José Meseguer

Computer Science Department
University of Illinois at Urbana-Champaign
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 \((\Sigma, E)\), where:

- \(\Sigma\), 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 \(\Sigma\).
Our syntax $\Sigma$ 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 are equational theories \((\Sigma, E)\), declared with syntax

\[
\text{fmod } (\Sigma, E) \text{ endfm}
\]

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 \((\Sigma, E)\) 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 [ctor] .
   op s : Natural -> Natural [ctor] .
   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

sort Natural .

op 0 : -> Natural [ctor] .

op s_ : Natural -> Natural [ctor] .

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>
fmod NAT-LIST is
  protecting NAT-MIXFIX .
  sort List .
  op nil : -> List [ctor] .
  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 $\Sigma$ 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.
```

We can naturally represent a many-sorted signature as a labeled multigraphs whose nodes are the sorts, and whose labeled edges are the operation symbols.

In a normal labeled graph a directed edge links an input node to an output node. Instead, in a multigraph an edge links zero, one, or several input nodes to an output node. So, we view an operator like

\[
\text{op} \_\_ : \text{Natural List} \rightarrow \text{List}.
\]

as a labeled edge having two input nodes and one output node (see Picture 2.1). When all operations are unary, signatures are exactly labeled graphs (see Picture 2.2)
An many-sorted signature is a pair $\Sigma = (S, F)$, with:

- $S$ a set whose elements $s, s', s'', \ldots \in S$ are called sorts, and

- $F$, called the set of function symbols, is an $S^* \times S$-indexed set $F = \{F_w, (w, s) \in S^* \times S\}$, where if $f \in F_{s_1 \ldots s_n, s}$ then we display it as $f : s_1 \ldots s_n \to s$ and call sequence of sorts $s_1 \ldots s_n \in S^*$ the argument sorts, and $s \in S$ the result sort. When $n = 0$, we call $f \in F_{\text{nil}, s}$, with nil the empty sequence, a constant.
In full detail, the signature $\Sigma$ in our NAT-LIST example has:
set of sorts $S = \{\text{Natural}, \text{List}\}$, and indexed family $F$ of
sets of function symbols:

$$
F_{\text{nil, Natural}} = \{0\}, \quad F_{\text{nil, List}} = \{\text{nil}\}, \quad F_{\text{Natural, Natural}} = \{s\}, \quad F_{\text{Natural, Natural, Natural}} = \{\_+-, \_\ast\_\}, \quad F_{\text{Natural, List, List}} = \{\_\_\_\}, \quad F_{\text{List, Natural}} = \{\text{length}\}.
$$

Similarly, the signature $\Sigma$ in our NAT-PREFIX example has $S = \{\text{Natural}\}$ an indexed family $G$ of sets of function symbols:

$$
G_{\text{nil, Natural}} = \{0\}, \quad G_{\text{Natural, Natural}} = \{s\}, \quad G_{\text{Natural, Natural, Natural}} = \{\text{plus}\}.
$$
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,

\[
\begin{align*}
\text{op first} & : \text{List} \rightarrow \text{Natural} . \\
\text{op p} & : \text{Natural} \rightarrow \text{Natural} . 
\end{align*}
\]

we have then the awkward problem of defining the values of \( \text{first}(\text{nil}) \) and of \( p \ 0 \), which in fact are undefined.
A much better solution is to recognize that these functions are \textit{partial} with the typing just given, but \textit{become total} on appropriate \textit{subsorts} NeList < List of nonempty lists, and NzNatural < Natural of nonzero natural numbers. If we define,

\begin{verbatim}
  op s_ : Natural -> NzNatural .
  op _._ : Natural List -> NeList .
  op first : NeList -> Natural .
  op p_ : NzNatural -> Natural .
\end{verbatim}

everything is fine. Subsorts also allow us to \textit{overload} operator symbols. For example, Natural < Integer, and

\begin{verbatim}
  op _+_ : Natural Natural -> Natural .
  op _+_ : Integer Integer -> Integer .
\end{verbatim}
fmod NATURAL is
  sorts Natural NzNatural .
  subsorts NzNatural < Natural .
  op 0 : -> Natural [ctor] .
  op s_ : Natural -> NzNatural [ctor] .
  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
fmod NAT-LIST-II is
    protecting NATURAL .
sorts NeList List .
subsorts NeList < List .
op nil : -> List [ctor] .
op length : List -> Natural .
op first : NeList -> Natural .
op rest : NeList -> List .
var N : Natural .
var L : List .
eq length(nil) = 0 .
eq length(N . L) = s length(L) .
eq first(N . L) = N .
eq rest(N . L) = L .
endfm
An order-sorted signature \( \Sigma \) is a pair \( \Sigma = ((S, <), F) \) where 
\( (S, F) \) is a many-sorted signature, and where \( < \) is a partial 
order relation on the set \( S \) of sorts called subsort inclusion.

That is, \( < \) is a binary relation on \( S \) that is:

- **irreflexive**: \( \neg (x < x) \)

- **transitive**: \( x < y \) and \( y < z \) imply \( x < z \)

Any such relation \( < \) has an associated \( \leq \) relation that is
reflexive, antisymmetric, and transitive. We will move back 
and forth between \( < \) and \( \leq \) (see **STACS 7.4**).

**Note**: Unless specified otherwise, by a **signature** we will 
always mean an **order-sorted signature**.
Given a signature $\Sigma$, we can define an equivalence relation (see *STACS* 7.6) $\equiv_\leq$ 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 (see *STACS* 7.6, Exercise 68).
Connected Components Example

\[
S/ \equiv \leq = \\{\{\text{NzNatural, Natural, NzInteger, Integer}\}, \{\text{Nelist, List}\}, \{\text{Bool, Prop}\}\}
\]
In general, the same operator name may have different declarations in the same signature $\Sigma$. For example, in the NATURAL module we have,

\[
\begin{align*}
\text{op } \_\_+\_ & \text{ : Natural Natural } \rightarrow \text{ Natural } . \\
\text{op } \_\_+\_ & \text{ : NzNatural NzNatural } \rightarrow \text{ NzNatural } . 
\end{align*}
\]

When we have two operator declarations, $f : w \rightarrow s$, and $f : w' \rightarrow s'$, with $w$ and $w'$ strings of equal length, then: (1) if $w \equiv w'$ and $s \equiv s'$, we call them subsort overloaded; (2) otherwise, e.g, $\_\_+$ for Natural and for exclusive or in Bool, we call them ad-hoc overloaded.
Since an order-sorted signature is a many-sorted signature whose set of nodes is a poset, we can describe them graphically as labeled multigraphs whose set of nodes is a poset.

We can picture subsort inclusions as usual for partial orders, and operators, as before, as labeled edges in the multigraph. For example, the order-sorted signature of the module NAT-LIST-II is depicted this way in Picture 2.3
Ex.2.1. Define in Maude the following functions on the naturals:

- \( > \) and \( \geq \) as Boolean-valued binary functions, either importing the built-in module \( \text{BOOL} \) with single sort \( \text{Bool} \), or, perhaps better, defining your own version of the Booleans (in that case, give it a different name, e.g., \( \text{BOOLEAN} \), to avoid clashes with \( \text{BOOL} \)),

- \textbf{max} and \textbf{min}, that yield the maximum, resp. minimum, of two numbers,

- \textbf{even} and \textbf{odd} as Boolean-valued functions on the naturals,

- \textbf{factorial}, the factorial function.
Ex. 2.1. Define in Maude the following functions on list of natural numbers:

- `append` and `reverse`, which appends two lists, resp. reverses the list,
- `max` and `min` that computes the biggest (resp. smallest) number in the list,
- `get.even`, which extracts the lists of even numbers of a list,
- `odd.even`, which, given a list, produces a pair of lists: the first the sublist of its odd numbers and the second the sublist of its even numbers.