Formal Methods: Lecture 2

José Meseguer

Computer Science Department
University of Illinois at Urbana-Champaign

Equational Theories

Theories in equational logic are called equational theories. In Computer Science they are sometimes referred to as algebraic specifications.

An equational theory is a pair (Σ, E), where:

- Σ, called the signature, describes the syntax of the theory, that is, what types of data and what operation symbols (function symbols) are involved;

- E is a set of equations between expressions (called terms) in the syntax of Σ.

Unsorted, Many-Sorted, and Order-Sorted Signatures

Our syntax Σ can be more or less expressive, depending on how many types (called sorts) of data it allows, and what relationships between types it supports:

- unsorted (or single-sorted) signatures have only one sort, and operation symbols on it;
- many-sorted signatures allow different sorts, such as `Integer`, `Bool`, `List`, etc., and operation symbols relating these sorts;
- order-sorted signatures are many-sorted signatures that, in addition, allow inclusion relations between sorts, such as `Natural < Integer`.

Maude Functional Modules

Maude functional modules are equational theories (Σ, E), declared with syntax

$$\texttt{fmod}\ (\Sigma, E)\ \texttt{endfm}$$

Such theories can be unsorted, many-sorted, or order-sorted, or even more general membership equational theories (to be discussed later in the course).

In what follows we will see examples of unsorted, many-sorted and order-sorted equational theories (Σ, E) expressed as Maude functional modules, and of how one can use such theories as functional programs by computing with the equations E.

Unsorted Functional Modules

```
*** 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>
```

Unsorted Functional Modules (II)

```
fmod NAT-MIXFIX is                          *** mixfix syntax
  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
Maude>
```

Many-Sorted Functional Modules

```
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
Maude>
```

Many-Sorted Signatures

The full signature Σ of the NAT-LIST example, that imports NAT-MIXFIX, is then,

```
sorts Natural List .
op 0 : -> Natural .
op s_ : Natural -> Natural .
op _+_ : Natural Natural -> Natural .
op _*_ : Natural Natural -> Natural .
op nil : -> List .
op _._ : Natural List -> List .
op length : List -> Natural .
```

Indexed Families of Sets

We will use the notion of an indexed family of sets. Intuitively, this is a family of sets indexed by some indices ranging over a given set I of indices.

If the indexed set $I = \mathbb{N}^+$ is the set of nonzero natural numbers, and $\mathbf{B} = \{0, 1\}$, we can define the "family of sets of Boolean vectors of increasing length" as the $\mathbb{N}^+$-indexed family $\{\mathbf{B}^n\}_{n \in \mathbb{N}^+}$, which we could envision as the sequence of sets,

$$\mathbf{B}^1 = \mathbf{B},\ \mathbf{B}^2,\ \mathbf{B}^3,\ \ldots,\ \mathbf{B}^n,\ \ldots$$

Similarly, if we define for each $n \in \mathbb{N}^+$ the set $[n] = \{1, \ldots, n\}$ of the first n naturals, the "family of all initial segments of the positive naturals" is the $\mathbb{N}^+$-indexed family $\{[n]\}_{n \in \mathbb{N}^+}$.

Indexed Families of Sets (II)

More precisely, given a set I, an I-indexed family of sets is just a function

$$X : I \longrightarrow \mathcal{U}$$

from I to a set of sets $\mathcal{U}$, typically called a universe. It is well known (Russell's paradox) that there is no such thing as "the set of all sets," but if we put restrictions on the size of the sets allowed as elements, there are many legitimate sets $\mathcal{U}$ that can be used as universes.

Then, for the function $X : I \longrightarrow \mathcal{U}$ we use the notation $X = \{X_i\}_{i \in I}$, where, by definition, $X_i = X(i)$.

Indexed Families of Sets (III)

The needed universe (set of sets) $\mathcal{U}$ does not have to be very big. For example, for the indexed family $\{[n]\}_{n\in\mathbb{N}^+}$ it is enough to take the powerset $\mathcal{U} = \mathcal{P}(\mathbb{N}^+)$, and then we can view the indexed set as a function,

$$[\_] : \mathbb{N}^+ \longrightarrow \mathcal{P}(\mathbb{N}^+).$$

Similarly, since $\mathbf{B}^*$ is the set of bit vectors of arbitrary length, the indexed family $\{\mathbf{B}^n\}_{n\in\mathbb{N}^+}$ is the function,

$$\mathbf{B}^- : \mathbb{N}^+ \longrightarrow \mathcal{P}(\mathbf{B}^*).$$

In general, we can always view $X = \{X_i\}_{i\in I}$ as the function, $X\_ : I \longrightarrow \mathcal{P}(\bigcup\{X_i \mid i \in I\})$.

Many-Sorted Signatures Mathematically

An S-sorted signature is a pair $\Sigma = (S, \{\Sigma_{w,s}\}_{(w,s)\in S^* \times S})$, where S is called the set of sorts, and $\{\Sigma_{w,s}\}_{(w,s)\in S^* \times S}$, is an $S^* \times S$-indexed family of sets of operation symbols.

We denote an operation symbol $f \in \Sigma_{w,s}$ by, $f : w \longrightarrow s$.

For the signature Σ in our `NAT-LIST` example, the operator declaration, say,

```
op _._ : Natural List -> List .
```

is an element of the set $\Sigma_{\texttt{Natural List},\texttt{List}}$.

Many-Sorted Signatures Mathematically (II)

In full detail, the signature Σ in our NAT-LIST example has: set of sorts $S = \{\texttt{Natural}, \texttt{List}\}$, and indexed family of sets of operation symbols:

$\Sigma_{nil,\texttt{Natural}} = \{\texttt{0}\}$, $\Sigma_{nil,\texttt{List}} = \{\texttt{nil}\}$,
$\Sigma_{\texttt{Natural},\texttt{Natural}} = \{\texttt{s\_}\}$,
$\Sigma_{\texttt{Natural Natural},\texttt{Natural}} = \{\texttt{\_+\_}, \texttt{\_*\_}\}$,
$\Sigma_{\texttt{Natural List},\texttt{List}} = \{\texttt{\_·\_}\}$, $\Sigma_{\texttt{List},\texttt{Natural}} = \{\texttt{length}\}$, and all other $\Sigma_{w,s} = \emptyset$.

Similarly, the signature Σ in our NAT-PREFIX example has:

$S = \{\texttt{Natural}\}$, $\Sigma_{nil,\texttt{Natural}} = \{\texttt{0}\}$, $\Sigma_{\texttt{Natural},\texttt{Natural}} = \{\texttt{s}\}$,
$\Sigma_{\texttt{Natural Natural},\texttt{Natural}} = \{\texttt{plus}\}$, and all other
$\Sigma_{w,\texttt{Natural}} = \emptyset$.

The Need for Order-Sorted Signatures

Many-sorted signatures are still too restrictive. The problem is that some operations are partial, and there is no natural way of defining them in just a many-sorted framework.

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 defining the values of `first(nil)` and of `p 0`, which in fact are undefined.

The Need for Order-Sorted Signatures (II)

A much better solution is to recognize that these functions are partial with the typing just given, 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 Functional Modules

```
fmod NATURAL is
  sorts Natural NzNatural .
  subsorts NzNatural < Natural .
  op 0 : -> Natural .
  op s_ : Natural -> NzNatural .
  op p_ : NzNatural -> Natural .
  op _+_ : Natural Natural -> Natural .
  op _+_ : NzNatural NzNatural -> NzNatural .
  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 Functional Modules (II)

```
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 Signatures 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 that is:

- *irreflexive*: $\neg(x < x)$
- *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 we will always mean an order-sorted signature.

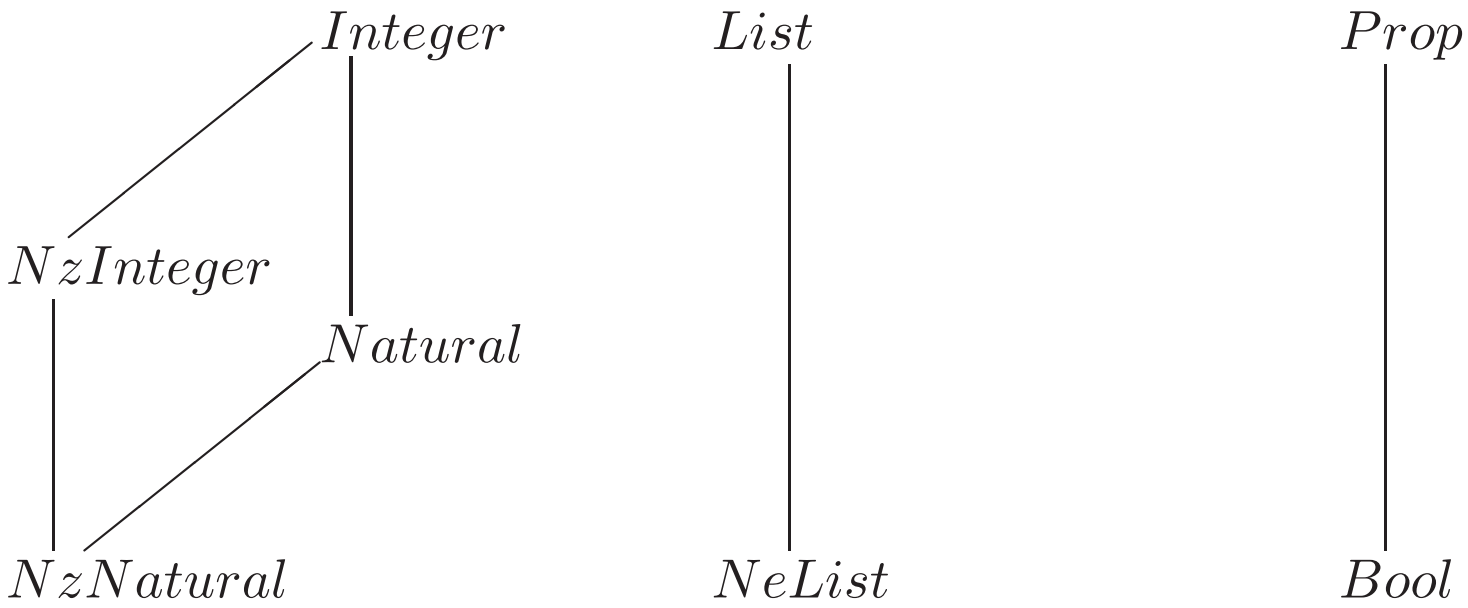
Connected Components of the Poset of Sorts

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



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

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, $f : w \longrightarrow s$, and $f : w' \longrightarrow s'$, 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.

Many-Sorted Algebras

A many-sorted signature $\Sigma = (S, \{\Sigma_{w,s}\}_{(w,s)\in S^* \times S})$ provides the syntax to talk about models of a certain kind. Each such model is called a Σ-algebra and is defined by:

- an S-indexed family of sets $A = \{A_s\}_{s\in S}$
- for each $a : nil \longrightarrow s$ in Σ, an element $a_A^{nil,s} \in A_s$
- for each $f : w \longrightarrow s$ in Σ, with $w = s_1 \ldots s_n$, $n > 0$, a function $f_A^{w,s} : A_{s_1} \times \ldots \times A_{s_n} \longrightarrow A_s$

Notation: if $w = s_1 \ldots s_n$, we write $A^w = A_{s_1} \times \ldots \times A_{s_n}$.

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)

Term Algebras

An obvious example of an order-sorted Σ-algebra is the term algebra T_Σ, where the family $T_\Sigma = \{T_{\Sigma,s}\}_{s \in S}$ and its operations are mutually defined by:

- for each $a : nil \longrightarrow s$ in Σ, $a_{T_\Sigma} = a \in T_{\Sigma,s}$
- for each $f : w \longrightarrow s$ in Σ, with $w = s_1 \ldots s_n$, $n > 0$, the function $f_{T_\Sigma} : T_{\Sigma,s_1} \times \ldots \times T_{\Sigma,s_n} \longrightarrow T_{\Sigma,s}$ maps the tuple $(t_1, \ldots, t_n) \in T_\Sigma^w$ to the expression (called a term) $f(t_1, \ldots, t_n) \in T_{\Sigma,s}$
- if $s < s'$, then $T_{\Sigma,s} \subseteq T_{\Sigma,s'}$

Examples of Terms for the NATURAL Specification

$T_{\texttt{NATURAL,NzNatural}} =$
$\{\texttt{s 0}, \texttt{s s 0}, \texttt{s s s 0}, \texttt{s p s 0}, \texttt{s(0 + s 0)}, \ldots\}$

$T_{\texttt{NATURAL,Natural}} =$
$T_{\texttt{NATURAL,NzNatural}} \cup \{\texttt{0}, \texttt{p s 0}, (0+0), \ldots\}.$

Although the mathematical definition of terms uses prefix notation, Maude allows general mixfix notation. This is just a (very useful) parsing and pretty-printing facility. If one insists (by giving the command `set print mixfix off .`) Maude can print even mixfix terms in prefix notation. For example, `s_(_+_(0,s_(0)))` instead of `s(0 + s 0)`.

Other Examples of Algebras

Check that:

- the natural numbers are, in somewhat different ways, models of the signatures `NAT-PREFIX`, `NAT-MIXFIX`, and `NATURAL`; in fact we shall see that they are the standard model for such specifications when equations are taken into account.

- for each natural number $n > 0$, the congruence classes modulo n provide an algebra for the signatures of `NAT-PREFIX` and `NAT-MIXFIX` (hint: define the successor function appropriately).

Variables

Note that in our definition of Σ-terms we only allowed constants and terms built up from them by other operation symbols, so-called ground terms. Therefore, terms with variables, such as those appearing in the equations

```
vars N M : Natural .
eq N + 0 = N .
eq N + s M = s(N + M) .
```

do not seem to fall within our definition. What can we say about such terms? First, note that `N` and `M` are variables in the mathematical sense, not at all in the sense of variables in an imperative language. Second, we can reduce the notion of terms with variables to that of terms without variables (ground terms) in an extended signature.

A Sample Extended Signature

We can extend the signature of our above example by adding the variables as additional constants to get the new signature,

```
sort Natural .
op 0 : -> Natural .
op N : -> Natural .
op M : -> Natural .
op s_ : Natural -> Natural .
op _+_ : Natural Natural -> Natural .
```

in which a term such as `s(N + M)` is now a well-defined term of sort `Natural`.

The Extended Signature $\Sigma(X)$

The general way of extending a signature with variables is as follows. We assume a family $X = \{X_s\}_{s \in S}$ of sets of variables for the different sorts $s \in S$ in the signature Σ. Such that:

- variables of different sorts are different, i.e., $X_s \cap X_{s'} = \emptyset$ if $s \neq s'$
- the variables in X are different from the constants in Σ, i.e., $(\cup_{s \in S} X_s) \cap (\cup_{s \in S} \Sigma_{nil,s}) = \emptyset$.

Then we define the extended signature $\Sigma(X)$ as follows: (1) $\Sigma(X)_{nil,s} = \Sigma_{nil,s} \cup X_s$, and (2) $\Sigma(X)_{w,s} = \Sigma_{w,s}$ for $w \neq nil$.

The Term Algebra $T_{\Sigma(X)}$

Therefore, Σ-terms with variables in X are the elements of the term algebra $T_{\Sigma(X)}$ associated to the extended signature $\Sigma(X)$.

Getting to Use Maude

You should begin with functional modules with syntax as exemplified in the above examples. You should write such modules in files using Emacs.

You can download Maude 2.2 from the Maude web page `http://maude.cs.uiuc.edu`. Chapter 2 in the Maude 2.2 manual (also in that web page) explains how you start Maude and interact with it.

To enter a module into Maude can use cut and paste, or the "in *filename*" command inside Maude, and can change or list directories using Unix commands.

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`

Readings and Exercises

Before the next lecture try to:

- read the rest of chapter 4 of the Maude manual (functional modules) and up to page 65 (Section 2.4 included) of Ölvecky's Notes.

- begin playing with Maude. Try to define your own version of a `LIST` module with a function that reverses a list, e.g., $rev(a\ b\ c) = c\ b\ a$.

Readings and Exercises (II)

Ex.2.1. Denote by $C_1, \ldots C_n, \ldots$ the different connected components of the poset of sorts in a signature Σ. Also, given a sort s, denote by $C(s)$ its connected component. Then, given a Σ-algebra A and a connected component C define $A_C = \cup_{s \in C} A_s$.

Show that, given an operator $f : s_1 \ldots s_n \longrightarrow s$ in Σ, all the other subsort-overloaded operators related to it "glue together" into a single partial function $f_A : A_{C(s_1)} \times \ldots \times A_{C(s_n)} \longrightarrow A_{C(s)}$.

Therefore, the qualifications $f_A^{w,s}$ are in some sense unnecessary.

Readings and Exercises (III)

Ex.2.2. Let Σ be the signature:

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

And let $A = \{a, b, c\}$. How many different Σ-algebra structures can be defined on the set A (explain, and also state the total number of such algebras)? Can you justify why the number comes out that way? For example, can your supposed justification predict (without having to explicitly construct them) exactly how many such algebras will there be on A if we add to the above Σ a binary function, say,

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