

Specifying the Semantics of Maude

Ken Keefe

Final Project for CS 576, Spring 2005

1 Introduction to Maude

Maude is a system that allows its users to create and use membership equational logic and rewriting logic specifications. Maude has seen many applications including program verification, theorem proving, and some real-time applications. An equational theory is typically specified in Maude as a module.

```
fmod NATURAL is
  sort Natural .
  op 0 : -> Natural .
  op s_ : Natural -> Natural .
  op _+_ : Natural Natural -> Natural .
  vars N M : Natural .
  eq N + 0 = N .
  eq N + s M = s(N + M) .
endfm
```

Figure 1: A typical Maude module that specifies the natural numbers and addition over the natural numbers.

In figure 1, we see an example of a Maude module. We can break this up into its parts, which will be useful for our discussion later. The first line inside the module is a sort declaration. Each module has a set of sorts and an implicitly defined set of kinds. The next three lines are the operators in the module, also called the signature. An `op` statement basically defines the type of a function and, when mixfix notation is used, the syntax also. The next line simply defines a set of variables and their types that will be used in the definition of the module's equations. The last two lines are the set of equations that define, in this module, the properties of addition in Peano notation.

2 Encoding the Abstract Syntax Tree

The first step in my work on Maude involved specifying the abstract syntax tree of the Maude language. The Maude manual provided a pretty formal grammar specification. I was able to mostly adhere to that specification, straying from it occasionally to avoid some specification problems introduced by syntactic sugar. For example, the grammar included the following two lines:

```
op <OpForm> : <Type>* <Arrow> <Type> [<Attr>] .
ops {<OpId> | ( <OpForm> ) }+ : <Type>* <Arrow> <Type> [<Attr>] .
```

The `ops` command allows you to specify several operators that have the same type at the same time, for example: “`ops _+_ *_ : Nat Nat -> Nat .`”. While this is syntactically nice, it doesn't need to be included in our specification since it is semantically equivalent to a set of `op` commands.

I have specified all the parts of the grammar that are necessary to write a Maude module. I have omitted the specification of top-level commands because it is unclear whether or not they will be necessary in my future work.

3 Mathematical Foundations

The fundamental type of Maude is the sort. Every sort has a unique kind. When a sort is declared in a Maude module, its kind is implicitly declared right alongside of it. When a sort is involved in a subsort relation, the sort's kind and all the kinds of the other sorts involved in the relation are merged into one kind. It is clear that a sort will always have a unique kind.

The documentation that I have used in my work [1] [2] [3] [4] has demonstrated that the rewriting and reasoning that is used in Maude always works at the kind level. For this reason, my specification has focused on kinds as the fundamental type of the mathematics of Maude. In my specification, I have defined a sort to be simply a string and a kind to be a set of sorts.

Thus far, I have defined in my specification the following concepts.

- Sorts, kinds, and types and how they are related.
- Operators and a basic signature.
- Many kinded signatures.
- Order sorted signatures.
- Σ terms.
- Σ algebras.
- Σ homomorphisms.
- The initial term algebra.

4 Conclusion and Future Work

It has been a significant struggle to get my sources to agree on a common ground. Some sources seemed to say that reasoning from sorts and then inferring about kinds was the way to go and others pushed the other direction. Once I had decided the correct direction to head, many of the pieces of this specification fell into place.

A lot of work still lies ahead. Some of the next concepts to be worked on are many kinded substitutions, many kinded equation, many kinded rewrite rules, termination, confluence, and sort orders. I plan to continue to work on this project over the summer and in the coming Fall.

References

- [1] Adel Bouhoula, Jean-Pierre Jouannaud, and José Meseguer. *Specification and Proof in Membership Equational Logic*. Theoretical Computer Science, Vol. 236, pg. 35-132, 2000.
- [2] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *Maude Manual (Version 2.1)*. <http://maude.cs.uiuc.edu/>, March 2004.
- [3] Theodore McCombs. *Maude 2.0 Primer*. <http://maude.cs.uiuc.edu/>, August 2003.
- [4] José Meseguer. *Program Verification*. Slides for lectures of CS 476. <http://www-courses.cs.uiuc.edu/cs476/>, Spring 2005.