Foundations of Systems Development

Martin Wirsing

in cooperation with
Axel Rauschmayer

WS 05/06

Goals

- Get to know algebraic specifications
- Write first specifications with Maude
- Order-sorted signatures and specifications

Algebraic Specifications

Definition:

Let $\Sigma = (S, E)$ be a signature and E a set of (closed) Σ-formulas.

- $SP=<\Sigma,E>$ is called an algebraic specification.
- If E is a set of equations, SP is called an equational specification.

Moreover, depending on the semantics we distinguish loose specification and initial algebra specifications:

- The semantics of a loose specification SP is given by the class of all models of SP:

  $Mod(SP) =_{def} \{ A \in Alg(SP) \mid A \models E \}$
- The semantics of an initial algebra specification SP is given by all initial models of SP:

  $I(SP) =_{def} \{A \in Mod(SP) \mid A \text{ is initial in } Mod(SP)\}$

Maude

- Maude is an executable specification language for equational specifications and term rewriting.
- Maude is being developed by Jose Meseguer and his group at Univ. of Illinois and by the group of Carolyn Talcott at SRI.
- You can download Maude 2.0 from the Maude web page http://maude.cs.uiuc.edu. Chapter 2 in the Maude 2.0 manual (also in that web page) explains how you start Maude and interact with it.

Maude Functional Modules and Theories

In Maude,

- A **loose specification** is called **theory,** declared with the syntax
  th <name> is (Σ,E) endth
  Maude theories are not executable!

- An **initial specification** is called **functional module**, declared with syntax
  fmod <name> is (Σ,E) endfm

Maude Theories

- The **trivial theory** consisting of one sort

```
fth TRIV is
  sort Elt .
endfth
```

- The theory of **partial orderings**

```
fth PARTIAL-ORDER is
  protecting BOOL .
  including TRIV .
  op _<=_ : Elt Elt -> Bool .
  vars X Y Z : Elt .
  ceq X <= Z = true if X <= Y and Y <= Z [nonexec label transitive] .
  ceq X = Y if X <= Y /\ Y <= X [nonexec label antisymmetric] .
  eq  X <= X = true [nonexec label reflexive] .
endfth
```

Natural Numbers (prefix syntax)

```
fmod NAT-PREFIX is
   sort Natural .

   op 0 : -> Natural .
   op s : Natural -> Natural .
   op plus : Natural Natural -> Natural .

   vars N M : Natural .

   eq plus(N,0) = N .
   eq plus(N,s(M)) = s(plus(N,M)) .
endfm

Maude> red plus(s(s(0)),s(s(0))) .
reduce in NAT-PREFIX : plus(s(s(0)), s(s(0))) rewrites: 3 in
-10ms cpu (0ms real) (~rewrites/second)
result Natural: s(s(s(s(0))))
Maude>
```


Natural Numbers (mixfix syntax)

```
fmod NAT-MIXFIX is
   sort Natural .

   op 0 : -> Natural .
   op s_ : Natural -> Natural .
   op _+_ : Natural Natural -> Natural .
   op _*_ : Natural Natural -> Natural .

   vars N M : Natural .

   eq N + 0 = N .
   eq N + s M = s(N + M) .
   eq N * 0 = 0 .
   eq N * s M = N + (N * M) .
endfm

Maude> red (s s 0) + (s s 0) .
reduce in NAT-MIXFIX : s s 0 + s s 0
rewrites: 3 in 0ms cpu (0ms real) (~ rewrites/second)
result Natural: s s s s 0
```

Lists of Natural Numbers

```
fmod NAT-LIST is
   protecting NAT-MIXFIX .
   sort List .

   op nil : -> List .
   op _._ : Natural List -> List .
   op length : List -> Natural .

   var N : Natural .
   var L : List .

   eq length(nil) = 0 .
   eq length(N . L) = s length(L) .
endfm

Maude> red length(0 . (s 0 . (s s 0 . (0 . nil)))) .
reduce in NAT-LIST : length(0 . s 0 . s s 0 . 0 . nil)
rewrites: 5 in 0ms cpu (0ms real) (~ rewrites/second)
result Natural: s s s s 0
```


Specifying partial functions in total algebras?

Problem

- How to specify partial functions in a framework of algebras with total functions?
- Consider for example defining a function
  - `first` that takes the first element of a list of natural numbers, or
  - a predecessor function `p` that assigns to each natural number its predecessor.

  What can we do? If we define,

  ```
  op first : List -> Natural .
  op p_ : Natural -> Natural .
  ```

  we have then the awkward problem of having to define the values of `first(nil)` and of `p 0`, which in fact are undefined.

Some Common Mistakes

- not ending declarations for sorts, operators, etc. with a space followed by a period, e.g.,

```
sort Natural
op 0 : -> Natural .
op s : Natural -> Natural
```

- not putting enough parentheses to disambiguate expressions, e.g., `p s s 0 + 0`
- not leaving spaces between a mixfix operator and its arguments, e.g., `0+0`

Order-sorted signatures

Solution:

Recognize that these functions are partial, but become total on **appropriate subsorts**

`NeList < List` of nonempty lists, and

`NzNatural < Natural` of nonzero natural numbers.

If we define,

```
op s_ : Natural -> NzNatural .
op _._ : Natural List -> NeList .
op first : NeList -> Natural .
op p_ : NzNatural -> Natural .
```

everything is fine.

Subsorts also allow us to **overload operator symbols**. For example,

`Natural < Integer` , and

```
op _+_ : Natural Natural -> Natural
op _+_ : Integer Integer -> Integer
```

Order-sorted Natural Numbers

```
fmod NATURAL is
   sorts Natural NzNatural .
   subsorts NzNatural < Natural .
   op 0 : -> Natural .
   op s_ : Natural -> NzNatural .
   op p_ : NzNatural -> Natural .
   op _+_ : Natural Natural -> Natural .

   vars N M : Natural .

   eq p s N = N .
   eq N + 0 = N .
   eq N + s M = s(N + M) .
endfm

Maude> red p((s s 0) + (s s 0)) .
   reduce in NATURAL : p (s s 0 + s s 0)
   rewrites: 4 in 0ms cpu (0ms real) (~ rewrites/second)
   result NzNatural: s s s 0
```


Order-sorted Lists

```
fmod NAT-LIST-II is
   protecting NATURAL .
   sorts NeList List .
   subsorts NeList < List .

   op nil : -> List .
   op _._ : Natural List -> NeList .
   op length : List -> Natural .
   op first : NeList -> Natural .

   var N : Natural .
   var L : List .

   eq length(nil) = 0 .
   eq length(N . L) = s length(L) .
   eq first(N . L) = N .
endfm
```

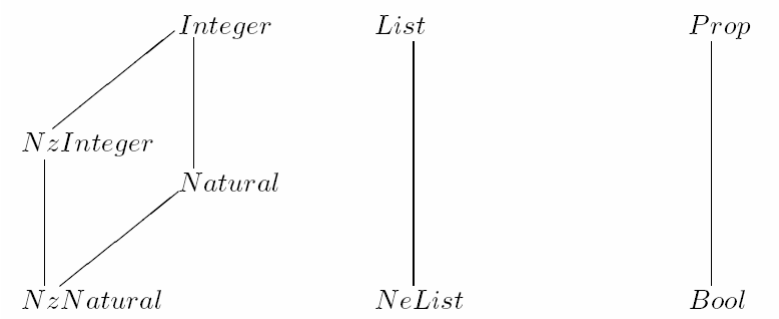
Order-sorted Signature (mathematically)

- An order-sorted signature Σ is a triple
  $\Sigma = ((S, \Sigma_{w,s})_{(w,s) \in S^* \times S}, <)$,
  where $((S, \Sigma_{w,s})_{(w,s) \in S^* \times S}, <)$ is an S-sorted signature, and where $<$ is a partial order relation on S called subsort inclusion. That is, $<$ is a binary relation on S which is:
  - irreflexive: $\neg(x<x)$ and
  - transitive: $x<y$ and $y<z$ imply $x<z$
- Of course, any such relation $<$ has an associated
  $\leq$ relation that is reflexive, antisymmetric, and transitive,
  and we will move back and forth between $<$ and $\leq$.
- **Note:** Unless specified otherwise, by a signature in Maude we will always mean an order-sorted signature.

Connected Components

- Given a signature Σ, we can define an equivalence relation
  $\equiv_{\leq}$ between sorts $s, s' \in S$
  as the smallest relation such that:
  - if $s \leq s'$ or $s' \leq s$ then $s \equiv_{\leq} s'$
  - if $s \equiv_{\leq} s'$ and $s' \equiv_{\leq} s''$ then $s \equiv_{\leq} s''$
- We call the equivalence classes modulo $\equiv_{\leq}$ the **connected components** of the poset order $(S, \leq)$.
- Intuitively, when we view the poset as a directed acyclic graph, they are the connected components of the graph.

Connected Components Example



$S/\equiv_{\leq}$ =
{ Integer, NzInteger, Natural, NzNatural}, {NeList, List}, {Bool, Prop} }

Subsort vs. Ad-hoc Overloading

- In general, the same operator name may have different declarations in the same signature Σ.
  For example, in the NATURAL module we have,

```
op _+_ : Natural Natural -> Natural .
op _+_ : NzNatural NzNatural -> NzNatural .
```

- When we have two operator declarations,
  f : w → s, and f : w′ → s′ ,
  with w and w′ strings of equal length, then:
  (1) if $w \equiv_{\leq} w'$ and $s \equiv_{\leq} s'$, we call them **subsort overloaded**;
  (2) otherwise, e.g, + for Natural and for "exclusive or" in Bool, we call them **ad-hoc overloaded**.

Order-Sorted Algebras

Given an order-sorted signature $\Sigma = (S, \{\Sigma_{w,s}\}_{(w,s)\in S^* \times S}, <)$ an order-sorted Σ-algebra is defined as a many-sorted $(S, \{\Sigma_{w,s}\}_{(w,s)\in S^* \times S})$-algebra A such that:

- In $A = \{A_s\}_{s\in S}$, if $s < s'$ then $A_s \subseteq A_{s'}$
- if f is subsort overloaded, so that we have, $f : w \longrightarrow s$, and $f : w' \longrightarrow s'$, with w and w' strings of equal length, and with $w \equiv_{\leq} w'$ and $s \equiv_{\leq} s'$, then:
  - if $w = w' = nil$, then f is a constant and we have $f_A^{nil,s} = f_A^{nil,s'}$ (subsort overloaded constants coincide)
  - otherwise, if $(a_1, \ldots, a_n) \in A^w \cap A^{w'}$, then $f_A^{w,s}(a_1, \ldots, a_n) = f_A^{w',s'}(a_1, \ldots, a_n)$ (subsort overloaded operations agree)

Summary

- Maude is an executable language for equational specifications.
- Loose specifications are called theories, initial algebra specifications are called functional modules.
- In Maude partial functions are modelled by total functions on subsorts.
- Subsort overloading vs. ad-hoc overloading of functions.