Specification and Proof in Membership Equational Logic*

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

[§] SRI International, Computer Science Laboratory
333 Ravenswood Avenue, Menlo Park, California 94025, USA
[†] INRIA Lorraine and CRIN, 615 rue du Jardin Botanique
B.P. 101, 54602 Villers-lès-Nancy Cedex, France
[‡] LRI, CNRS and Université de Paris-Sud
Bât 405, 91405 Orsay Cedex, France

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 basis 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.

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 order-sorted equational logic and partial algebra approaches, while Horn logic 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 sufficient completeness of a specification. Using tree-automata techniques, we develop a test set based approach for proving inductive theorems about a specification. Narrowing and proof techniques for parameterized specifications are investigated as well. Finally, we discuss the generality of our approach and how it extends several previous approaches.

*Supported by Office of Naval Research contracts N00014-95-C-0225 and N00014-96-C-0114, by the Information Technology Promotion Agency, Japan, and by the Centre National de la Recherche Scientifique, France

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 basis 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 [22, 20], which extends in substantial ways the OBJ language [10, 16].

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, 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 [20], 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 [6, 15].

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 disequality tests introduced in [4] and further studied in [8]. We also consider the extension of these techniques to reason about *parameterized* specifications satisfying a separability principle. 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. Separability then guarantees than the resulting conjectures can be broken into a parameterized part for which an oracle is to be used, and a constructor part to which tree automata techniques apply [7].

Due to space limitations, the set of references and the discussion of related work in the present version of this work are still incomplete. We nevertheless can mention that, besides extending the more standard formulation of order-sorted algebra [14], our approach has some similarities with the order-sorted approaches in [25] and in the work of Poigné. It is also quite close to the work of Wadge et al. on classified algebras, and has some similarities with the typed algebra approaches like those of Manca, Salibra and Scollo, of Mosses, of Hintermeier, Kirchner and Kirchner, and of Poigné.

Three additional papers further develop the ideas presented here in a summarized form: a full version of the present paper [3]; a detailed model theoretic study of the logic and the semantic connections with order-sorted and partial equational logics [21]; and an original study of the tree automata based inductive theorem proving techniques that are further developped here within the framework of membership equational logic [2].

We describe our Horn clause language in section 3. Functional computations with these Horn clauses is described in section 4, where confluence, type-decreasingness, and regularity are introduced. They are further investigated in section 5. Relationships with tree automata are investigated in section 6, and its application to compute induction schemas in Section 7. Sufficient completeness is adressed in section 8. Proving inductive consequences is sketched in section 9. Related work is discussed in section 10, and concluding remarks appear in section 11.

2 Preliminaries

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* Σ is made of: (i) a set of *kinds* $\mathcal{K}$; and (ii) a $\mathcal{K}^* \times \mathcal{K}$-indexed family of sets $\Sigma = \{\Sigma_{\overline{K} \to K}\}_{(\overline{K} \in \mathcal{K}^*, K \in \mathcal{K})}$ so that each *function symbol* $f \in \Sigma_{\overline{K} \to K}$ is equipped with input kinds in $\overline{K}$ 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 $\Sigma_{K_1 \times \ldots \times K_n \to K} \cap \Sigma_{K_1 \times \ldots \times K_n \to K'} = \emptyset$ if $K \neq K'$.

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 $\mathcal{A}_f : \mathcal{A}_{K_1} \times \ldots \times \mathcal{A}_{K_n}$. 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.

Given a $\mathcal{K}$-kinded set $\mathcal{X} = \uplus_{K \in \mathcal{K}} \mathcal{X}_K$ of *variables*, whose disjoint subsets $\mathcal{X}_K$, for $K \in \mathcal{K}$ are all denumerable (and disjoint from Σ), we define the set of *many-kinded 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. A term has a unique parse, hence a unique kind. The capital letters $L, M, N, R, S, T, U, V, W$ will denote terms.

Terms are identified with finite labelled trees as usual. *Positions* are strings of positive integers. Λ is the empty string (root position), $\cdot$ is the concatenation of strings. 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 $M \trianglerighteq M|_p$. We will use the property that $\rightarrow \cup \triangleright$ is well-founded for any terminating rewrite relation $\rightarrow$. The result of replacing $M|_p$ with N at position p in M is denoted by $M[N]_p$, where 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. Substitutions are written as in $\{x_1 \mapsto M_1, \ldots, x_n \mapsto M_n\}$, where M_i is assumed different from x_i. We use greek letters for substitutions and postfix notation for their application. 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 $mgu(S, T)$.

3 Language

Our language is a many-kinded first-order language whose only predicates are an infix equality, denoted by $\_ = \_$, and a family of unary membership predicates, denoted by $\_ : s$, where s ranges over a set of sorts, as defined later. 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 of a many-kinded signature $(\mathcal{K}, \Sigma, \mathcal{X})$, and of a disjoint $\mathcal{K}$-kinded family of sets of* sorts $\mathcal{S} = \{\mathcal{S}_K\}_{K \in \mathcal{K}}$. *$\mathcal{X}$ may be omitted if irrelevant.*

It is convenient to identify $\mathcal{S}_K$ with a subset of K, for all $K \in \mathcal{K}$. Identifying $\mathcal{S}_K$ with K itself would not be correct, since the kind K acts as a built-in error type for those computations taking place in kind K which do not return a value inhabiting a sort.

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, and s is a sort.* 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.*

Note that the universally quantified variables in the axioms are $\mathcal{K}$-kinded.

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 powerful enough to encode many (usually meta-theoretic) concepts: *Subsorts declarations* are syntactic sugar for membership axioms of the form $x : s \texttt{ if } x : s'$. *Order-sorted signature declarations* of the form $f : s_1 \times \ldots \times s_n \rightarrow s$ where $s_1, \ldots, s_n, s$ are sorts, are syntactic suger for 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.

3.2 Membership Algebras and Satisfaction

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

Definition 3 *For $\Omega = ((\mathcal{K}, \Sigma, \mathcal{X}), S)$ a signature in membership equational logic, an* Ω-algebra *is a Σ-algebra $\mathcal{A}$ together with the assignment to each sort $s \in K$ 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 $s \in K$, we have $f_K(\mathcal{A}_s) \subseteq \mathcal{B}_s$. This defines a category Alg_Ω in the obvious way.*

A $\mathcal{K}$-kinded map $a : \mathcal{X} \rightarrow \mathcal{A}$, called an *assignement*, extends in a unique way, by the freeness of the $\mathcal{K}$-kinded algegra $\mathcal{T}_\Sigma(\mathcal{X})$, to a Σ-homomorphism $\overline{a} : \mathcal{T}_\Sigma(\mathcal{X}) \rightarrow \mathcal{A}$. We then say that the Ω-algebra $\mathcal{A}$ with assignement 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 $\overline{a}(t) \in \mathcal{A}_s$.

Definition 4 *An Ω-algebra A satisfies a conditional axiom $(\forall \overline{x})\ \phi \texttt{ if } \phi_1 \wedge \ldots \wedge \phi_n$, written $A \models_\Omega (\forall \overline{x})\phi \texttt{ if } \phi_1 \wedge \ldots \wedge \phi_n$, iff $A, a \models_\Omega (\forall \overline{x})\phi$ for each assignment $a : \overline{x} \rightarrow A$ such that $A, a \models_\Omega (\forall \overline{x})\phi_i$ 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 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.*

Specifications in membership equational logic generalize the more familiar notion of order-sorted specifications, which have been the subject of numerous studies since their introduction by Joseph Goguen in the late seventies [12, 10, 14]. This work extends the order-sorted framework while keeping its conceptual elegance and making progress in four different directions. First, all terms are many-sorted, hence there is a well-defined syntactic notion of a term which makes sense. Second, our language provides for 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 3.4. Third, the logic is the simplest, yet most expressive, first-order logic we can think of for defining functions, a claim supported in Section 10. Fourth, as a Horn logic, it has a simple proof theory, and enjoys an initial algebra semantics. The latter is true of Order-sorted logic as well, but its proof theory is complicated by several technical anomalies that disappear in the richer framework of membership equational logic.

Figure 1 presents a specification of numbers, aiming at illustrating the expressive power of membership equational logic, 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. After the header, giving the name `NUMBER` to the specification, comes the imported module `BOOL` whose kind is called `Error-Bool`, and the many-sorted signature, with one kind, `Number`, refined in three sorts, `Nat`, `Int`, `Complex`, each one being a subsort of the next. We use the keywords `fmod` for functional modules, `cop` for constructors, `op` for defined symbols, `mb` for memberships and `eq` for equations. Expressions like `s : Number` shortcut the enumeration of all possible sorts in the kind `Number`. Important remarks are:

There are several categories of membership constraints. The first four encode the order-sorted signature of the constructors, while the next take care of the operations. The latter five constraints are not necessary in theory, since the corresponding properties can be deduced for the initial model from the equations defining the operations by using an inductive argument. It is 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.

```
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
  op _+_, _*_ : Number × Number → Number [comm]
  op _>_ : Number × Number → Error-Bool
  op Conj, |_| : Number → Number
  mb 0 : Nat
  mb S(x) : s if x : s and s : {Nat, Int}
  mb P(x) : Int if x : Int
  mb <x,y> : Complex if x : Int and y : Int
  mb x+y, x*y : s if x : s and y : s and s : Number
  mb x-y : Nat if x,y : Nat and x>y = T
  mb x-y : Int if x,y : Int
  mb Conj(x) : Complex if x : Complex
  mb |x| : Nat if x : Complex
  mb x*x : Nat if x : Int
  mb x*y : Nat if x : Complex and y : Complex and y = Conj(x)
  eq P(S(x)) = x if x : Int
  eq S(P(x)) = x if x : Int
  eq <x,0> = x if x : Int
  eq (x>0) = T if x : Nat
  eq (0>S(x)) = F if x : Nat
  eq S(x)>S(y) = x>y if x,y : Int
  eq P(x)>P(y) = x>y if x,y : Int
  eq S(x)>P(y) = x>P(P(y)) if x,y : Int
  eq P(x)>S(y) = x>S(S(y)) if x,y : Int
  eq x+0 = x
  eq x+S(y) = S(x+y) if x : Int and y : Int
  eq x+P(y) = P(x+y) if x : Int and y : Int
  eq x-0 = x
  eq x-S(y) = P(x-y) if x : Int and y : Int
  eq x-P(y) = S(x-y) if x : Int and y : Int
  eq x*0 = 0
  eq x*S(y) = (x*y)+x if x : Int and y : Int
  eq x*P(y) = (x*y)-x if x : Int and y : Int
  eq <x,y>+<x',y'> = <x+x',y+y'> if x,x',y,y' : Int
  eq <x,y>*<x',y'> = <x*x'-y*y',x*y'+x'*y> if x,x',y,y' : Int
  eq x+<x',y'> = <x+x',y> if x,x',y' : Int
  eq x*<x',y'> = <x*x',x*y'> if x,x',y' : Int
  eq Conj(<x,y>) = <x,-y> if x,y : Int
  eq Conj(x) = x if x : Int
  eq |x| = x*Conj(x) if x : Complex
endfm
```

Figure 1: A specification of numbers in membership equational logic

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$\times$Nat and Int$\times$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 element* of kind Number when x,y are complex numbers. This is an example of the use of kinds to catch error terms.

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

The equation x+0 = 0 does not specify the sort of x. Since the signature is many sorted, x has to range over some kind, here the kind Number. This equation may in particular apply to a term of kind Number not belonging to any of the sorts Nat, Int, Complex. 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 apply to terms in a kind.

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 discussed later.

Figure 2 shows how a bounded stack of complex numbers with a recovery operator can be naturally specified in membership equational logic. This example also shows how sort constraints in order-sorted algebra [13] 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 variables with no sort assigned to them are of the appropriate kind, that can be inferred from the expressions in which they appear. Note also 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.

```
fmod BD-STACK is protecting NUMBER
  kind ErrStack[NeStack < Stack]
  cop Push : Number × ErrStack → ErrStack
  cop Empty : → ErrStack
  op Recover : ErrStack → ErrStack
  op Top : ErrStack → Number
  op Pop : ErrStack → ErrStack
  op Lenght : 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 Lenght (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
endfm
```

Figure 2: A specification of bounded stacks

3.4 Deduction, Soundness and Completeness

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 Π_K is S_K and the postfix notation $t : s$ is used for $s(t)$. Hence, there is a sound and complete inference system for membership equational logic.

Unlike kinds, sorts are of a semantic nature. Given a specification, a term has one kind, `0` has kind `Number` in the specification of figure 1. Using the axioms, we may be able to prove that the same term inhabits some specific sort of that kind. `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)`. Hence deduction of sorts and deduction of equalities depend on each other in our inference system: to test whether a given term has a given sort becomes semi-decidable. To this end, we will make use of *environments* assigning sorts to finitely many variables occuring in a proof: an environment is a partial $\mathcal{K}$-kinded function $\Gamma : \overline{x} \mapsto S$, where $\overline{x}$ is a finite vector of variables

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

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

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

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

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

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

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

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

Figure 3: Equality and membership judgements

with disjoint components, denoted as a set of pairs $x : s$ with x a variable in $\overline{x}$ and s the sort $\Gamma(x)$ if Γ is defined, or the kind K of x otherwise. We call *sorted term* a pair made of a term and an environment assigning sorts to (some of) the variables in the term.

We adopt the familiar view of deduction by using environments and judgements: our typing judgements 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 judgements are written as $\Gamma \vdash_{\mathcal{E}} M = N$ if the term M can be proved equal to the term N in the environment Γ by using the equations and memberships in $\mathcal{E}$. 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 inference system of figure 3. A term M has a (not necessarily unique) sort if there exists an environment Γ and a sort $s \in \Gamma$ such that $\Gamma \vdash_{\mathcal{E}} M : s$.

Theorem 6 *(Soundness and Completenes) For any atomic Ω-sentence φ, $\mathcal{E} \vdash_\Omega \varphi$ iff $\mathcal{E} \models_\Omega \varphi$.*

Given an Ω-algebra $\mathcal{A}$ and an environment Γ, an assignment $a : \Gamma \mapsto \mathcal{A}$ is a $\mathcal{K}$-kinded map $a : \overline{x} \mapsto \mathcal{A}$ of the corresponding variables such that $a(x) \in \mathcal{A}_s$ if $\Gamma(x) = s$.

Theorem 7 *(Initial and Free Algebras) For $(\Omega, \mathcal{E})$ a specification in membership equational logic such that $\Omega = (\mathcal{K}, \Sigma, \mathcal{X})$, there is an $(\Omega, \mathcal{E})$-algebra $\mathcal{T}_{\Omega,\mathcal{E}}(\mathcal{X})$ and an assignment $\eta_\mathcal{X} : \mathcal{X} \mapsto \mathcal{T}_{\Omega,\mathcal{E}}(\mathcal{X})$ such that for each assignment $a : \mathcal{X} \mapsto A$ with $A \in Alg_{\Omega,\mathcal{E}}$ there is a unique Ω-homomorphism $\overline{a} : \mathcal{T}_{\Omega,\mathcal{E}}(\mathcal{X}) \mapsto A$ 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 $Alg_{\Omega,\mathcal{E}}$.*

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 $\mathcal{T}_\Sigma(\mathcal{X})/\equiv_\mathcal{E}^X$, where

$$t \equiv_\mathcal{E}^\mathcal{X} t' \text{ iff } \mathcal{E} \vdash_\Omega (\forall \overline{x}) t = t'$$

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 vertue of *Subject Reduction*.

4 Computations

In this section, we provide an operational semantics for the efficient computation by rewriting supported by Maude for functional modules.

4.1 Conditional Rewriting and Membership Rules

The idea of reductive conditional rules appeared first in [18], was then generalized in [17] and again slightly in [9]. We adapt the latter.

Definition 8 *A CRMS, or* conditional rewriting/membership system, *is defined by* (conditional) membership rules *and* (conditional) rewrite rules*:*

$$L(\overline{x}) : s \texttt{ if } \overline{U}(\overline{x}) : \overline{t'} \wedge \overline{V}(\overline{x}) \Downarrow \overline{W}(\overline{x})$$
$$L(\overline{x}) \to R(\overline{x}) \texttt{ if } \overline{U}(\overline{x}) : \overline{t'} \wedge \overline{V}(\overline{x}) \Downarrow \overline{W}(\overline{x})$$

where $u \Downarrow v$ is a shorthand for $\exists w$ s.t. $u \xrightarrow{*} w \xleftarrow{*} v$. For both kinds of rules, L is called the lefthand side. *We usually omit mention of the set $\overline{x}$*

of variables. We denote by $\mathcal{E}_{\mathcal{R}}$ the set of Horn clauses obtained by replacing arrows and joinability symbols in the rules of $\mathcal{R}$ by the equality symbol $=$.

A reductive *CRMS $\mathcal{R}$ has two kinds of rules,* subsort membership rules *of the form $x : s$ if $x : t$, defining the subsort ordering $\leq_{\mathcal{K}}$ generated by the set of pairs $\{t \leq_{\mathcal{K}} s \mid x : s \text{ if } x : t \in \mathcal{R}\}$, and* reductive rules *satisfying the following* reductivity *requirement: there exists a reduction ordering $\succ$ s.t.*

(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 $L \to R$ if $\overline{U} : \overline{t} \wedge \overline{V} \Downarrow \overline{W}$ in $\mathcal{R}$,

(iii) $L\ (\succ \cup \rhd)^+\ U, V, W$, $\forall U \in \overline{U}, \forall V \in \overline{V}, \forall W \in \overline{W}$, s.t. for all rules $L \to R$ if $\overline{U} : \overline{t} \wedge \overline{V} \Downarrow \overline{W}$ and $L : s$ if $\overline{U} : \overline{t} \wedge \overline{V} \Downarrow \overline{W}$ in $\mathcal{R}$.

The subsort ordering definition does not involve deduction: a semantic definition would yield the same ordering under the assumption that the specification is confluent and sort-decreasing, a property introduced next.

Given a CRMS $\mathcal{R}$, we reformulate our inference system in figure 4, replacing equalities by rewrites or joinability, therefore exploiting the full power of rewriting to replace a search by a computation. The notation $\vdash^0_{\mathcal{R}}$ indicates that the rule used at the root of a proof tree is *Replacement* or *Membership*, and hence, $\Gamma \vdash_{\mathcal{R}} M \longrightarrow N$ appears as the reflexive transitive closure of $\Gamma \vdash^0_{\mathcal{R}} M \longrightarrow N$. We therefore write $M \longrightarrow_{\Gamma,\mathcal{R}} N$ for $\Gamma \vdash^0_{\mathcal{R}} M \longrightarrow N$, in which case we say that S is *reducible* in the environment Γ, $M \longrightarrow^*_{\Gamma,\mathcal{R}} N$ for $\Gamma \vdash_{\mathcal{R}} M \longrightarrow N$, and $M \Downarrow_{\Gamma,\mathcal{R}} N$ for $\Gamma \vdash_{\mathcal{R}} M \Downarrow N$, and use $M{\downarrow_{\Gamma,\mathcal{R}}}$ for the set of $\mathcal{R}$-normal forms (the $\mathcal{R}$-normal form if it is unique) of M in the environnement Γ. We also write $S :_{\Gamma,\mathcal{R}} s$ for $\Gamma \vdash^0_{\mathcal{R}} S : s$, in which case we say that S is *sortable* in the environment Γ, and $S :^*_{\Gamma,\mathcal{R}} s$ for $\Gamma \vdash_{\mathcal{R}} S : s$.

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

Definition 9 *A CRMS $\mathcal{R}$ is* sort decreasing *if whenever $M :_{\Gamma,\mathcal{R}} s$ and $M \longrightarrow N$ for some N and s, there exists $t \leq_{\mathcal{K}} s$ s.t. $N :_{\Gamma,\mathcal{R}} t$.*

Proposition 10 *Let $\mathcal{R}$ be a confluent, sort-decreasing CRMS, and $\mathcal{R}_{\mathcal{E}}$ be its associated set of Horn clauses. Then*

$$\Gamma \vdash_{\mathcal{R}_{\mathcal{E}}} M : s \text{ iff } M \xrightarrow[\Gamma,\mathcal{R}]{*} N :_{\Gamma,\mathcal{R}} t \text{ for some } N \text{ and } t \leq_{\mathcal{K}} s$$

$$\Gamma \vdash_{\mathcal{R}_{\mathcal{E}}} M = N \text{ iff } M \Downarrow_{\Gamma,\mathcal{R}} N$$

A slightly different version of the inference system, closer to the actual deduction system used in the Maude implementation when no evaluation strategies are declared for the operators, is obtained by eliminating *Subject Reduction* and rewriting instead the terms $\overline{U}\sigma$ before sorting them in the conditions of *Membership* and *Replacement.*

Variable:	$\dfrac{x : s \in \Gamma}{\Gamma \vdash_{\mathcal{R}} x : s}$
Subject Reduction:	$\dfrac{\Gamma \vdash_{\mathcal{R}} N : s \quad \Gamma \vdash_{\mathcal{R}} M \longrightarrow N}{\Gamma \vdash_{\mathcal{R}} M : s}$
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 $\overline{U}(\overline{x}) : \overline{t'} \wedge \overline{V}(\overline{x}) \Downarrow \overline{W}(\overline{x}) \in \mathcal{R}$
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 $\overline{U}(\overline{x}) : \overline{t'} \wedge \overline{V}(\overline{x}) \Downarrow \overline{W}(\overline{x}) \in \mathcal{R}$

Figure 4: Rules of deduction for sort-decreasing CRMS's

4.2 Decidability of Equality and Membership Statements

The key properties investigated here are decidability of rewriting and of computation of normal forms, termination, and confluence, which in turn imply decidability of equality and membership statements:

Proposition 11 *Assume that $\mathcal{R}$ is a reductive CRMS. Then, $\longrightarrow_{\mathcal{R}}$ is terminating. Furthermore, $M :_{\Gamma,\mathcal{R}} s$, $M \longrightarrow_{\Gamma,\mathcal{R}} N$ and $N \in M{\downarrow}_{\Gamma,\mathcal{R}}$ for some N are decidable properties of M and N.*

Theorem 12 *Assume that $\mathcal{R}$ is a confluent, reductive, sort decreasing CRMS, and let $\mathcal{R}_{\mathcal{E}}$ be its associated set of Horn clauses. Then $\Gamma \vdash_{\mathcal{R}_{\mathcal{E}}} M = N$ iff $M{\downarrow}_{\mathcal{R}} = N{\downarrow}_{\mathcal{R}}$, $\Gamma \vdash_{\mathcal{R}_{\mathcal{E}}} M : s$ iff $M{\downarrow} :_{\Gamma,\mathcal{R}} s'$ for some $s' \leq_{\mathcal{K}} s$, hence equality and membership are decidable.*

Bottom-up evaluation strategies allow obtaining reduced substitutions when matching a lefthand side of a rule, hence sorts can be computed once and for all and stored in the term structure. In case of multiple sorts for a given term, the combinatorial explosion may slow down the sort-checking, although Maude uses a very efficient implementation of sorts by boolean vectors. It is therefore interesting to have a kind of unique sort property:

Definition 13 *A specification is* regular *if each term has a unique minimal sort, and* strongly regular *if each term has a unique minimal sort w.r.t.* $\vdash^0_{\mathcal{R}}$.

Strong regularity allows to improve efficiency for arbitrary computation strategies. 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 $U : s$ cannot be true, therefore allowing us to avoid wasting time in normalizing the condition of the rule. This stronger notion of regularity appeared already in OBJ for the case of order-sorted equational logic, and is the one used in Maude [22].

5 Confluence Properties and Completion

We define first the Church-Rosser property needed in our framework, and show that it follows from confluence, sort-decreasingness and regularity.

Definition 14 *A specification $\mathcal{R}$ is Church-Rosser iff $\forall \Gamma, S, s, T, t$ such that $S \longleftrightarrow^*_{\Gamma,R} T$, $S :^0_\Gamma s$ and $T :^0_\Gamma t$, $\exists U, u$ such that $S \longrightarrow^*_{\Gamma,R} U$, $T \longrightarrow^*_{\Gamma,R} U$, $U :_\Gamma u$, with $u \leq_\mathcal{K} s$ and $u \leq_\mathcal{K} t$.*

Theorem 15 *A specification $\mathcal{R}$, whose righthand sides of rules are irreducible, is Church-Rosser iff it is confluent, sort-decreasing and regular.*

We now characterize our properties by means of critical inference steps:

Definition 16 *Given two conditional rewrite rules $L \to R$ if $\overline{U} : \overline{s'} \wedge \overline{V} \Downarrow \overline{W}$ and $G \to D$ 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 the* critical pair*:*

$L\sigma[D\sigma]_p = R\sigma$ 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 confluent *if $(L\sigma[D\sigma]_p)\gamma \Downarrow R\sigma\gamma$ for all substitutions γ which satisfy the condition of the critical pair.*

Given a conditional membership rule $L : s$ if $\overline{U} : \overline{s'} \wedge \overline{V} \Downarrow \overline{W}$ and a conditional rewrite rule $G \to D$ 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 the critical reduced membership

$L\sigma : s \to L\sigma[D\sigma]_p$ 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 sort decreasing *if for each substitutions* γ *satisfying the condition of the critical reduced membership, there exists* $t \leq_{\mathcal{K}} s$ *such that* $(L\sigma[D\sigma]_p)\gamma :^0_{\mathcal{R}} t$.

Given two conditional membership rules $L : s$ if $\overline{U} : \overline{s'} \wedge \overline{V} \Downarrow \overline{W}$ *and* $G : t$ 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 the* critical membership
$L\sigma : s, t$ 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 strongly regular *if for each substitutions* γ *satisfying the condition of the critical membership, there exist* $u \leq_{\mathcal{K}} s, t$ *such that* $L\sigma\gamma :^0_{\mathcal{R}} u$.

Critical reduced memberships were already used in OBJ to check for sort decreasingness. Note the use of plain unification in the definition of our critical pairs and memberships. Confluence, sort-decreasingness and regularity can now be reduced to their respective critical instances:

Theorem 17 *(i) (Sort-decreasingness) Let* $\mathcal{R}$ *be a confluent, reductive CRMS whose memberships are left-linear. Then* $\mathcal{R}$ *is sort decreasing iff its critical reduced memberships are sort decreasing.*

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

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

Confluence of critical pairs or sort decreasingness of critical reduced memberships is undecidable. Decidable sufficient conditions exist for conditional rewrite rules. Regularity is easy to refute, since the existence of a minimum for a given set of sorts does not depend on any computation. To infer regularity is as difficult as to infer confluence and sort-decreasingness, since the substitutions satisfying a given condition must be considered.

Sort-decreasing order-sorted specifications are not closed under computation of critical pairs. Comon solved this by showing that confluence of non-decreasing specifications was reducible to the confluence of critical pairs computed by a decidable restricted form of second order unification. We solve the same problem in a different way, by considering a more expressive specification language closed under computation of critical pairs and critical memberships (computed via plain unification). It is then easy to add these critical axioms to the starting specification, as it is done in Knuth and Bendix completion. The corresponding completion procedure achieves confluence and sort-decreasingness. Achieving regularity as well is possible at the price of adding new sorts at completion time.

6 Bottom-up Tree Automata

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 signature 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 [5].

Definition 18 *A* bottom-up tree automaton, *or simply* automaton, *is a quadruple $(\mathcal{S}, \leq_{\mathcal{S}}, \Sigma, \mathcal{F})$, where $(\mathcal{S}, \leq_{\mathcal{S}}, \Sigma)$ 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:

Definition 19 *To an automaton $\mathcal{A} = (\mathcal{S}, \leq_{\mathcal{S}}, \Sigma, \mathcal{F})$, we associate a many-sorted signature $(\mathcal{S}, \Sigma_{\mathcal{A}})$ and a rewrite system $\mathcal{R}_{\mathcal{A}}$ over the signature $\Sigma_{\mathcal{A}}$:*

$$\Sigma_{\mathcal{A}} = \{f_{\overline{s},s} : \overline{s} \to s\}_{f:\overline{s}\to s\in\Sigma} \cup \{s : s\}_{s\in\mathcal{S}}$$
$$\mathcal{R}_{\mathcal{A}} = \{f_{\overline{s},s}(\overline{s}) \to s\}_{f:\overline{s}\to s\ \in\Sigma} \cup \{s \to t\}_{s\leq_{\mathcal{S}} t}$$

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

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

Theorem 20 *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 computable bottom-up tree automaton $\mathcal{A}_{\mathcal{R}}$, called the* normal form *automaton of $(\Sigma, \mathcal{R})$ s.t.:*
(i) Each $\mathcal{R}$-irreducible ground term S is recognized at an accepting state u of the automaton, s.t. s is accessible from u by empty transitions iff $S :_{\mathcal{R}} s$.
(ii) Each $\mathcal{R}$-reducible ground terms T is recognized at the non-accepting state t of the automaton iff $T :_{\mathcal{R}} t$. If $\mathcal{R}$ is sort decreasing, the normal form S of T is recognized at a state t' s.t. t is accessible from t' by empty transitions.

The remark that the language of ground normal terms in normal form is recognizable is due to Gallier and Book for the simple case of left-linear many-sorted specifications, and to Comon for the general case of order-sorted specifications. When the set of rules has the unique normal form property, the automaton can be seen as a realization of the initial algebra:

Corollary 21 *Let* $(\Sigma, \mathcal{R})$ *be an order-sorted specification s.t.* $\mathcal{R}$ *is a set of left linear rewrite rules. Assume 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 give a simple example of an order-sorted specification of integers together with its associated normal form automaton in Figure 5.

```
obj 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 obj
```

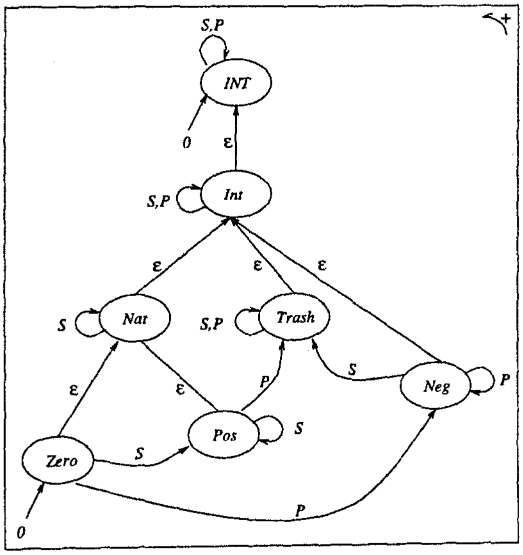

Figure 5: An order-sorted specification and its normal form automaton

Non-linear non-conditional rewrite and membership rules can also be expressed by using bottom-up tree automata with equality/disequality tests labelling the transitions [4]. The intuition 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. It is also possible to express associativity, commutativity, identity and idempotency, by labelling the transitions with formulae of Presburger's arithmetic [19].

7 Induction schemas

In this section, we relate normal form automata and induction.

Definition 22 *Given a CRMS* $\mathcal{R}$, *a term* (T, Γ) *is said to be* ground reducible *(resp.* irreducible, sortable*) if* $T\gamma$ *is reducible (resp. irreducible,* *sortable) for each irreducible ground substitution γ. We also say that T is ground reducible (irreducible, sortable) in the environment Γ.* $\diamond$

Definition 23 *A sort s is* free *iff every ground term inhabiting 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 every irreducible ground constructor term T inhabiting s inhabits a unique sort in $\mathcal{S}$. Cover sorts are extended to Cartesian products of sorts as expected.*

A finite set $\mathcal{T}$ of order-sorted terms inhabiting a free sort s is a cover set *of s iff every ground term inhabiting s is an instance of a term in $\mathcal{T}$.*

A test term *is a ground-reducible order-sorted term (T, Γ), all variables of which inhabit free sorts.* $\diamond$

The normal form automaton actually separates sorts into three categories, the free ones, the ones inhabited by reducible ground terms only, and the ones inhabited by both reducible and irreducible ground terms.

Definition 24 *(Induction Variables) Given a set $\mathcal{R}$ of rules, the set $\mathcal{I}nd\mathcal{P}os(f, \mathcal{R})$ of* induction positions *of $f \in \mathcal{F}$ is the set $\{p = i \cdot q \mid \exists f(\overline{L}) \rightarrow R$ `if` $C \in \mathcal{R}$, s.t. $q \in \mathcal{F}\mathcal{P}os(L_i))\}$. The set $\mathcal{I}nd\mathcal{V}ar((T, \Gamma), \mathcal{R})$ of* induction variables *of an order-sorted term (T, Γ) is the set $\{x \in \mathcal{X} \mid \exists p$ s.t. $T|_p = f(\overline{S}), \exists q$ s.t. $T|_{p \cdot q} = x$, and $q \in \mathcal{I}nd\mathcal{P}os(f, \mathcal{R})\}$.* $\diamond$

Since the initial algebra is characterized by terms inhabiting free sorts, other terms are eliminated by repeatedly instantiating them by elements in a cover set before simplifying them, which requires an additional property. To each non-left linear rule $L \rightarrow R$ `if` P, we associate its *linearized* version $L' \rightarrow R'$ `if` $P' \wedge P''$, such that L' is linear, $L = L'\sigma$ for some renaming σ, $R = R'\sigma$, $P = P'\sigma$, and $x = y \in P''$ iff $x\sigma = y\sigma$.

Definition 25 *A term (T, Γ) is* strongly ground reducible *if either:*

(i) T is reducible in the environment Γ, or

(ii) the formula $P_1\sigma_1 \vee \ldots \vee P_n\sigma_n$ is an inductive theorem of $\mathcal{R}$, where $\{L_i \rightarrow R_i$ `if` $P_i\}_{i \in [1..n]}$ is the set of linearized rules in $\mathcal{R}$ whose lefthand sides match a subterm of T with respective substitutions $\sigma_1, \ldots, \sigma_n$. $\diamond$

Case (ii) of strong reducibility is undecidable, while case (i) is a particular decidable case, but case (ii) can be checked (and hopefully solved) by using an inductive theorem prover, as the one described in Section 9. The following property is crucial for testing completeness of definitions in Section 8.

Property 26 *Let (T, Γ) be a test term free of induction variables. Then (T, Γ) is strongly ground reducible.*

8 Complete Definitions

The evaluation of any term should result in a term expressed by means of *constructors*, together with its sort. In Maude, specific keywords allow us to specify the constructors. Besides, the membership rules for the defined symbols specify the appropriate input sorts for which a function is completely defined, therefore always evaluates to a constructor term of the appropriate sort. Terms whose result is not a constructor term are considered as *error terms* inhabiting a kind, but no sort. Adapted from [3], our procedure for testing completeness of a function f exhibits the sorts on which f is only partially defined, so that this can be matched against the user declarations.

To prove completeness, all algorithms found in the literature assume either that constructors are free, or that rules for defined symbols are unconditional. Instead, we assume given a *complete specification*, which comes in two parts: a *complete specification of constructor symbols* $(\mathcal{C}, \mathcal{R}_\mathcal{C})$, and a *complete specification of defined symbols* $(\mathcal{D}, \mathcal{R}_\mathcal{D})$ relative to $(\mathcal{C}, \mathcal{R}_\mathcal{C})$.

8.1 Complete Specifications of Constructor Symbols

Constructor symbols may be free for some sorts, and completely defined in all other sorts. For example, the successor function S is free on $Zero \cup Pos$ and completely defined on Neg in the specification given in figure 5.

Definition 27 *Given a constructor specification $(\mathcal{C}, \mathcal{R}_\mathcal{C})$, obtained as a refinement by additional free sorts of the user-defined subspecification of constructor by using the tree automata technique given in Section 6, 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 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 ground reducible 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 complete *at sort $s_1 \times \ldots \times s_n$ if there exists a cover sort $\mathcal{S} = \{s^i_1 \times \ldots \times s^i_n\}_{i \in I}$ of $s_1 \times \ldots \times s_n$ such that c is free at all sorts in some subset $\mathcal{Q}$ of $\mathcal{S}$ and defined at all sorts in $\mathcal{S} - \mathcal{Q}$.*

A constructor c 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 (w.r.t. $\leq_\mathcal{K}$) free sort if so do its variables.

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

Theorem 28 ([5]) *It is decidable whether a specification of constructors in membership equational logic is complete.*

8.2 Complete Specifications of Defined Symbols

We denote by $\mathcal{T}_{\Omega,\mathcal{R}}|_{\mathcal{C}}$ the restriction to the constructor signature $\mathcal{C}$ of the initial algebra $\mathcal{T}_{\Omega,\mathcal{R}}$, and by h the unique $\mathcal{C}$-homomorphism $\mathcal{T}_{\mathcal{C},\mathcal{R}_\mathcal{C}} \mapsto \mathcal{T}_{\Omega,\mathcal{R}}|_{\mathcal{C}}$.

Definition 29 *Let $(\Omega, \mathcal{R})$ be a specification. A specification of defined symbols is* complete *relative to the constructor subspecification $(\mathcal{C}, \mathcal{R}_\mathcal{C})$ iff the unique homomorphism $h : \mathcal{T}_{\mathcal{C},\mathcal{R}_\mathcal{C}} \to \mathcal{T}_{\Omega,\mathcal{R}}|_{\mathcal{C}}$ is injective and for each sort s the component $h_s : (\mathcal{T}_{\mathcal{C},\mathcal{R}_\mathcal{C}})_s \to (\mathcal{T}_{\Omega,\mathcal{R}}|_{\mathcal{C}})_s$ is bijective.* ◇

The idea is that each ground term of sort s can be proved equal to a constructor ground term having the sort s in the constructor specification $\mathcal{R}_\mathcal{C}$, and that $\mathcal{R}_\mathcal{D}$ does not impose new equalities on constructor terms. Error ground terms not having a sort are new error terms created by the symbols in $\mathcal{D}$. We now give an operational version of completeness:

Definition 30 *Let $(\Omega, \mathcal{R})$ be a specification in which $\mathcal{R}$ is a reductive CRMS. A function symbol $f \in \mathcal{D}$ is* operationally complete *relative to the constructor subspecification $(\mathcal{C}, \mathcal{R}_\mathcal{C})$ iff for each term T of the form $f(T_1, \ldots, T_n)$ where for all i, $T_i \in \mathcal{T}(\mathcal{C})$ and $T :_\mathcal{R} s$ for some sort s, there exists $T' \in \mathcal{T}(\mathcal{C})$ such that $f(T_1, \ldots, T_n) \to^+_\mathcal{R} T'$ and $T' :_\mathcal{R} s$. A specification of defined symbols is* operationally complete *relative to the constructor subspecification $(\mathcal{C}, \mathcal{R}_\mathcal{C})$ iff each $f \in \mathcal{D}$ is operationally complete.* ◇

Proposition 31 *Let $(\Omega, \mathcal{R})$ be a constructor specification in which $\mathcal{R}$ is a ground confluent, ground sort-decreasing, reductive CRMS, containing a subspecification of constructors $(\mathcal{C}, \mathcal{R}_\mathcal{C})$. Then, the completeness of $(\Omega, \mathcal{R})$ is equivalent to the operational completeness of $(\Omega, \mathcal{R})$.*

Sort declarations for defined symbols seem superflous when computing with ground terms: all sort declarations involving defined symbols are 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:

Theorem 32 *Let $(\mathcal{C} \cup \mathcal{D}, \mathcal{R})$ be a complete specification relative to a subspecification of constructors $(\mathcal{C}, \mathcal{R}_\mathcal{C})$ in which $\mathcal{R}$ is a confluent, ground sort-decreasing, reductive CRMS. Let $\mathcal{M}$ be the set of membership rules in $\mathcal{R}$ whose head contains a defined symbol.*

Then, all memberships in $\mathcal{M}$ are inductive consequences of $\mathcal{R}' = \mathcal{R} - \mathcal{M}$, and $\mathcal{R}'$ is a reductive system which is ground-confluent and ground sort-decreasing, and has the same ground normal forms as $\mathcal{R}$. Besides, if $\mathcal{M}'$ is a set of membership rules whose head contains a defined symbol and which are inductive consequences of $\mathcal{R}$, then $(\mathcal{C} \cup \mathcal{D}, \mathcal{R} \cup \mathcal{M}')$ is also ground confluent.

Operational completeness becomes undecidable in presence of conditional rewrite rules. A complete test is based on the notion of a pattern [2]:

Definition 33 *A* pattern *is an order-sorted term* $(f(\overline{T}), \{\overline{x} : \overline{s}\})$ *such that* $f \in \mathcal{D}$ *and* $T_i \in \mathcal{T}(\mathcal{C}, \mathcal{V}ar(\overline{T}))$ *for each* $T_i \in \overline{T}$. $\diamond$

Our test computes *pattern trees* for the defined symbols. A pattern tree for $f \in \mathcal{D}_{\overline{K} \to K}$ at sort $\overline{s} \in \overline{K}$ is a tree whose nodes are labeled by patterns, whose root is labeled by the initial pattern $(f(\overline{x}), \{\overline{x} : \overline{s}\})$, and such that the successors of each internal node labeled by a pattern $(f(\overline{T}), \Gamma)$ are obtained by either covering the sort or the set of values of an induction variable in $f(\overline{T})$. As a result of the covering operations, the patterns in the tree grow until they become strongly reducible. If there exists a symbol $f \in \mathcal{D}$ that is only partially defined, our procedure will output a description of the ground instances on which the function f is not defined.

9 Proof by Induction

Our method for proving inductive consequences of of a complete specification is adapted from [2]. It has three ingredients: by exhibiting free sorts for ground constructor terms, the normal form automaton allows us to compute a canonical induction schema. This schema is then used to eliminate all terms in a conjecture that have reducible instances, resulting when it terminates in conjectures whose (constructor) terms inhabit free sorts only. These conjectures are then solved by using a powerful theorem of Comon and Delor [7]. The obtained method is both sound and refutationally complete.

More precisely, our inference system $\vdash_{I(\mathcal{R})}$ builds inductive proofs by instantiating induction variables of a goal (or subgoal) with test terms, and then simplifying the obtained instances, therefore producing new subgoals. $\vdash_{I(\mathcal{R})}$ applies to pairs $(\mathcal{E}, \mathcal{H})$, where $\mathcal{E}$ is the set of current conjectures and $\mathcal{H}$ is the set of inductive hypotheses. Soundness and completeness proofs of our inference system follow [1], showing that a *minimal* counterexample clause is preserved along a *fair* derivation when one exists.

Finite success is obtained when the set of conjectures to be proved is exhausted. Infinite success is obtained when the procedure diverges, assuming fairness. When this happens, the thing to do is to guess and prove a lemma, which is used to subsume or simplify the generated infinite family of subgoals, therefore stopping the divergence. This is possible in our approach, since lemmas (proved beforehand) can easily be used in the same way as axioms are.

Theorem 34 *Assume given a complete specification. Then $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{E}_0$ iff $(\mathcal{E}_0, \emptyset) \vdash_{I(\mathcal{R})} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{I(\mathcal{R})} \ldots$ is a successful derivation.*

We obtain as a corollary that our inference system is refutationally complete: all fair derivations originating from $(\mathcal{E}_0, \emptyset)$ end up eventually in a disproof iff $\mathcal{R} \not\models_{\mathcal{I}nd} \mathcal{E}_0$. Let us point out that our procedure for checking inductive conjectures is sound when the symbols in $\mathcal{D}$ are not completely defined, but it is no more refutationnally complete: in case the given conjecture is not valid, there is no guarantee anymore that a counterexample will eventually be found. But divergence is precluded in this case, since divergence implies the validity of the inductive conjectures.

10 Generality of Membership Equational Logic

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 $Eqtl^{:}$, we have a conservative map of logics $\Phi : OSEqtl \longrightarrow Eqtl^{:}$ from order-sorted equational logic to membership equational logic, and a conservative map $\Psi : PEqtl \longrightarrow Eqtl^{:}$ from partial equational logic with conditional existence equations [24] to membership equational logic: both partial and order-sorted algebra are subsumed in membership algebra [21].

These extensions are *bicompatible*, so that for each order-sorted (resp. partial) theory T there is a full inclusion 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. 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, we can do our computation and 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 $\texttt{Horn}^{=}$, so that we have an obvious inclusion of logics $Eqtl^{:} \hookrightarrow \texttt{Horn}^{=}$, but we can also define what at the model-theoretic level amounts to another "inclusion" $\texttt{Horn}^{=} \hookrightarrow Eqtl^{:}$ 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

$$\textit{Varieties} \subset \textit{Semivarieties} \subset \textit{Horn} \subset \textit{PartialSemivarieties}$$

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 conditional existence equations.

The last family of model classes can be characterized more abstractly as *finitely locally presentable categories* [11]. Mossakowsky [23] 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.

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. 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.

11 Conclusion

Membership equational logic is a simple and general framework for algebraic specification that extends both order-sorted algebra and partial algebra approaches. We have given conditions under which membership algebra specifications can be efficiently executed by rewriting. These results extend in several directions: extra variables in conditions; rewriting modulo equational axioms like commutativity, associativity, identity, idempotency and their combinations; and parameterized specifications.

All this provides an operational semantics for Maude's functional sublanguage, in which these rewriting techniques have been implemented [20]. The current Maude interpreter implementation can support efficient equational logic computation reaching up to 200K rewrites per second for typical examples on a 90 MHz Sun Hyper SPARC [20], which appears to be competitive with up-to-date implementations of PROLOG and ML.

Directions for future research include the following: generalization of tree-automata techniques, to handle more complex membership tests that emerge naturally in membership equational specifications; 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; extension of membership equational logic with sort functions to achieve polymorphism in a more convenient way than via parameterization alone, as advocated by Moses; elaboration of a higher-order membership equational logic; and, more generally, investigating membership equational logic as a formalism for defining inductive types from which more complex types could be generated by means of function space construction and polymorphism.

References

[1] Adel Bouhoula. Automated Theorem Proving by Test Set Induction. *Journal of Symbolic Computation*, to appear.

[2] Adel Bouhoula and Jean-Pierre Jouannaud. Automata-driven automated induction. submitted, 1996.

[3] Adel Bouhoula, Jean-Pierre Jouannaud, and José Meseguer. Specification and proof in membership equational logic. Draft, 1996.

[4] A-C. Caron, J.-L. Coquidé, and M. Dauchet. Encompassment properties and automata with constraints. In *Proc. 5th RTA*, Montréal, LNCS 690, 1993.

[5] Hubert Comon. Inductive proofs by specifications transformation. In *Proc. 3rd RTA*, Chapel Hill, LNCS 355, 1989.

[6] Hubert Comon. Completion of rewrite systems with membership constraints. In *Proc. 19th ICALP*, Vienna, LNCS 623, 1992.

[7] Hubert Comon and Catherine Delor. Equational formulae with membership constraints. *Information and Computation*, 112(2):167–216, 1994.

[8] Hubert Comon and Florent Jacquemard. Ground reducibility and automata with disequality constraints. In *Proc. 11th STACS*, Caen, 1994.

[9] Nachum Dershowitz and Mitsuhiro Okada. A rationale for conditional equational programming. *Theoretical Computer Science*, 75:111–138, 1990.

[10] Kokichi Futatsugi, Joseph Goguen, Jean-Pierre Jouannaud, and Jose Meseguer. Principles of OBJ2. In *Proc. 12th ACM POPL*, 1985.

[11] P. Gabriel and F. Ulmer. *Lokal präsentierbare Kategorien*. Springer Lecture Notes in Mathematics No. 221, 1971.

[12] J. A. Goguen, J. W. Thatcher, and E. G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In *Current Trends in Programming Methodology, vol. 4*, pages 80–149, 1978.

[13] Joseph Goguen, Jean-Pierre Jouannaud, and José Meseguer. Operational semantics for order-sorted algebra. In *Proc. 12th ICALP*, LNCS 194, 1985.

[14] Joseph Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.

[15] Claus Hintermeier, Claude Kirchner, and Hélène Kirchner. Dynamically-typed computations for order-sorted equational presentations. *Proc. 20th ICALP*, Jerusalem, LNCS 700, 1994.

[16] Jean-Pierre Jouannaud, Claude Kirchner, Hélène Kirchner, and Aristide Megrelis. OBJ: Programming with equalities, subsorts, overloading and parametrization. *Journal of Logic Programming*, 12:257–279, 1992.

[17] Jean-Pierre Jouannaud and B. Waldmann. Reductive conditional term rewriting systems. In *Proc. Third IFIP Working Conference on Formal Description of Programming Concepts*, Ebberup, Denmark, 1986.

[18] Stephane Kaplan. Conditional rewrite rules. *Theoretical Computer Science*, 33:175–193, 1984.

[19] Denis Lugiez and Jean-Luc Moysset. Tree automata help one to solve equational formulae in ac-theories. *Journal of Symbolic Computation*, 18(4):297–318, 1994.

[20] M. Clavel, S. Eker, P. Lincoln and J. Meseguer. Principles of Maude. In *Proceedings of the 1st International Workshop on Rewriting Logic and its Applications, Electronic Notes in Theoretical Computer Science 4*, 1996.

[21] José Meseguer. Membership algebra, 1996. Lecture at the Dagstuhl Seminar on Specification and Semantics, Report 151, 9628, 19-21, 1996.

[22] Jose Méseguer and Timothy 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.

[23] T. Mossakowski. Equivalences among various logical frameworks of partial algebras. In Proc. *9th CSL*, Paderborn, 1995, LNCS 1092, 1996.

[24] H. Reichel. *Initial Computability, Algebraic Specifications, and Partial Algebras.* Oxford University Press, 1987.

[25] Gert Smolka. Order-sorted Horn logic: Semantics and deduction. Research Report SR-86-17, Univ. Kaiserslautern, October 1986.