*URL:* [*http://www.elsevier.nl/locate/entcs/volume59.html*](http://www.elsevier.nl/locate/entcs/volume59.html) *15 pages*

*Interpreting Abstract Interpretations in* Membership Equational Logic

# *Bernd Fischer and Grigore Ro su*

*Research Institute for Advanced Computer Science Automated Software Engineering Group*

*NASA Ames Research Center*

*Mo ett Field, California, 94035, USA* [*ffisch,grosug@ptolemy.arc.nasa.gov*](mailto:ffisch%2Cgrosug@ptolemy.arc.nasa.gov)

*Abstract*

*We present a logical framework in which abstract interpretations can be naturally speci ed and then veri ed. Our approach is based on membership equational logic which extends equational logics by membership axioms, asserting that a term has a certain sort. We represent an abstract interpretation as a membership equational logic speci cation, usually as an overloaded order-sorted signature with member- ship axioms. It turns out that, for any term, its least sort over this speci cation corresponds to its most concrete abstract value. Maude implements membership equational logic and provides mechanisms to calculate the least sort of a term ef-*

*ciently. We rst show how Maude can be used to get prototyping of abstract interpretations \for free." Building on the meta-logic facilities of Maude, we fur- ther develop a tool that automatically checks an abstract interpretation against a set of user-de ned properties. This can be used to select an appropriate abstract interpretation, to characterize the speci c loss of information during abstraction, and to compare di erent abstractions with each other.*

# *1 Introduction*

*Abstract interpretation [5] has become a unifying framework for a variety of* program analysis techniques. It is also increasingly nding application in other areas of computer science, e.g., model checking [2,13] and theorem proving [12]. Its appeal is precisely the involved abstraction which simpli es calculation and search, thus allowing us to eÆciently nd approximate but correct solutions to problems which may otherwise be too hard.

*In practice, however, nding the right abstraction for a problem at hand* is often all but trivial. Di erent abstractions o er di erent trade-o s be- tween eÆciency and approximation which can often be validated and evalu- ated only with a prototype implementation of the abstract interpretation. A

*c 2001 Published by Elsevier Science B. V. Open access under* [*CC BY-NC-ND license*](http://creativecommons.org/licenses/by-nc-nd/3.0/)*.*

*more de nitive evaluation of the abstraction, e.g., in the form of a correctness* or completeness proof, even requires support for formal veri cation.

*In this paper, we are addressing these validation and veri cation issues.* We show how abstract interpretation can be speci ed and interpreted in mem- bership equational logic (MEL) [11,1], an expressive logic which generalizes the various forms of equational logics used in algebraic speci cation tech- niques. With this re-interpretation, we can use the Maude system [4,3] as environment to execute the speci cation of an abstraction, or in other words, we get a prototyping environment \for free." We further developed a tool which uses Maude's meta-logic capabilities to verify abstract interpretations against properties that are also speci ed as MEL sentences. More precisely, we provide a Maude module that allows us to write commands like

*Maude> reduce verify PROPERTIES on ABSTRACTION .*

*where ABSTRACTION is a speci cation of an abstract interpretation and PROPERTIES* is another MEL speci cation. verify on returns a list of warnings showing all sentences in PROPERTIES that are violated by the abstract interpretation, together with (abstract) counterexamples. We can see at this stage three applications of such a tool, depending on what the speci cation PROPERTIES actually represents:

*If PROPERTIES is a set of desired properties an abstract interpretation should* have in order to be employed in a certain context, then we can use our tool to retrieve from a library abstract interpretations that satisfy them;

*If PROPERTIES is an axiomatic presentation of the concrete domain, then* we can use the tool to see how much and what kind of information the abstraction looses and then decide if it is acceptable for a speci c task;

*If PROPERTIES is another abstract interpretation of the same concrete do-* main, then an empty list of warnings means that ABSTRACTION re nes this second abstraction.

*The basic idea of our approach is relatively straightforward. If we think of* an abstract interpretation as a lattice, then a sort is associated to each node in the lattice and a subsort is associated to each edge. The tables de ning the abstract operations are replaced by a series of overloaded operators, while properties that cannot be captured by tables are declared via MEL sentences.

*sorts Bottom Neg Pos Real . subsorts Bottom < Neg Pos < Real . op \_+\_ : Neg Neg -> Neg .*

*op \_+\_ : Pos Pos -> Pos . op \_\*\_ : Neg Pos -> Neg . op \_\*\_ : Neg Neg -> Pos .*

*op \_/\_ : Real Real -> Real .*

*op \_/\_ : Real Zero -> Bottom . mb X / X : Pos-one .*

*is a fragment of the usual sign abstraction 1 . The abstract value of an ex-*

*1 The precise formalism used here will be explained later.*

*pression in the concrete domain then corresponds to its least sort over the* speci ed signature; an eÆcient automatic least sort calculation is provided by Maude.

*Our approach to abstract interpretation in MEL is more logic-oriented* than the usual lattice-oriented approach. We advocate a loose semantics for the speci cations corresponding to abstract interpretations. The intended abstract interpretation is then just one model of the speci cation, namely that one that adds exactly one new element to each carrier, besides those resulting from the inclusion of carriers required by the subsort declarations. Another model is the one that structures the concrete domain into subsets of elements of the same \type". This model is thus similar to the powerset construction usually used as concrete domain in the lattice-oriented approach.

*The ultimate goal of abstract interpretations is to simplify reasoning about* the concrete domain. We thus prefer to add to the speci cation as many sentences that are valid properties of the concrete domain as needed rather than to add exactly those that make the intended abstract interpretation an initial model. Our approach requires no structure on the concrete domain and the abstract domain need not be a complete lattice or a lower semilattice. Instead, the associated speci cation is expected to be regular [1] in order for Maude to correctly compute the least sort of a term. Interestingly, the fact that the partial order on sorts is a lattice tends to be suÆcient for regularity in practice.

*Our work here represents only rst steps towards a full re-interpretation* of abstract interpretations in MEL. Our approach is currently restricted to

*nite number of sorts, or abstract types, and we cannot yet handle xpoint* operators. These restrictions will be subject to further investigation. The focus of this paper is on the tool aspects rather than on the theoretical aspects.

*The remainder of the paper is organized as follows. In the next section* we brie y introduce the fundamental notions and notations of membership equational logic and the Maude-system, as far as they are required for our purposes. We then show in Section 3 how abstract interpretations can be seen as membership algebras. Section 4 shows how a MEL speci cation can be associated to an abstract interpretation and how abstract types can be automatically calculated at the level of that speci cation. Section 5 nally discusses how abstract interpretations can be veri ed in the framework pro- posed in this paper.

# *2 Membership Equational Logic and Maude*

*Membership equational logic [11,1] extends many-sorted equational logic [8]* with membership assertions t : s stating that a term t belongs to a sort s. It subsumes a wide variety of speci cation formalisms, including order-sorted [7,9] and partial equational logics. Despite its generality, it still enjoys the good properties of equational logics: it is simple, eÆciently implementable, and

*admits sound and complete deduction as well as free models. In this section* we informally present membership equational logic and some of the notational conventions used, referring to [11,1,4,3] for a comprehensive exposition. We assume basic familiarity with many-sorted equational logic.

*In membership equational logic, sorts are grouped in kinds and operations* are de ned on kinds. A signature consists of a set S of sorts, a set K of kinds, a map : S ! K, and a K? K-indexed set = f w;k j (w; k) 2 K? Kg of operations. An -algebra is a -algebra A together with a subset As Ak for each k 2 K and each s 2 1(k). For any K-indexed set of variables X, T(X) is the usual (K; )-algebra of terms. The sentences of MEL generalize the conditional equations (8X) t = t0 if C of equational logics by allowing membership assertions. They are universally quanti ed Horn clauses of the form (8X) t : s if C. In both cases, the condition C is a nite set fu1 =

*v ; :::;u = v ;t : s ; :::;t : s g and t; t0 ;t ; :::t ;u ;v ; :::;u ;v*

*are terms*

*1 n n 1 1 m m 1 m 1 1 n n*

*in T(X). If n = m = 0 then the sentence is called unconditional or atomic.*

*We discuss only satisfaction of atomic sentences in this paragraph; the* general case follows easily. For an -algebra A and an assignment a: X ! A we use a?: T(X) ! A to denote the unique extension of a to a morphism of (K; )-algebras. We then say that A satis es (8X) t = t0 (or (8X) t : s) if and only if for each assignment a we also have that a?(t) = a?(t0 ) (or a?(t) 2 A ). A MEL speci cation or theory is a pair ( ; ), where is a set of -sentences, and it de nes a class of -algebras (those that satisfy it) denoted Alg( ; ).

*s*

*The MEL proof theory is derived from the standard proof theory of equa-*

*tional logic. Its distinctive characteristic is that it allows to infer the member-* ships of terms to sorts in addition to the standard equalities of terms. Given a speci cation ( ; ), there are two inference rules that facilitate that. One is a modi cation of the modus ponens rule to deduce a membership from a (conditional) sentence in once its condition has been proven. The second rule is an extensionality rule asserts that equal terms have the same sort.

*Membership:*

*` (8X) t = t0 `*

*(8X) t : s*

*` (8X) t0 : s*

*Maude [4,3] is a high-performance rewrite system in the OBJ family [10]* that supports both membership equational logic and rewriting logic. Its cur- rent version processes up to 3 million rewritings per second on standard hard- ware. We use Maude notation in this paper not only to represent abstract interpretations in membership equational logic but also to verify them. A few notational conventions are introduced next. Equations and conditional equa- tions are declared via the keywords eq and ceq, respectively; membership and conditional membership assertions are declared with mb and cmb. Operations can be declared using mix- x notation. They can also be overloaded; however, this is only syntactic sugar for appropriate conditional membership assertions. For example, \* : Pos Neg -> Neg can be replaced by cmb X \* Y : Neg if

*X : Pos and Y : Neg, where \* is de ned on appropriate kinds. Declarations*

*of the form var X : S are used to introduce variables; their scope is bounded*

*by the enclosing module, introduced by fmod ... end. Kinds need not to* be declared explicitly. They are automatically calculated as the connected components of the partial order de ned by subsort declarations.

*A typical problem of speci cation formalisms that allow order-sorting and* operator overloading is that some terms may have various correct sorts at the same time. The possible sorts of a term can be deduced using the complete deduction system of MEL, i.e., sorts(t) is the set fs 2 S j ` (8X) t : sg. A speci cation ( ; ) is called regular if and only if for each term t, sorts(t) is either empty or has a minimal element w.r.t. the subsort relation. A detailed discussion on regularity can be found in [1], together with decidability results and various syntactic criteria that imply regularity. Maude implements some of these criteria. We can thus assume that each valid speci cation is regular and that we can calculate the least sort of any term eÆciently.

# *3 Abstract Interpretations as -Algebras*

*To illustrate our approach, we describe a generalized sign/range abstraction* for the real numbers, in which both the positive and the negative numbers are split into large and small. This interpretation is a simpli ed version of a more complex one that we implemented in a program synthesis system for the statistical data analysis domain [6]. In this domain, probabilities (i.e., the interval [0; 1]) and thus the real numbers \0" and \1" are of speci c interest. In order to handle division by zero appropriately, we need to make an assumption about the real numbers. We can consider division either as a partial function and leave the set of real numbers unchanged, or as a total function, in which case we need to add a new element to the real numbers which represents unde nedness. For simplicity, we choose the second assumption; note that MEL also supports partiality, so our choice does not re ect a limitation of the framework. Let ? be the special symbol denoting \unde ned"; we use the same symbol ? for the set f?g, and let IR? denote the set of real numbers plus ?. The abstract domain then consists of eleven intervals of IR? which are partially ordered by inclusion. Each interval of real numbers also contains ?, re ecting the intuition that the result of an expression can be either a number within that interval or unde ned. This domain can, as usual, be represented by the following lattice:

*( 1; +1¸ )?*

*zzzz*

*zzzz*

*( 1; 0,]?*

*¸¸¸¸*

*¸¸¸*

*[0; +1¸)?*

*||||*

*( 1; ¸1]?*

*,,*

*[ 1; 0]¸?*

*eee*

*[0; +1,]?*

*¸¸¸*

*[+1; +1)?*

*¸¸¸*

*ccc*

*¸¸¸*

*{{{{*

*,, {{{{*

*1?*

*¸¸*

*¸¸¸¸*

*0? +1?*

*xxxxx*

*¸¸¸*

*?*

*xxxx*

*We can now provide the abstract versions of operations. The typical, concise* way to do it is by tables, as for example the following table for division 2 :

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *X=Y* | *( 1; 1)* | *( 1; 0]* | *[0; 1)* | *( 1; 1]* | *[ 1; 0]* | *[0; +1]* | *[+1; 1)* | *1* | *0* | *+1* | *?* |
| *( 1; 1)* | *( 1; 1)* | *( 1; 1)* | *( 1; 1)* | *( 1; 1)* | *( 1; 1)* | *( 1; 1)* | *( 1; 1)* | *( 1; 1)* | *?* | *( 1; 1)* | *?* |
| *( 1; 0]* | *( 1; 1)* | *[0; 1)* | *( 1; 0]* | *[0; 1)* | *[0; 1)* | *( 1; 0]* | *( 1; 0]* | *[0; 1)* | *?* | *( 1; 0]* | *?* |
| *[0; 1)* | *( 1; 1)* | *( 1; 0]* | *[0; 1)* | *( 1; 0]* | *( 1; 0]* | *[0; 1)* | *[0; 1)* | *( 1; 0]* | *?* | *[0; 1)* | *?* |
| *( 1; 1]* | *( 1; 1)* | *[0; 1)* | *( 1; 0]* | *[0; 1)* | *[+1; 1)* | *( 1; 1]* | *( 1; 0]* | *(+1; 1]* | *?* | *( 1; 1]* | *?* |
| *[ 1; 0]* | *( 1; 1)* | *[0; 1)* | *( 1; 0]* | *[0; +1]* | *[0; 1)* | *( 1; 0]* | *[ 1; 0]* | *[0; +1]* | *?* | *[ 1; 0]* | *?* |
| *[0; +1]* | *( 1; 1)* | *( 1; 0]* | *[0; 1)* | *[ 1; 0]* | *( 1; 0]* | *[0; 1)* | *[0; +1]* | *[ 1; 0]* | *?* | *[0; +1]* | *?* |
| *[+1; 1)* | *( 1; 1)* | *( 1; 0]* | *[0; 1)* | *( 1; 0]* | *( 1; 1]* | *[+1; 1)* | *[0; 1)* | *( 1; 1]* | *?* | *[+1; 1)* | *?* |
| *1* | *( 1; 1)* | *[0; 1)* | *( 1; 0]* | *[0; +1]* | *[+1; 1)* | *( 1; 1]* | *[ 1; 0]* | *+1* | *?* | *1* | *?* |
| *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *?* | *0* | *?* |
| *+1* | *( 1; 1)* | *( 1; 0]* | *[0; 1)* | *[ 1; 0]* | *( 1; 1]* | *[+1; 1)* | *[0; +1]* | *1* | *?* | *+1* | *?* |
| *?* | *?* | *?* | *?* | *?* | *?* | *?* | *?* | *?* | *?* | *?* | *?* |

*However, it is known that tables are often not suÆcient to fully describe the* intended behavior of abstract operations. For example, there is no possibility to specify that the abstract value of X=X is +1?. Hence, the abstraction becomes less precise than one may wish. We will show in the next section how MEL allows us to use membership assertions to specify such properties.

*In the context of MEL, it is more convenient to regard the lattice and* the tables associated with an abstract interpretation as an -algebra over an appropriate signature . This general algebraic view gives us the freedom to consider an abstract interpretation as \just one particular model" of an ( ; )- speci cation. Since a speci cation can have many di erent models, this allows us also to relate di erent abstractions to each other. In the remainder of this section we concentrate on showing how abstract interpretations can be seen as

*-algebras. In the next sections we will then show how a MEL speci cation*

*can naturally be associated with an intended abstract interpretation and how* we can prove properties at the level of that speci cation rather than at the level of the abstract model.

*The lattice actually gives some structure to the originally at concrete* domain of real numbers. It is now natural to associate a MEL-signature to the lattice above by providing a sort for each node in the lattice and using subsort relations to re ect the lattice structure. In Maude, this is represented as:

*sorts Real --- =^*

*Neg --- =^*

*Neg-l --- =^*

*Neg-s --- =^*

*Neg-one --- =^*

*Pos --- =^*

*Pos-l --- =^*

*Pos-s --- =^*

*Pos-one --- =^*

*Zero --- =^*

*Bottom . --- =^*

*( 1; +1)?*

*( 1; 0]?*

*( 1; 1]?*

*[ 1; 0]?*

*1?*

*[0; +1)?*

*[+1; +1)?*

*[0; +1]?*

*+ 1?*

*0?*

*?*

*2 For typographical reasons we omit the ?-subscript here.*

*subsorts Neg Pos < Real .*

*subsorts Neg-one < Neg-l Neg-s < Neg . subsorts Pos-one < Pos-s Pos-l < Pos . subsorts Zero < Neg-s Pos-s .*

*subsorts Bottom < Neg-one Zero Pos-one .*

*Since the lattice is a single connected component, all sorts belong to the same* kind k and all operations are declared on that kind, i.e., we have ; =; +; : k k ! k and : k ! k, or again using the Maude notation,

*ops \_\*\_, \_/\_, \_+\_, \_-\_ : Real Real -> Real . op -\_ : Real -> Real .*

*Now, the extended real numbers IR? can be viewed as an -algebra A as* follows. For each sort s we de ne as its carrier As the appropriate subset of IR?. For example, we de ne Ak = AReal = IR?, APos = IR+ = fx 2 IR j x

*?*

*0 or x = ?g and so on, until we get to ABottom = f?g. A's operations are*

*the usual arithmetic operations on IR?. Notice that most of the carriers are*

*in nite sets.*

*However, A is not the only interesting -algebra we can construct. Instead* of using the set of the real numbers within the respective intervals as carrier we can as well use the intervals themselves. We thus get ABottom = BBottom = f?g and APos-one = BPos-one = f+1; ?g but BPos-s = f[0; +1]; 0; +1; ?g and BPos-s = f[+1; +1]; +1; ?g and so on. Hence, B's carrier Bk is nite| in fact, it consists of exactly the same eleven elements as the lattice. B's operations are de ned via operator tables, as for example the division table.

*The main di erence between the two -algebras A and B above is obvi-* ously that B is nite. Hence, it is easy to handle in practice but it looses information about the concrete domain of real numbers, so that many useful properties of real numbers do not hold anymore. A just structures the at concrete domain but still carries all the burden of analyzing the real numbers directly. Ideally, we would want to keep both algebras, as opposed to just discard A and keep B as it often happens in practice. We next show that MEL is a framework under which we can write a ( nite) speci cation which accepts both A and B as models. This speci cation can be seen as a \ nite abstraction" of the concrete domain that can contain as many properties of the domain as needed.

# *4 Interpreting Abstract Interpretations*

*We present our technique on the example in the previous section. We start in* Maude with a module ABSTRACTION-REAL by de ning the abstract domain as a partial order via sorts and subsorts:

*fmod ABSTRACTION-REAL is*

*sorts Real Neg Pos Neg-l Pos-l Neg-s Pos-s Neg-one Zero Pos-one Bottom .*

*subsorts Neg-one < Neg-l Neg-s < Neg < Real . subsorts Zero < Neg-s Pos-s .*

*subsorts Pos-one < Pos-s Pos-l < Pos < Real . subsorts Bottom < Neg-one Zero Pos-one .*

*Note that we can declare more than one subsort in a subsorts declaration; a* sort is smaller than any other sort occurring between any subsequent < symbol and the next period. The concrete real numbers will be provided in Maude 2.0 in a module MACHINE-REAL. This module is imported by the abstract interpre- tation using the protecting MACHINE-REAL clause; the reals are abstracted via a map | | : MachineReal -> Real de ned using membership assertions:

*protecting MACHINE-REAL .*

*op |\_| : MachineReal -> Real . var R : MachineReal .*

*mb | 0 | : Zero .*

*mb | 1 | : Pos-one . mb | -1 | : Neg-one .*

*cmb | R | : Neg-l if R <= -1 .*

*cmb | R | : Neg-s if R >= -1 and R <= 0 . cmb | R | : Pos-s if R >= 0 and R <= 1 . cmb | R | : Pos-l if R >= 1 .*

*Note that | 0 |, the abstraction of 0, can be of three sorts: Zero, Neg-s and* Pos-s. However, Maude automatically calculates for each term its smallest sort, so | 0 | has the sort Zero.

*The next step is to abstract the operations de ned on the concrete domain,* in our case addition, multiplication, negation, subtraction and division. We do this by making use of operator overloading, which basically represents an elegant way to encode the tables often used to de ne abstract interpretations:

*op \_+\_ : Bottom Real -> Bottom . op \_+\_ : Real Bottom -> Bottom .*

*op \_+\_ : Neg-one Zero -> Neg-one . op \_+\_ : Zero Neg-one -> Neg-one . op \_+\_ : Zero Zero -> Zero .*

*op \_+\_ : Neg-one Pos-one -> Zero . op \_+\_ : Pos-one Neg-one -> Zero . op \_+\_ : Pos-one Zero -> Pos-one . op \_+\_ : Zero Pos-one -> Pos-one . op \_+\_ : Neg-l Neg -> Neg-l .*

*op \_+\_ : Neg Neg-l -> Neg-l .*

*op \_+\_ : Neg-s Zero -> Neg-s . op \_+\_ : Zero Neg-s -> Neg-s . op \_+\_ : Pos-s Zero -> Pos-s . op \_+\_ : Zero Pos-s -> Pos-s . op \_+\_ : Pos-l Pos -> Pos-l . op \_+\_ : Pos Pos-l -> Pos-l . op \_+\_ : Neg Neg -> Neg .*

*op \_+\_ : Neg-l Pos-s -> Neg .*

*op \_+\_ : Pos-s Neg-l -> Neg . op \_+\_ : Pos Pos -> Pos .*

*op \_+\_ : Pos-l Neg-s -> Pos . op \_+\_ : Neg-s Pos-l -> Pos . op \_+\_ : Real Real -> Real .*

*op \_\*\_ : Bottom Real -> Bottom . op \_\*\_ : Real Bottom -> Bottom .*

*op \_\*\_ : Neg-one Pos-one -> Neg-one . op \_\*\_ : Pos-one Neg-one -> Neg-one . op \_\*\_ : Real Zero -> Zero .*

*op \_\*\_ : Zero Real -> Zero .*

*op \_\*\_ : Pos-one Pos-one -> Pos-one . op \_\*\_ : Neg-one Neg-one -> Pos-one . op \_\*\_ : Neg-l Pos-l -> Neg-l .*

*op \_\*\_ : Pos-l Neg-l -> Neg-l . op \_\*\_ : Neg-s Pos-s -> Neg-s . op \_\*\_ : Pos-s Neg-s -> Neg-s . op \_\*\_ : Pos-s Pos-s -> Pos-s . op \_\*\_ : Neg-s Neg-s -> Pos-s . op \_\*\_ : Neg-l Neg-l -> Pos-l . op \_\*\_ : Pos-l Pos-l -> Pos-l . op \_\*\_ : Neg Pos -> Neg .*

*op \_\*\_ : Pos Neg -> Neg .*

*op \_\*\_ : Pos Pos -> Pos . op \_\*\_ : Neg Neg -> Pos .*

*op \_\*\_ : Real Real -> Real .*

*In this case, we are able to de ne the intended abstractions of addition* and multiplication completely by operator overloading (i.e., in table form). In practice, however, tables are often not suÆcient to properly de ne abstract versions of operators. Some important domain speci c properties of operators cannot be speci ed only looking at the abstract type of their arguments. For example, there is no way to declare that the abstract type of X/X is Pos-one

*by just analyzing the arity (or the table) of (the abstraction of) division.* Fortunately, membership equational logic allows us to declare membership as- sertions, which then can be used to infer the smallest type of an abstract term:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *op* | *-\_ :* | *Bottom -> Bottom .* | *op* | *\_/\_ :* | *Bottom Real -> Bottom .* |
| *op* | *-\_ :* | *Pos-one -> Neg-one .* | *op* | *\_/\_ :* | *Real Zero -> Bottom .* |
| *op* | *-\_ :* | *Zero -> Zero .* | *op* | *\_/\_ :* | *Neg-one Pos-one -> Neg-one .* |
| *op* | *-\_ :* | *Neg-one -> Pos-one .* | *op* | *\_/\_ :* | *Pos-one Neg-one -> Neg-one .* |
| *op* | *-\_ :* | *Pos-l -> Neg-l .* | *op* | *\_/\_ :* | *Zero Real -> Zero .* |
| *op* | *-\_ :* | *Pos-s -> Neg-s .* | *op* | *\_/\_ :* | *Pos-one Pos-one -> Pos-one .* |
| *op* | *-\_ :* | *Neg-s -> Pos-s .* | *op* | *\_/\_ :* | *Neg-one Neg-one -> Pos-one .* |
| *op* | *-\_ :* | *Neg-l -> Pos-l .* | *op* | *\_/\_ :* | *Neg-l Pos-s -> Neg-l .* |
| *op* | *-\_ :* | *Pos -> Neg .* | *op* | *\_/\_ :* | *Pos-l Neg-s -> Neg-l .* |
| *op* | *-\_ :* | *Neg -> Pos .* | *op* | *\_/\_ :* | *Neg-s Pos-l -> Neg-s .* |
| *op* | *-\_ :* | *Real -> Real .* | *op* | *\_/\_ :* | *Pos-s Neg-l -> Neg-s .* |
|  |  |  | *op* | *\_/\_ :* | *Pos-s Pos-l -> Pos-s .* |

*vars X Y : Real .*

*mb X + (- X) : Zero .*

*op \_-\_ : Real Real -> Real . eq X - Y = X + (- Y) .*

*op \_/\_ : Neg-s Neg-l -> Pos-s . op \_/\_ : Neg-l Neg-s -> Pos-l . op \_/\_ : Pos-l Pos-s -> Pos-l . op \_/\_ : Neg Pos -> Neg .*

*op \_/\_ : Pos Neg -> Neg . op \_/\_ : Neg Neg -> Pos . op \_/\_ : Pos Pos -> Pos .*

*op \_/\_ : Real Real -> Real .*

*mb X / X : Pos-one .*

*mb (- X) / X : Neg-one . endfm*

*Notice that we can use the equation \eq X - Y = X + (- Y) ." to eliminate* the abstract subtraction in a way consistent with its concrete de nition, thus reducing the size of the speci cation.

*It may be worth mentioning here that Maude provides internal algorithms* to insure the regularity of a signature and that it warned us a few times while testing the overloaded signature above that the signature was not regular. After careful examination, we discovered that our \table" was not consistent, in the sense that certain concrete expressions couldn't have a most precise abstract value. For example, we forgot to add the third declaration of \* ; then Maude issued a warning saying that the regularity check failed: indeed, the product of negative one and positive one could have any sort in the set fNeg-s,Neg-l,Neg,Realg, but this set has no minimal element. The regular- ity checks thus helped us cover all the situations. We strongly believe that there is a deep relationship between the regularity of order-sorted overloaded signatures and the existence of most concrete abstract values for abstract inter- pretations where the abstract values form a lattice, and that this relationship is worth investigating.

*The module above can now be used as a decision procedure to abstract* expressions. The following are a few examples showing how it works:

*red | 1 - 1 | . red | 2 - 2 | .*

*red | 1 | - | 1 | . red | 2 | - | 2 | . red | 1 | + | -1 | .*

*red | 2 | + | -2 | .*

*red (| -1 | + | 1 |) + | 1 | . red | -1 | + (| 1 | + | 1 |) .*

*red | -2 | / (| 3 | / | 3 |) . red (| -2 | \* | 3 |) / | 3 | .*

*The rst ve expressions are all correctly abstracted to Zero; that is, they all* reduce to terms of sort Zero, but for di erent reasons: the rst two reduce to

*| 0 | whose sort is Zero, the next two rst use the equation eq X - Y = X + (- Y) to eliminate the subtraction and then use the membership assertion mb*

*X + (- X) : Zero, and the last one rst calculates the sorts of | 1 | and |*

*-1 | and then use the declaration op + : Pos-one Neg-one -> Zero. The following is the Maude output for the other ve reductions:*

*result Real: | 2 | + | -2 |*

*result Pos-one: (| -1 | + | 1 |) + | 1 | result Pos: | -1 | + (| 1 | + | 1 |) result Neg-l: | -2 | / (| 3 | / | 3 |) result Neg: (| -2 | \* | 3 |) / | 3 |*

*The reductions above re ect the intuition that abstract interpretations loose* information. For example, since the most speci c abstract values of 2 and -2 are Pos-l and Neg-l, respectively, we cannot say anything about the abstract type of | 2 | + | -2 |, so the returned type is Real. The next two reductions show that the abstraction of addition is not associative anymore, which was an unexpected problem that we encountered in our data analysis synthesis system. Finally, the last two reductions show that other expected equational properties of the concrete domain do not hold at the abstract level either, in this case the equation eq X / (Y / Z) = (X \* Z) / Y.

# *5 Proving Properties about Abstract Interpretations*

*Since by abstraction of a concrete domain one actually looses information, a* natural and important question is \how much information is lost?". This is a diÆcult question both to ask and to answer, and probably new theoretical results are behind it; instead of pursuing theoretical research, we decided to take a pragmatic attitude and develop an environment in which one can test an abstract interpretation as a Maude module for various equational or mem- bership properties that it is expected or not to satisfy. For example, suppose that one considers the following properties (also written as a Maude module):

*fmod PROPERTIES is sorts Pos Neg Real .*

*subsorts Pos Neg < Real . op -\_ : Real -> Real .*

*ops (\_+\_) (\_-\_) (\_\*\_) (\_/\_) : Real Real -> Real . vars X Y Z : Real .*

*eq X + Y = Y + X . \*\*\*> true*

*eq X \* Y = Y \* X . \*\*\*> true*

*eq X + (Y + Z) = (X + Y) + Z . \*\*\*> false*

*eq X \* (Y \* Z) = (X \* Y) \* Z . \*\*\*> true eq X \* (Y + Z) = (X \* Y) + (X \* Z) . \*\*\*> false eq - (- X) = X . \*\*\*> true*

*eq X \* (- Y) = - (X \* Y) . \*\*\*> true*

*eq (- X) / Y = - (X / Y) . \*\*\*> true*

*eq (X / Y) / Z = X / (Y \* Z) . \*\*\*> true*

*eq X / (Y / Z) = (X \* Z) / Y . \*\*\*> false eq (X + Y) / Z = (X / Z) + (Y / Z) . \*\*\*> false*

*eq X \* X = X \* X \* X \* X . \*\*\*> true*

*mb X \* X : Pos . \*\*\*> true*

*mb ((X \* X) + (Y \* Y)) + (X \* Y) : Pos . \*\*\*> false mb (X - Y) \* (Y - X) : Neg . \*\*\*> false*

*endfm*

*All one needs to know in order to write up desired properties of an abstract* interpretation are the sorts and the operations one wants to refer to. Notice that the sorts Pos and Neg are only needed for the last three properties, so one could remove them if one was not interested in the membership properties. The fact that the main sort Real has 10 subsorts in our previous abstract interpretation is part of the abstraction, not of its desired properties w.r.t. the concrete domain. Since one may want to test the same properties against many di erent abstract interpretations of the same domain and to chose one that best ts one's purpose, one should refer to as few sorts speci c to only one abstraction as possible when writing properties.

*One can now test the properties above against the abstract interpretation* presented in the previous section using the simple command

*red verify PROPERTIES on ABSTRACTION-REAL .*

*after rst loading the module which de nes the operation verify on and its* semantics, whose implementation is brie y presented in the appendix. The ftrue, falseg comments in the module above summarize the execution of this command. In fact, for each of the properties above, Maude detected all counterexamples, i.e., possible abstract values for X, Y, Z), that violated it; there were 780 counterexamples in total and it took Maude about 30s on a 1.5GHz linux machine to nd all of them. For example, the associativity of addition does not hold when X : Neg-one, Y : Pos-one, Z : Pos-one, and the last membership assertion does not hold when X : Pos-one, Y : Pos, Z : Zero. Interestingly, notice the last equation holds for the presented abstract interpretation, even if it does not hold for the concrete domain of real numbers. The lesson that we learned from experiments with this system is that the loss of information due to an abstraction is re ected not only by lost properties of the concrete domain, but also by properties the abstraction has that do not hold in the concrete domain.

*The meta level of Maude was needed to implement the above environment,* that is, the functionality of the operation verify on . At the meta level, the modules are just terms of sort FModule, so verify on has the arity FModule FModule -> Warnings, where Warnings is a list of warnings. In our current implementation, we traverse the rst module and for each equation or mem- bership assertion, we take the second module and generate all possible abstract sorts for variables and then calculate the least sorts of the terms involved (two terms if the statement is an equation and one if it is a membership). If the statement is an equation then we output a warning if and only if the two terms

*have di erent least sorts, and if the statement is a membership assertion then* a warning is output if and only if the calculated least sort is not smaller than or equal to the sort speci ed in the membership statement. The interested reader may consult the appendix for more detail on the implementation.

*The presented tool is still in an experimentation stage. One of our major* goals was to keep it simple and exible, such that to easily adapt it for concrete uses. We can currently foresee three applications of this tool, depending on what the speci cation PROPERTIES is:

*Abstract interpretation retrieval. If it is a collection of required prop-* erties manually or automatically generated in the context of a concrete situation where an abstract interpretation is needed, then one can use our tool to automatically retrieve from a library those abstract interpretations that satisfy the requirements;

*Comparison with the concrete domain. If it is an axiomatic presenta-* tion of the concrete domain, such as a Peano speci cation of natural num- bers, then one can use the tool to understand or test how much and what kind of information a certain abstract interpretation looses and then decide whether it is acceptable or not for a speci c task;

*Abstract interpretation re nement. If it is another abstract interpreta- tion of the same concrete domain, then an empty list of warnings means that ABSTRACTION re nes this second abstraction.*

# *6 Conclusion and Future Work*

*We presented a new way to represent and verify abstract interpretations. We* abstracted a concrete domain by a nite membership equational logic speci ca- tion, which, under loose semantics, also admits as model the common abstract interpretations whose abstract values form a lattice and whose abstract oper- ations are given by a table. This exible framework allows one to add needed properties of the concrete domain that cannot be captured by tables via equa- tions and membership assertions. Then we showed how one can use Maude, an eÆcient rewriting engine that implements membership equational logic, to execute such speci cations and thus calculate the most concrete abstract type (or the least sort) of an expression. Building on the meta-capabilities of Maude, we further implemented a tool by which one can actually verify an abstract interpretation, given as a MEL speci cation, against equational or membership properties. Such a tool can be used to retrieve appropriate abstract interpretations from a library, to characterize the loss of information due to an abstraction, and to show re nements of abstract interpretations.

*We use this approach in a programming synthesis tool that generates data* analysis C programs from stochastic speci cations, in order to symbolically prove the conditions of algorithm schemas before they are applied. However, there is much interesting research to be done to prove the viability of our

*approach for a broader range of practical situations. For example, we do not* know at this stage how to represent in nite abstract interpretations, such as arbitrary integer intervals. Should the sort representation of the lattice be given up? Or should membership equational logic be extended to allow free generated algebras of sorts? What would regularity mean in such a context? Another important aspect of abstract interpretations that we didn't investi- gate is the computation of xed points. Is there any relationship between regularity of MEL signatures and xpoints?

*Acknowledgments: We thank Jos e Meseguer and Steven Eker for encour-* agements and productive discussions. We also thank Steven Eker for providing us an alpha version of Maude 2.0 which uses the membership assertions in cal- culating the least sort of a term, so we could test our programs and claims.

# *References*

*[1] Adel Bouhoula, Jean-Pierre Jouannaud, and Jos e Meseguer. Speci cation and proof in membership equational logic. Theoretical Computer Science, 236:35{ 132, 2000.*

*[2] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. ACM Trans. Programming Languages and Systems, 16(5):1512{ 1542, September 1994.*

*[3] Manuel Clavel, F. Duran, Steven Eker, Patrick Lincoln, N. Mart -Oliet, Jos e Meseguer, and J. F. Quesada. The Maude system. In P. Narendran and*

*M. Rusinowitch, editors, Proc. 10th Intl. Conf. Rewriting Techniques and Applications, volume 1631 of Lect. Notes Comp. Sci., pages 240{243, Trento, Italy, July 1999. Springer. System Description.*

*[4] Manuel Clavel, Francisco J. Duran, Steven Eker, Patrick Lincoln, Narciso Mart -Oliet, Jos e Meseguer, and Jos e F. Quesada. Maude: Speci cation and Programming in Rewriting Logic, March 1999. Maude System documentation at* [*http://maude.csl.sri.com/papers.*](http://maude.csl.sri.com/papers)

*[5] Patrick M. Cousot and Radhia Cousot. Abstract Interpretation: A Uni ed Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In Proc. 4th ACM Symp. Principles of Programming Languages, pages 238{252, Los Angeles, California, January 1977. ACM Press.*

*[6] Bernd Fischer, Johann Schumann, and Thomas Pressburger. Generating data analysis programs from statistical models (position paper). In Walid Taha, editor, Proc. Intl. Workshop Semantics Applications, and Implementation of Program Generation, volume 1924 of Lect. Notes Comp. Sci., pages 212{229, Montreal, Canada, September 2000. Springer.*

*[7] Joseph Goguen. Order sorted algebra. Technical Report 14, UCLA Computer Science Department, 1978. Semantics and Theory of Computation Series.*

*[8] Joseph Goguen and Jos e Meseguer. Completeness of many-sorted equational logic. Houston Journal of Mathematics, 11(3):307{334,* *1985.*

*[9] Joseph Goguen and Jos e Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Theoretical Computer Science, 105(2):217{273,* *1992.*

*[10] Joseph Goguen, Timothy Winkler, Jos e Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. In Joseph Goguen and Grant Malcolm, editors, Software Engineering with OBJ: algebraic speci cation in action. Kluwer, 2000.*

*[11] Jos e Meseguer. Membership algebra as a logical framework for equational speci cation. In Proceedings, WADT'97, volume 1376 of Lecture Notes in Computer Science, pages 18{61. Springer, 1998.*

*[12] David A. Plaisted. Theorem proving with abstraction. Art. Intell., 16(1):47{ 108, 1981.*

*[13] Willem Visser, S. Park, and John Penix. Using predicate abstraction to reduce object-oriented programs for model checking. In Proceedings of the 3rd ACM SIGSOFT Workshop on Formal Methods in Software Practice, August* *2000.*

*To appear.*

# *A Some Implementation Details*

*In this appendix we give a few details on how we implemented the operation verify on : FModule FModule -> Bool that veri es an abstract interpreta- tion against user de ned properties, both represented as Maude modules. The interestead reader can download all the Maude source code discussed in this paper at* [*http://ase.arc.nasa.gov/grosu/download/absint-code.html.*](http://ase.arc.nasa.gov/grosu/download/absint-code.html)

*We rst implemented some auxiliary operations on top of Maude's meta-* level. The code below is not executable; because of space limitations we replaced straightforward code by three dots:

*fmod META-AUXILIARY is including META-LEVEL .*

*...*

*sort VarDeclTuple .*

*op |\_| : VarDeclSet -> VarDeclTuple .*

*...*

*op getVarTuples : FModule VarDeclSet -> VarDeclTupleList .*

*var M : FModule . var Var : VarDecl . var Vars : VarDeclSet . eq getVarTuples(M, none) = | none | .*

*eq getVarTuples(M, Var Vars)*

*= mergeVarDeclSet(getVarsFromVar(M, Var),getVarTuples(M, Vars)) .*

*op mergeVarDeclSet : VarDeclSet VarDeclTupleList -> VarDeclTupleList . op mergeVarDecl : VarDecl VarDeclTupleList -> VarDeclTupleList .*

*var Ts : VarDeclTupleList .*

*eq mergeVarDeclSet(none, Ts) = nil . eq mergeVarDeclSet(Var Vars, Ts)*

*= mergeVarDecl(Var, Ts) mergeVarDeclSet(Vars, Ts) . eq mergeVarDecl(Var, nil) = nil .*

*eq mergeVarDecl(Var, | Vars | Ts) = | Var Vars | mergeVarDecl(Var, Ts) .*

*op getVarsFromVar : FModule VarDecl -> VarDeclSet . vars V S : Qid .*

*eq getVarsFromVar(M, var V : S .)*

*= getVarsFromNameAndSorts(V, lesserSorts(M, S)) .*

*...*

*endfm*

*Notice the use of the builtin operation lesserSorts which takes a module* and a sort name and gives a set of all smaller sorts. This set of sorts is then used to generate all possible combinations of declarations of variables, which will be subsequently used to calculate the least sorts of the terms involved in properties.

*The next module implements the main operation. It imports both the* auxiliary operations and a module de ning warnings which we skipped; the warnings are currently some adhoc messages concatenated by @.

*fmod VERIFY-TYPE is including META-AUXILIARY + WARNING . op verify\_on\_ : FModule FModule -> Warnings .*

*vars ABS, PROP : FModule . eq verify PROP on ABS =*

*verifyEqns(getName(ABS), getVarTuples(ABS, getVars(PROP)), getEqns(PROP))*

*@ verifyMbs(getName(ABS), getVarTuples(ABS, getVars(PROP)), getMbs(PROP)) .*

*op verifyEqns : Qid VarDeclTupleList EquationSet -> Warnings . op verifyEqn : Qid VarDeclTupleList Equation -> Warnings .*

*var Name : Qid . var Ts : VarDeclTupleList . var Vars : VarDeclSet . var Eqn : Equation . var Eqns : EquationSet .*

*eq verifyEqns(Name, Ts, none) = ok . eq verifyEqns(Name, Ts, Eqn Eqns) =*

*verifyEqn(Name, Ts, Eqn) @ verifyEqns(Name, Ts, Eqns) . eq verifyEqn(Name, nil, Eqn) = finishedEqn(Eqn) .*

*eq verifyEqn(Name, | Vars | Ts, Eqn) =*

*(if leastSort(buildModule(Name, Vars), leftHandSide(Eqn)) == leastSort(buildModule(Name, Vars), rightHandSide(Eqn))*

*then ok else errorEqn?(Vars, Eqn) fi) @ verifyEqn(Name, Ts, Eqn) .*

*op verifyMbs : Qid VarDeclTupleList MembAxSet -> Warnings . op verifyMb : Qid VarDeclTupleList MembAx -> Warnings .*

*var Mb : MembAx . var Mbs : MembAxSet . eq verifyMbs(Name, Ts, none) = ok .*

*eq verifyMbs(Name, Ts, Mb Mbs) =*

*verifyMb(Name, Ts, Mb) @ verifyMbs(Name, Ts, Mbs) . eq verifyMb(Name, nil, Mb) = finishedMb(Mb) .*

*eq verifyMb(Name, | Vars | Ts, Mb) =*

*(if sortLeq(buildModule(Name, Vars),*

*leastSort(buildModule(Name, Vars), leftHandSide(Mb)), rightHandSide(Mb))*

*then ok else errorMb?(Vars, Mb) fi) @ verifyMb(Name, Ts, Mb) .*

*endfm*

*All equations and membership assertions are thoroughly tested on the speci-*

*cation representing the abstract interpretation. In the case of equations, the* least sorts of both terms are calculated (using the builtin operation leastSort) and if they are di erent, a warning is issued. In the case of membership as- sertions, the least sort of the declared term is calculated and then compared with the declared sort (using the builtin operation sortLeq); if it is not smaller then a warning is generated.