 1214, pages 67–92. Springer-Verlag, 1997.

[12] A. Bouhoula and M. Rusinowitch. Implicit Induction in Conditional Theories. *Journal of Automated Reasoning*, 14(2):189–235, 1995.

[13] R. S. Boyer and J. S. Moore. *A computational Logic Handbook.* Academic Press inc., 1988.

[14] P. Burmeister. Partial algebras—Survey of a unifying approach towards a two-valued model theory for partial algebras. *Algebra Universalis*, 15:306–358, 1982.

[15] A.-C. Caron, H. Comon, J.-L. Coquidé, M. Dauchet, and F. Jacquemard. Pumping, cleaning and symbolic constraints solving. In *Proc. Int. Conference on Algorithms, Languages and Programming, Jerusalem*, LNCS 820, pages 436–449. Springer-Verlag, 1994.

[16] A-C. Caron, J.-L. Coquidé, and M. Dauchet. Encompassment properties and automata with constraints. In Claude Kirchner, editor, *Proc. 5th Rewriting Techniques and Applications, Montréal*, LNCS 690, pages 328–342. Springer-Verlag, 1993.

[17] M. Cerioli and J. Meseguer. May I borrow your logic? (Transporting logical structure along maps). *Theoretical Computer Science*, 173:311–347, 1997.

[18] J. Christian. Some termination criteria for narrowing and e-narrowing. In D. Kapur, editor, *Proc. 11th Int. Conf. on Automated Deduction, Saratoga Springs, NY*, LNAI 607, pages 582–588. Springer-Verlag, 1992.

[19] H. Comon. Sufficient completeness, term rewriting systems and "anti-unification". In *Proceedings 8th International Conference on Automated Deduction, Oxford, England.*, July 1986.

[20] H. Comon. An effective method for handling initial algebras. In *Proc. 1st Workshop on Algebraic and Logic Programming, Gaussig, LNCS 343*. Springer-Verlag, November 1988.

[21] H. Comon. Inductive proofs by specifications transformation. In *Proc. 3rd Rewriting Techniques and Applications, Chapel Hill*, LNCS 355, pages 76–91. Springer-Verlag, April 1989.

[22] H. Comon. Completion of rewrite systems with membership constraints. In W. Kuich, editor, *Proc. 19th Int. Coll. on Automata, Languages and Programming, Vienna*, LNCS 623, pages . Springer-Verlag,1992.

[23] H. Comon and C. Delor. Equational formulae with membership constraints. *Information and Computation*, 112(2):167–216, August 1994.

[24] H. Comon and F. Jacquemard. Ground reducibility and automata with disequality constraints. In P. Enjalbert, editor, *Proc. 11th Symp. on Theoretical Aspects of Computer Science, Caen*, LNCS 775, pages 151–162. Springer-Verlag, 1994.

[25] M. Coste. Localisation, spectra and sheaf representation. In M. P. Fourman, C. J. Mulvey, and D. S. Scott, editors, *Applications of Sheaves*, LNM 753, pages 212–238. Springer-Verlag, 1979.

[26] N. Dershowitz. Well-founded orderings. In *Information Science Research Office, ATR-83-8478-3*, The Aerospace Corporation, El Segundo, California USA, 1983.

[27] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North-Holland, 1990.

[28] N. Dershowitz and M. Okada. A rationale for conditional equational programming. *Theoretical Computer Science*, 75:111–138, 1990.

[29] N. Dershowitz and G. Sivakumar. Solving goals in equational languages. In Stéphane Kaplan and J.-P. Jouannaud, editors, *Proceedings of the First International Workshop on Conditional Term Rewriting Systems*, Orsay, 1987. LNCS 308, pages 45–55, Springer-Verlag, 1988.

[30] H. Comon et F. Jacquemard. Ground reducibility is exptime-complete. In *Proc. 12th IEEE Symp. Logic in Computer Science, Varsaw*, pages 26–34. IEEE Comp. Soc. Press, 1997.

[31] L. Fribourg. A strong restriction of the inductive completion procedure. *Journal of Symbolic Computation*, 8(3):253–276, September 1989.

[32] K. Futatsugi, J. Goguen, J.-P. Jouannaud, and J. Meseguer. Principles of OBJ2. In *Proc. 12th ACM Symp. on Principles of Programming Languages, New Orleans*, pages 52–66, 1985.

[33] P. Gabriel and F. Ulmer. *Lokal präsentierbare Kategorien*. LNM 221, Springer-Verlag, 1971.

[34] J.H. Gallier and R.V. Book. Reductions in tree replacement systems. *Theorical Computer Science*, 37:123–150, 1985.

[35] H. Ganzinger. A completion procedure for conditional equations. *Journal of Symbolic Computation*, 11:51–81, 1991.

[36] H. Ganzinger. Order-sorted completion: The many-sorted way. *Theoretical Computer Science*, 89(1):3–32, 1991.

[37] I. Gnaedig, C. Kirchner, and H. Kirchner. Equational completion in order-sorted algebras. *Theoretical Computer Science*, 72:169–202, 1990.

[38] M. Gogolla. Algebraic specifications with partially ordered sets and declarations. Technical report, Universität Dortmund, Dortmund, West Germany, 1983.

[39] J. A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. *Introducing OBJ*, chapter Applications of Algebraic Specifications Using OBJ. Cambridge University Press, to appear. J. A. Goguen editor.

[40] J. Goguen. Order sorted algebra. Technical Report Semantics and Theory of Computation Report 14, UCLA, 1978.

[41] J. Goguen, J.-P. Jouannaud, and J. Meseguer. Operational semantics for order-sorted algebra. In W. Brauer, editor, *Proc. 12th Int. Coll. on Automata, Languages and Programming, Nafplion*, LNCS 194, pages 221–231. Springer-Verlag, 1985.

[42] J. Goguen, C. Kirchner, H. Kirchner, Aristide Megrelis, J. Meseguer, and T. Winkler. An introduction to OBJ 3. Technical Report 88-R-001, CRIN, Nancy, France, 1988.

[43] J. Goguen and J. Meseguer. Models and equality for logical programming. In Hartmut Ehrig, Giorgio Levi, Robert Kowalski, and Ugo Montanari, editors, *Proceedings, 1987 TAPSOFT*, LNCS 250, pages 1–22. Springer-Verlag, 1987.

[44] J. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.

[45] John V. Guttag and James J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10:27–52, 1978.

[46] John V. Guttag. Abstract data types and software validation. *Communications of the ACM*, 21:1048–1064, 1978.

[47] C. Hintermeier, C. Kirchner, and H. Kirchner. Dynamically-typed computations for order-sorted equational presentations. *Proc. 20th Int. Coll. on Automata, Languages and Programming, Jerusalem* LNCS 880, pages 450–461. Springer-Verlag, 1994. *to appear in Journal of Symbolic Computation*, 1998.

[48] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2):239–266, October 1982. Preliminary version in Proceedings 21st Symposium on Foundations of Computer Science, IEEE, 1980.

[49] J.-M. Hullot. Canonical forms and unification. In *Proc. 5th Conf. on Automated Deduction, Les Arcs, France, LNCS 87*, pages 318–334. Springer-Verlag, July 1980.

[50] Florent Jacquemard. *Automates d'arbres et Applications à la Réécriture*. Thèse de doctorat, Université Paris-Sud, Orsay, France, 1996.

[51] J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 257–321. MIT-Press, 1991.

[52] J.-P. Jouannaud, C. Kirchner,H. Kirchner, and A. Megrelis. OBJ: Programming with equalities, subsorts, overloading and parametrization. *Journal of Logic Programming*, 12:257–279, 1992.

[53] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal on Computing*, 15(4):1155–1194, November 1986.

[54] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82(1):1–33, July 1989.

[55] J.-P. Jouannaud and M. Okada. Abstract data type systems. *Theoretical Computer Science*, 173(2):349–391, February 1997.

[56] J.-P. Jouannaud and A. Rubio. A recursive path ordering for higher-order terms in η-long β-normal form. In Ganzinger H., editor, *Proc. 7th Rewriting Techniques and Applications, New-Jersey, LNCS 1103*. Springer-Verlag, 1996. To appear in Theoretical Computer Science.

[57] J.-P. Jouannaud and B. Waldmann. Reductive conditional term rewriting systems. In *Proc. Third IFIP Working Conference on Formal Description of Programming Concepts, Ebberup, Denmark*, pages 223–244, 1986.

[58] S. Kaplan. Conditional rewrite rules. *Theoretical Computer Science*, 33:175–193, 1984.

[59] Hélène Kirchner. A general inductive completion algorithm and application to abstract data types. In R. Shostak, editor, *Proceedings 7th International Conference on Automated Deduction, Napa Valley (Calif., USA)*, volume 170 of *Lecture Notes in Computer Science*, pages 282–302. Springer-Verlag, 1984.

[60] C. Kirchner and H. Kirchner. Order-sorted computations in G-algebras. Unpublished draft, January 1993.

[61] C. Kirchner,H. Kirchner, and J. Meseguer. Operational semantics of OBJ-3. In *Proc. 15th Int. Coll. on Automata, Languages and Programming, Tampere*, LNCS 317, pages 287–301. Springer-Verlag, 1988.

[62] E. Kounalis. Completeness in data type specifications. In B. Buchberger, editor, *Proceedings EUROCAL Conference, Linz (Austria)*, volume 204 of *Lecture Notes in Computer Science*, pages 348–362. Springer-Verlag, 1985.

[63] S. Krisher and A. Bockmayr. Detecting redundant narrowing derivations by the LSE-SL reducibility test. In *Proc. 4th Rewriting Techniques and Applications, Como, Italy*, LNCS 488, pages 75–85. Springer-Verlag, 1991.

[64] D. Lugiez and J.-L. Moysset. Tree automata help one to solve equational formulae in ac-theories. *Journal of Symbolic Computation*, 18(4):297–318, 1994.

[65] P. Lincoln M. Clavel, S. Eker and J. Meseguer. Principles of Maude. In J. Meseguer, editor, *Proceedings of the 1st International Workshop on Rewriting Logic and its Applications, Electronic Notes in Theoretical Computer Science 4*, 1996.

[66] M. Clavel, F. Duran, S. Eker and J. Meseguer. Building equational proving tools by reflection in rewriting logic. SRI-International, 1997.

[67] V. Manca, A. Salibra, and G. Scollo. Equational type logic. *Theoretical Computer Science*, 77:131–159, 1990.

[68] C. Marché. Normalised rewriting and normalised completion. In *Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science, Paris*, pages 394–403. IEEE Comp. Soc. Press, July 1994.

[69] N. Martí-Oliet and J. Meseguer. Inclusions and subtypes I: First order case. *J. Logic and Computation*, 6:409–438, 1996.

[70] N. Martí-Oliet and J. Meseguer. Inclusions and subtypes II: Higher-order case. *To appear in J. Logic and Computation.*

[71] J. Meseguer. General logics. In H.-D. Ebbinghaus et al., editor, *Logic Colloquium'87*, pages 275–329. North-Holland, 1989.

[72] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[73] J. Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In P. Wegner G. Agha and A. Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, pages 314–390. MIT-Press, 1993.

[74] J. Meseguer. Membership algebra, 1996. Lecture at the Dagstuhl Seminar on Specification and Semantics, July 96. Extended manuscript version distributed at the 1st Int. Workshop on Rewriting Logic.

[75] J. Meseguer. Membership algebra as a semantic framework for equational specification. Invited talk at WADT'97, Tarquinia, Italy, June 1997; In LNCS 1376, pp 18–61, 1998.

[76] J. Meseguer and J. Goguen. Initiality, induction and computability. In M. Nivat and J. Reynolds, editors, *Algebraic Methods in Semantics*, chapter 14. Cambridge Univ. Press, 1985.

[77] J. Meseguer and J. Goguen. Order-sorted algebra solves the constructor-selector, multiple representation and coercion problems. *Information and Computation*, 103(1):114–158, 1993.

[78] J. Meseguer and T. Winkler. Parallel programming in Maude. In J.B. Banâtre and D. Le Métayer, editors, *Research Directions in High-Level Parallel Programming Languages*, pages 253–293. Springer-Verlag, June 1991.

[79] B. Monate. Automates de formes normales et réductibilité inductive, 1997. Rapport de DEA.

[80] T. Mossakowski. Equivalences among various logical frameworks of partial algebras. In H. Kleine Büning, editor, *Computer Science Logic, Paderborn, Germany, September 1995, Selected Papers*, LNCS 1092, pages 403–433. Springer-Verlag, 1996.

[81] P. D. Mosses. Unified algebras and institutions. In *Proc. Fourth Annual IEEE Symp. on Logic in Computer Science, Asilomar, California*, pages 304–312, June 1989.

[82] D. R. Musser. On proving inductive properties of abstract data types. In *Proceedings 7th ACM Symp. on Principles of Programming Languages*, pages 154–162. ACM, 1980.

[83] A. Poigné. Parametrization for order-sorted algebraic specification. *Journal of Computer and System Sciences*, 40(2):229–268, April 1990.

[84] G. Smolka. Order-sorted horn logic: Semantics and deduction. Research Report SR-86-17, Univ. Kaiserslautern, October 1986.

[85] G. Smolka, W. Nutt, J. A. Goguen, and J. Meseguer. Order-Sorted Equational Computation. In Hassan Aït-Kaci and Maurice Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2, Rewriting Techniques*, chapter 10, pages 297–367. Academic Press, New York, N.Y., 1989.

[86] G. F. Stuart and W. W. Wadge. Classified model abstract data type specification. Manuscript, University of Victoria, 1991.

[87] W. W. Wadge. Classified algebras. Technical report, University of Warwick, 1982.