Program Verification: Lecture 6

José Meseguer

University of Illinois at Urbana-Champaign (USA)
A functional module `fmod (Σ, E) endfm` with subsignature `Ω ⊆ Σ` satisfying: (1) **Unique termination**, (2) **Sufficient Completeness** and (1) **Sort Preservation**, and, also, `(∀ t ∈ T_Ω) t!_E = t`, has a canonical term algebra `C_{Σ/E}` as its semantics.

Conditions (1)–(3), plus requirement `(∀ t ∈ T_Ω) t!_E = t`, can best be understood by noting that the `red` command mapping `t ∈ T_Σ` to `t!_E ∈ T_Ω` is just rewriting `t` to termination with the rules `E`.

But a functional module can have **axioms** `B`. That is, it can be of the form `fmod (Σ, E ∪ B) endfm`. Then, the `red` command simplifies `t ∈ T_Σ` to `t!_{E/B} ∈ T_Ω` with the rules `E` modulo `B`.

What **executability conditions** ensure that `C_{Σ/E,B}` exists for `fmod (Σ, E ∪ B) endfm` in general? They will be executability requirements on the rewrite theory `(Σ, B, E)`. 
Executability Conditions (II)

In terms of the rewrite theory \((\Sigma, B, \bar{E})\),

1. **Unique Termination** will follow from \(\bar{E}\) being:
   - terminating modulo \(B\), and
   - confluent modulo \(B\)

2. **Sufficiently Completeness** will follow from \(\bar{E}\) being so modulo \(B\), and

3. **Sort Preservation** will follow from \(\bar{E}\) being sort decreasing.

The requirement \((\forall t \in T_{\Omega}) t!_{E/B} = t\) will not be needed: it was just a simplifying assumption. And we will make explicit an implicit assumption on variables in the rules \(\bar{E}\) essential for executability.

Under the above executability requirements we will then define the canonical term algebra \(C_{\Sigma/E,B}\) of a functional module \(\text{fmod} (\Sigma, E \cup B)\) with constructors \(\Omega \subseteq \Sigma\) in full generality.
No Extra Variables in Righthand Sides

Consider the rule $0 \rightarrow x \ast 0$. This rule is problematic: we have to guess how to instantiate the variable $x$ in $x \ast 0$ before applying it, and there is an infinite number of instantiations for $x$.

Instead, the rule $x \ast 0 \rightarrow 0$ can be applied without problems, since the same substitution obtained by matching for the lefthand side can be reused to generate the righthand side replacement.

Therefore, for any functional module $fmod (\Sigma, E \cup B) endfm$ and associated rewrite theory $(\Sigma, B, \vec{E})$ we will require:

For each $t \rightarrow t' \in \vec{E}$, any variable $x$ occurring in $t'$ must also occur in $t$, i.e., $vars(t') \subseteq vars(t)$. 
Sort Decreasingness

Another important requirement on \((\Sigma, B, \vec{E})\) is:

\[(SD) \text{ Sort-decreasingness: For each } t \rightarrow t' \in \vec{E}, s \in S, \text{ and substitution } \theta \text{ we have } t\theta : s \Rightarrow t'\theta : s.\]

where \(t : s\) abbreviates \(t \in T_{\Sigma,s}\). Prove by well-founded induction on the context \(C\) below which a rewrite \(C[t\theta] \rightarrow_R C[t'\theta]\) takes place, that under condition (SD), if \(u \rightarrow_R v\), then \(u : s \Rightarrow v : s\).

To see why without sort-decreasingness things can go wrong, let \(\Sigma\) have sorts \(C\) and \(D\) with \(C < D\), a constant \(c\) of sort \(C\), a constant \(d\) of sort \(D\), and a subsort-overloaded unary function \(f : C \rightarrow C, f : D \rightarrow D\). Let \(B = \emptyset\) and \(R = \{c \rightarrow d, f(f(x : C)) \rightarrow f(x : C)\}\). With the second rule \(f(f(c))\) rewrites to \(f(c)\), and then to \(f(d)\) with the first rule. But if we apply the first rule to \(f(f(c))\) we get \(f(f(d))\), which cannot be further rewritten because sort information has been lost!
Checking Sort-Decreasingness

Sort decreasingness can be easily checked, since we do not need to check it on the (infinite) set of all substitutions \( \theta \). If \( \{x_1 : s_1, \ldots, x_n : s_n\} = \text{vars}(t \rightarrow t') \), we only need to check it on the finite set of substitutions of the form
\[
\{x_1 : s_1 \mapsto x'_1 : s'_1, \ldots, x_n : s_n \mapsto x'_n : s'_n\}, \quad s'_i \leq s_i, \quad 1 \leq i \leq n,
\]
called the sort specializations of the variables \( \{x_1 : s_1, \ldots, x_n : s_n\} \).

For example, for sorts \( \text{Nat} < \text{Set} \), with \( \_ \cup \_ \) set union, the rule \( x \rightarrow x \cup x \), with \( x : \text{Set} \), is not sort-decreasing, since for the sort specialization \( \{x : \text{Set} \mapsto x' : \text{Nat}\} \) we have
\[
ls(x') = \text{Nat} < \text{Set} = ls(x' \cup x').
\]

**Exercise.** For \( \Sigma \) preregular, prove that the rules \( \vec{E} \) are sort decreasing iff for each sort specialization \( \rho \) and for each \( t \rightarrow t' \) in \( \vec{E} \) we have: \( ls(t\rho) \geq ls(t'\rho) \).
**B-Preregular Signatures**

Recall that if \( \Sigma \) is preregular each term \( t \) has a least sort \( ls(t) \). For axioms \( B \) we want the strongest property: that \( \Sigma \) is \( B \)-preregular, i.e., (1) \( \Sigma \) is preregular, and (2) \( t =_B t' \) implies \( ls(t) = ls(t') \).

How can we check that \( \Sigma \) is \( B \)-preregular? Very easily. All axioms \( B \) we shall consider are regular, i.e., such that for each \( (u = v) \in B \) we have \( vars(u) = vars(v) \). Now consider the following:

**Theorem**

For \( \Sigma \) preregular and \( B \) a set of regular \( \Sigma \)-axioms, \( \Sigma \) is \( B \)-preregular iff the rewrite theory \((\Sigma, \rightarrow \uparrow B \cup \leftarrow B)\) is sort decreasing.

The theorem follows easily from the properties of sort-decreasing rules stated in previous slides. Furthermore, it can be effectively checked for each \( (u = v) \in B \) using its sort specializations. Maude automatically checks that \( \Sigma \) is \( B \)-preregular and gives a warning if the property fails.
The Algebra $\mathbb{T}_{\Sigma/B}$

For $\Sigma B$-preregular we can easily define the algebra $\mathbb{T}_{\Sigma/B}$, whose elements are $B$-equivalence classes $[t]_B$ of terms modulo $=_B$, i.e., $t' \in [t]_B \iff t =_B t'$. Specifically, $\mathbb{T}_{\Sigma/B} = (T_{\Sigma/B}, -T_{\Sigma/B})$, where, abbreviating $[t]_B$ to just $[t]$, we define:

- $T_{\Sigma/B} = \{ T_{\Sigma,s/=B} \}_{s \in S}$, with $T_{\Sigma,s/=B}$ written $T_{\Sigma/B,s}$.
- For $a \mapsto s$ in $\Sigma$, $a_{T_{\Sigma/B}} = [a] \in T_{\Sigma/B,s}$.
- For $f : s_1 \ldots s_n \to s$ in $\Sigma$, $f_{T_{\Sigma/B}} : T_{\Sigma/B,s_1} \times \ldots \times T_{\Sigma/B,s_n} \ni ([t_1], \ldots, [t_n]) \mapsto [f(t_1, \ldots, t_n)] \in T_{\Sigma/B,s}$.

Note that the definition of $f_{T_{\Sigma/B}}$ does not depend on the choice of the $t_1, \ldots, t_n$, since if $t_i =_B t'_i$, $1 \leq i \leq n$, then we have:

$f(t_1, \ldots, t_n) =_B f(t'_1, \ldots, t'_n)$, since we can build a proof from those of $t_i =_B t'_i$, $1 \leq i \leq n$. 
Determinism

Another requirement on \((\Sigma, B, \vec{E})\) is determinism: if a term \(t\) is simplified by \(\vec{E}\) modulo \(B\) to two different terms \(u\) and \(v\), and \(u \neq_B v\), then \(u\) and \(v\) can always be further simplified by \(\vec{E}\) modulo \(B\) to a common term \(w\).

This implies (Exercise!) that if \(t \rightarrow^*_E u\) and \(t \rightarrow^*_E v\), and \(u\) and \(v\) cannot be further simplified by \(\vec{E}\) modulo \(B\), then we must have \(u =_B v\). This is the idea of determinism: if rewriting with \(\vec{E}\) modulo \(B\) yields a fully simplified answer, then that answer must be unique modulo \(B\).

That is, the final result of rewriting a term \(t\) with the rules \(\vec{E}\) modulo \(B\) should not depend on the particular order in which the rewrites have been performed.
Determinism = Confluence

Determinism is captured by confluence. The rules \( \vec{E} \) of \((\Sigma, B, \vec{E})\) are confluent modulo \( B \) iff for each \( t \in \bigcup T_{\Sigma(Y)} \), whenever \( t \xrightarrow{\vec{E}/B} u, t \xrightarrow{\vec{E}/B} v \), there is a \( w \in \bigcup T_{\Sigma(Y)} \) such that \( u \xrightarrow{\vec{E}/B} w \) and \( v \xrightarrow{\vec{E}/B} w \). This can be described diagrammatically (dashed arrows denote existential quantification):

We call \( \vec{E} \) ground confluent modulo \( B \) if the above is only required for \( t \in \bigcup T_{\Sigma} \).
Termination

Another requirement on \((\Sigma, B, \vec{E})\) is termination modulo \(B\):

**Definition**

For the rewrite theory \((\Sigma, B, \vec{E})\), rules \(\vec{E}\) are called terminating modulo \(B\) iff \(\rightarrow_{\vec{E}/B}\) is well-founded. \(\vec{E}\) is called weakly terminating modulo \(B\) iff any \(t \in \bigcup \mathcal{T}_\Sigma(\mathcal{Y})\) has a \(\vec{E}/B\)-normal form, i.e., \(\exists v \in \bigcup \mathcal{T}_\Sigma(\mathcal{Y})\) s.t. \(t \rightarrow^*_{\vec{E}/B} v \land \forall w \in \bigcup \mathcal{T}_\Sigma(\mathcal{Y})\) s.t. \(v \rightarrow_{\vec{E}/B} w\).

**Notation:** \(t \rightarrow^!_{\vec{E}/B} v\).

If \((\Sigma, B, \vec{E})\) is confluent and terminating modulo \(B\), each \(t \in \mathcal{T}_\Sigma\) reduces to an \(\vec{E}/B\)-normal form \(t!_{E/B}\), i.e., \(t \rightarrow!_{\vec{E}/B} t!_{E/B}\), and \(t!_{E/B}\) is unique modulo \(B\). Furthermore, if \(\Sigma\) is \(B\)-preregular, and \(\vec{E}\) is sort-decreasing, both **Unique Termination** and **Sort Preservation** hold, and we have an \(S\)-sorted function:

\[!_{E/B} : \mathcal{T}_\Sigma \ni t \mapsto [t!_{E/B}] \in \mathcal{T}_{\Sigma/B}\]
Joinability and the Church-Rosser Property

Call two terms \( t, t' \in \bigcup T_{\Sigma(Y)} \) joinable with \( \vec{E} \) modulo \( B \), denoted \( t \downarrow_{\vec{E}/B} t' \), iff \( (\exists w \in \bigcup T_{\Sigma(Y)}) \ t \rightarrow^*_{\vec{E}/B} w \land t' \rightarrow^*_{\vec{E}/B} w \).

Exercise. Prove that if \( (\Sigma, E \cup B) \) is an order-sorted equational theory whose rules \( \vec{E} \) are confluent modulo \( B \), then the following equivalence, called the Church-Rosser property, holds for any two terms \( t, t' \in T_{\Sigma(Y)} \):

\[
(\dagger) \quad t =_{E \cup B} t' \iff t \downarrow_{\vec{E}/B} t'.
\]

Prove that if \( \vec{E} \) is also terminating modulo \( B \) we also have:

\[
(\ddagger) \quad t =_{E \cup B} t' \iff t!_{E/B} =_B t!'_{E/B}
\]

Since \( =_B \) (with \( B, A, C, U \) axioms) is decidable, we can decide \( t =_{E \cup B} t' \) by deciding \( t!_{E/B} =_B t!'_{E/B} \), which we can do in Maude by typing: \( \text{red } t = t' \). \((\dagger)\) reduces equational deduction to rewriting, and \((\ddagger)\) makes it decidable.
Subsignatures and Constructor Subsignatures

Before defining sufficient completeness we make more precise the notions of subsignature and constructor subsignature.

**Definition**

An order-sorted signature $\Sigma' = ((S', <'), G), \Sigma'$ is called a **subsignature** of an order-sorted signature $\Sigma = ((S, <), F, \Sigma)$, denoted $\Sigma' \subseteq \Sigma$, iff:

1. $S' \subseteq S$, $<')\subseteq<$, and $G \subseteq F$.
2. $\Sigma' \subseteq \Sigma$, i.e., for each $(f' : w' \rightarrow s') \in \Sigma'$ we have $(f' : w' \rightarrow s') \in \Sigma$.

If $S' = S$ and $<')=<$ we say that $\Sigma' \subseteq \Sigma$ on the same sort poset.

In a functional module $\texttt{fmod } (\Sigma, E \cup B) \texttt{ endfm}$, the ctor declaration defines a subsignature $\Omega \subseteq \Sigma$ on the same sort poset $(S, <)$, called the **constructor subsignature**.
Sufficient Completeness Defined

Definition

Let the rewrite theory \((\Sigma, B, \vec{E})\) be terminating, and \(\Omega \subseteq \Sigma\) a subsignature inclusion, where \(\Omega\) has the same poset of sorts as \(\Sigma\). We call the rules \(\vec{E}\) are sufficiently complete modulo \(B\) with respect to the constructor subsignature \(\Omega\) iff for each \(t \in T_\Sigma\) and each \(\vec{E}/B\)-normal form of \(t\), i.e., each \(u \in T_\Sigma\) s.t. \(t \rightarrow! \vec{E}/B u\), we have \(u \in T_\Omega\).
More on Sufficient Completeness

If $\Sigma^\square$ is kind-complete, then the above requirement that for each $t \in T_\Sigma$, if $t \rightarrow!_{\vec{E}/B} u$ then $u \in T_\Omega$ should apply only to $t \in T_{\Sigma,s}$ with $s \in S$ in the original set of sorts, before adding the “kind” $[s]$ on top of each connected component $[s]$ and lifting operators to kinds. I.e., the sufficient completeness for $\vec{E}$ modulo $B$ should be required only for terms in the original signature $\Sigma$ before kind-completing it to $\Sigma^\square$.

**Example.** For sorts $Nat$ and $NzNat$ with $Nat < NzNat$, and constructors $0 : \rightarrow Nat$ and $s : Nat \rightarrow NzNat$, the predecessor function $p : NzNat \rightarrow Nat$ defined by the equation $p(s(x)) = x$ is sufficiently complete. But the term $p(0)$ of kind $[Nat]$ is in normal form, yet is not a constructor term.
More on Sufficient Completeness (II)

If \((\Sigma, B, \vec{E})\) has \(\Omega \subseteq \Sigma\) as a constructor subsignature with \(\vec{E}\) terminating modulo \(B\), we say that the constructors \(\Omega\) are free modulo \(B\) in \((\Sigma, B, \vec{E})\) iff for each sort \(s\) which is not a kind and each \(u \in T_{\Omega,s}\) we have \(u = u!_{E/B}\). That is, each \(u \in T_{\Omega,s}\) is in \(\vec{E}, B\)-normal form.

Example. Multisets of natural numbers, with \(\text{Nat} < \text{MSet}\), and constructors \(\emptyset : \rightarrow \text{MSet}\) and \(\_, \_ : \text{MSet} \times \text{MSet} \rightarrow \text{MSet}\) and axioms \(ACU\) for \(\_, \_\) are free modulo \(ACU\). But Sets of natural numbers, obtained by adding the equation \(n, n = n\), where \(n\) has sort \(\text{Nat}\) are not free modulo \(ACU\). For example, the set \(0, 0, s(0)\) is not in \(\vec{E}, B\)-normal form, since \((0, 0, s(0))!_{E/ACU} = 0, s(0)\).
The Canonical Term Algebra

Let \( f_{\text{mod}}(\Sigma, E \cup B) \) enfm have \( \Sigma B \)-preregular and constructor subsignature \( \Omega \subseteq \Sigma \); and let \( (\Sigma, B, \vec{E}) \) be sort-decreasing, confluent, terminating and sufficiently complete modulo \( B \) (w.r.t. \( \Omega \)). Then, the semantics of \( f_{\text{mod}}(\Sigma, E \cup B) \) enfm is defined by its canonical term algebra \( C_{\Sigma/E,B} = (C_{\Sigma/E,B}, C_{\Sigma/E,B}) \), where:

- for each \( s \in S \), \( C_{\Sigma/E,B,s} = \{[u] \in T_{\Omega/B,s} \mid u = !_{E/B}u\} \)
- For \( a : \rightarrow s \) in \( \Sigma \), \( a_{C_{\Sigma/E,B}} = [a!_{E/B}] \in C_{\Sigma/E,B,s} \).
- For \( f : s_1 \ldots s_n \rightarrow s \) in \( \Sigma \),
  \[
  f_{C_{\Sigma/E,B}} : C_{\Sigma/E,B,s_1} \times \ldots \times C_{\Sigma/E,B,s_n} \ni ([t_1], \ldots, [t_n]) \mapsto [f(t_1, \ldots, t_n)!_{E/B}] \in C_{\Sigma/E,B,s}.
  \]

Confluence and termination imply \textbf{Unique Termination.}

\textbf{Sufficient Completeness} is guaranteed. Sort-decreasingness and \( B \)-preregularity imply \textbf{Sort Preservation}. \( C_{\Sigma/E,B} \) now allows: (i) axioms \( B \), and (ii) constructors that need not be free modulo \( B \).
Consider the following:

- A signature $\Omega$ of constructors with sorts $Nat$ and $Set$, subsort $Nat < Set$, and constructors $0 : \rightarrow Nat$, $s : Nat \rightarrow Nat$, $\emptyset : \rightarrow Set$ and $\_ , \_ : Set Set \rightarrow Set$ and axioms $B = ACU$ for $\_ , \_$. 
- $\Sigma$ adds to $\Omega$ the function symbol $+1 : Set \rightarrow Set$. 
- $E = \{(n, n) = n, +1(\emptyset) = \emptyset, +1(n, S) = s(n), +1(S)\}$, where $n$ has sort $Nat$ and $S$ has sort $Set$.

Then, up to the slight change of representation $n_1, \ldots, n_k$ versus $\{n_1, \ldots, n_k\}$, $C_{\Sigma/E,B}$ is the algebra with sorts $Nat$, resp. $Set$, interpreted as $N$, resp. $P_{fin}(N)$, set union function, denoted $\_ , \_ \_ \Sigma/E,B$, and the function $+1 \_ \_ \_ \_ \Sigma/E,B$ increases by 1 each set element.
Examples of Sufficient Completeness Modulo $B$

For example, consider the reverse function in the list module

\[
\text{fmod MY-LIST is protecting NAT .}
\]
\[\begin{array}{l}
\text{sorts NeList List .} \\
\text{subsorts Nat < NeList < List .} \\
\text{op _;_ : List List -> List [assoc] .} \\
\text{op _;_ : NeList NeList -> NeList [assoc ctor] .} \\
\text{op nil : -> List [ctor] .} \\
\text{op rev : List -> List .} \\
\text{eq rev(nil) = nil .} \\
\text{eq rev(N:Nat ; L:List) = rev(L:List) ; N:Nat .} \\
\end{array}
\]
endfm

Are nil and _;_ (plus 0 and s) really the constructors of this module as claimed?
Examples of Sufficient Completeness Modulo $B$ (II)

The answer is that they are **not**, as witnessed by:

Maude> red rev(7) .
reduce in MY-LIST : rev(7) .
rewrites: 0 in 0ms cpu (0ms real) (~ rewrites/second)
result List: rev(7)

The problem is that the above two equations would have been sufficient if we had also declared the `id: nil` attribute for `_;_` but do not fully define `rev` if only the `assoc` attribute is used.

In future lectures we shall see how sufficient completeness can be automatically checked under reasonable assumptions.
Examples of Sufficient Completeness Modulo $B$ (III)

So, suppose we add an extra equation for `rev`

```plaintext
fmod MY-LIST is protecting NAT .
  sorts NeList List .
  subsorts Nat < NeList < List .
  op nil : -> List [ctor] .
  op rev : List -> List .
  eq rev(nil) = nil .
  eq rev(N:Nat ; L:List) = rev(L:List) ; N:Nat .
endfm
```

Is now this module sufficiently complete?
Examples of Sufficient Completeness Modulo $B$ (IV)

Indeed we now have

Maude> red rev(7) .
reduce in MY-LIS

But it is still not sufficiently complete, since

Maude> red nil ; 7 .
reduce in MY-LIST : nil ; 7 .
result List: nil ; 7

is not a constructor term, since _;_; is a constructor on NeList but a defined function on List.
Examples of Sufficient Completeness Modulo $B$ $(V)$

The really sufficiently complete specification, making the constructors free modulo assoc, is

```plaintext
fmod MY-LIST is protecting NAT . sorts NeList List .
  subsorts Nat < NeList < List .
  op nil : -> List [ctor] .
  op rev : List -> List .
  eq rev(nil) = nil .
  eq rev(N:Nat ; L:List) = rev(L:List) ; N:Nat .
  eq nil ; L:List = L:List .
  eq L:List ; nil = L:List .
endfm

Maude> red nil ; 7 .
reduce in MY-LIST : nil ; 7 .
result NzNat: 7
```
The following example shows an equational theory whose constructors are not free.

```plaintext
fmod NAT/3 is
  sorts Nat .
  op 0 : -> Nat [ctor] .
  op s : Nat -> Nat [ctor] .
  op _+_ : Nat Nat -> Nat .
  vars N M : Nat .
  eq N + 0 = N .
  eq N + s(M) = s(N + M) .
  eq s(s(s(0))) = 0 .
endfm
```