ISR: Lecture 3

José Meseguer

University of Illinois at Urbana-Champaign (USA)

Executability Conditions

Given a rewrite theory (Σ, B, R), what executabilty conditions should be placed on the rules R to effectively use it for equational simplification modulo B in the equational theory $(\Sigma, B \cup eq(R))$, in which the rules $t \rightarrow t' \in R$ are now understood as equations $t = t' \in eq(R)$?

We will see that there are essentially four conditions needed:

1. each $t \rightarrow t' \in R$ shoud be such that $vars(t') \subseteq vars(t)$
2. R is sort-decreasing
3. R is confluent modulo B
4. R is terminating modulo B (highly desirable but not essential)

and will consider some variants of such conditions.

No Extra Variables in Lefthand Sides

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

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

Therefore, we should require:

> *(1) for each $t \to t' \in R$, any variable x occuring in t' must also occur in t.*

Sort Decreasingness

A second important requirement is:

> *(2) sort-decreasingness: for each $t \to t' \in R$, sort $s \in S$, and substitution θ we should have $t\theta : s \Rightarrow t'\theta : s$.*

Prove by well-founded induction on the context C below which a rewrite $C[t\theta] \to_R C[t'\theta]$ takes place, that under condition (2), if $u \to_R v$, then $u : s \Rightarrow v : s$.

To see why without sort-decreasingness things can go wrong, let Σ 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 $f : C \longrightarrow C$, $f : D \longrightarrow D$. Let $B = \varnothing$ and $R = \{c \to d, f(f(x : C)) \to f(x : C)\}$. 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 θ. If $\{x_1 : s_1, \ldots, x_n : s_n\} = vars(t \to t')$, we only need to check it on the (typically finite) set of substitutions of the form $\{(x_1 : s_1, x'_1 : s'_1), \ldots, (x_n : s_n, x'_n : s'_n)\}$ with $s'_i \leq s_i$, $1 \leq i \leq n$, called the sort specializations of the variables $\{x_1 : s_1, \ldots, x_n : s_n\}$.

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

Exercise. For Σ preregular (i.e., each $t \in T_\Sigma$ has a least sort $ls(t) \in S$), prove that R is sort decreasing iff for each sort specialization ρ and for each $t \to t'$ in R we have: $ls(t\rho) \geq ls(t'\rho)$.

Determinism

A third requirement is determinism: if a term t is simplified by R modulo B to two different terms u and v, and $u \neq_B v$, then u and v can always be further simplified by R modulo B to a common term w.

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

That is, the final result of a reduction with the rules R modulo B should not depend on the particular order in which the rewrites have been performed.

Determinism = Confluence

Determinism is captured by: (3) confluence. The rules R of (Σ, B, R) are confluent modulo B iff for each $t \in T_{\Sigma(Y)}$, whenever $t \rightarrow^*_{R/B} u$, $t \rightarrow^*_{R/B} v$, there is a $w \in T_{\Sigma(Y)}$ such that $u \rightarrow^*_{R/B} w$ and $v \rightarrow^*_{R/B} w$. This can be described diagrammatically (dashed arrows denote existential quantification):

$$\begin{tikzcd} & t \arrow[dl, "R/B"', "*"] \arrow[dr, "*", "R/B"'] & \\ u \arrow[dr, dotted, "*"', "R/B"] & & v \arrow[dl, dotted, "*", "R/B"'] \\ & w & \end{tikzcd}$$

We call R (3′) ground confluent modulo B if the above is only required for $t \in T_{\Sigma}$.

Joinability and the Church-Rosser Property

Call two terms $t, t' \in \bigcup T_{\Sigma(Y)}$ joinable with R modulo B, denoted $t \downarrow_{R/B} t'$, iff $(\exists w \in T_{\Sigma(Y)})\ t \to^{*}_{R/B} w \ \wedge \ t' \to^{*}_{R/B} w$.

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

$$t =_{E \cup B} t' \ \Leftrightarrow \ t \downarrow_{E/B} t'.$$

where we abbreviate $t \downarrow_{\vec{E}/B} t'$ to just $t \downarrow_{E/B} t'$.

Termination

It is highly desirable that rewriting with R modulo B terminates.

Definition

Let (Σ, B, R) be a rewrite theory. R is called terminating or strongly normalizing modulo B iff $\rightarrow_{R/B}$ is well-founded. R is called weakly terminating or normalizing modulo B iff any $t \in T_{\Sigma(Y)}$ has a R/B-normal form, i.e., $\exists v \in T_{\Sigma(Y)}$ s.t. $t \rightarrow^{*}_{R/B} v \wedge \ \nexists w \in T_{\Sigma(Y)}$ s.t. $v \rightarrow_{R/B} w$.
(**Notation:** $t \rightarrow^{!}_{R/B} v$).

Therefore, a highly desirable fourth requirement is:

> *(4) the rules R are terminating modulo B, or at least the weaker requirement $(4')$ that the rules R are (ground) weakly terminating modulo B.*

Conditions on the Axioms B

Even with requirements (1)–(4) all satisfied, some further requirements should be placed on axioms B so that they can be effectively "built in."

- There shoud be a B-matching algorith, that is, and algorithm such that, given Σ-terms t and t', gives us a complete set of substitutions θ such that $t\theta =_B t'$, or fails if no such θ exists. If $t\theta =_B t'$ holds, we say that t' B-matches the pattern t.
- The variables in the axioms B should all be at the kind level, i.e., of the form $x : \top_{[s]}$, for $[s]$ a kind in $(S, <)$, so that the equations B apply in their fullest possible generality.
- The equations B should be B-preregular, in the sense that, given a B-equivalence class $[t]_B$, the set $\{s \in S \mid t' \in [t]_B \wedge t' : s\}$ has a minimum element, denoted $ls([t]_B)$.
  (Maude automatically checks B-preregularity for $B \subseteq ACU$).

The Canonical Form of a Term

Suppose $(\Sigma, E \uplus B)$ is oriented as the rewrite theory $(\Sigma, B, \vec{E})$ and satisfies the executability conditions (1)–(4), or at least the slightly weaker (1)–(2), and (3')–(4').

Then, every term $t \in T_\Sigma$ can be simplified to a unique normal form $can_{E/B}(t)$ modulo B, called its canonical form, so that $t \rightarrow!_{E/B} can_{E/B}(t)$.

Furthermore, by the Church-Rosser property we have the following extremely useful equivalence for any $t, t' \in T_\Sigma$ (resp. $t, t' \in T_{\Sigma(Y)}$ if $(\Sigma, B, \vec{E})$ is confluent):

$$t =_{E \uplus B} t' \quad \Leftrightarrow \quad t \downarrow_{E/B} t' \quad \Leftrightarrow \quad can_{E/B}(t) =_B can_{E/B}(t').$$

Therefore, to know if t, t' are provably equal in $(\Sigma, E \uplus B)$, reduce them to canonical form and test if $can_{E/B}(t) =_B can_{E/B}(t')$, which is decidable if B has a B-matching algorithm.

The Terms in Canonical Form

This suggests considering the terms in E/B-canonical form as an S-indexed family of sets $Can_{\Sigma/E,B}$.

Consider the example of an unsorted signature Σ with a constant 0, a unary successor function s, and a binary addition function $\_ + \_$, and the equations: $E = \{x + 0 = x, x + s(y) = s(x + y)\}$.

It is easy to check that the term rewriting system $(\Sigma, \vec{E})$ is confluent and terminating. It is also easy to check that the set of ground terms in $\vec{E}$-canonical form is the set $Can_{\Sigma/E} = \{0, s(0), s(s(0)), \ldots, s^n(0), \ldots\}$, that is the natural numbers in Peano notation.

Since reduction to E/B-canonical form is just functional evaluation, $Can_{\Sigma/E,B}$ is the set of values computed by the functional program $(\Sigma, B, \vec{E})$. These are the values computed by Maude's `red` command!

The Terms in Canonical Form (II)

Here is the general definition:

Definition

Let $(\Sigma, E \uplus B)$ satisfy conditions (1)–(2), and (3′)–(4′). Then $Can_{\Sigma/E,B}$ is the S-indexed family of sets

$$Can_{\Sigma/E,B} = \{Can_{\Sigma/E,B,s}\}_{s \in S}$$

where for each $s \in S$ we define $Can_{\Sigma/E,B,s} = \{[can_{E/B}(t)]_B \in T_{\Sigma,[s]}/{=_B} \mid t \in T_{\Sigma,[s]} \wedge \exists t' \in [can_{E/B}(t)]_B \text{ s.t. } t' \in T_{\Sigma,s}\}$.

Therefore, since we are reasoning modulo axioms B, in $Can_{\Sigma/E,B}$ we consider all terms B-equivalent to a term $can_{E/B}(t)$ as the same value obtained from evaluating t using $(\Sigma, B, \vec{E})$.

The Idea of Sufficient Completeness

Consider the equations $E = \{x + 0 = x, x + s(y) = s(x + y)\}$ and observe that the set $Can_{\Sigma/E}$ is precisely the set T_{DL} of terms in the signature Σ_{DL} with symbols 0 and s. That is, the addition symbol has completely disappeared! This is as it should be, since the equations $E = \{x + 0 = x, x + s(y) = s(x + y)\}$ provide a complete definition of the addition function on natural numbers. Note that we have a strict inclusion $\Sigma_{DL} \subset \Sigma$.

In general, if $(\Sigma, E \uplus B)$ satisfies (1)–(2) and (3′)–(4′), we can use operations in a subsignature $\Omega \subseteq \Sigma$ as data constructors, so that the remaning operations in $\Sigma - \Omega$ are functions operating on data built with the data constructors Ω and returning as result another data value built with the constructors Ω.

The functions $f \in \Sigma - \Omega$ are then completely defined if for each $t \in T_{\Sigma}$, we have $can_{E/B}(t) \in T_{\Omega}$.

Subsignatures

Before defining sufficient completeness we need to make more precise the notion of subsignature.

Definition

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

1. $S' \subseteq S$ and $<' \subseteq <$, and
2. each operator declaration $f : s_1 \ldots s_n \to s$ in G is also an operator declaration in F, which we abbreviate with the notation $G \subseteq F$.

Sufficient Completeness Defined

Definition

Let (Σ, B, R) be a rewrite theory that is weakly ground terminating, and let $\Omega \subseteq \Sigma$ be a subsignature inclusion where Ω has the same poset of sorts as Σ, that is, $\Sigma = ((S, <), F)$, $\Omega = ((S, <), G)$, and $G \subseteq F$. We say that the rules R are sufficiently complete modulo B with respect to the constructor subsignature Ω iff for each $s \in S$ and each $t \in T_{\Sigma,s}$ there is a $t' \in T_{\Omega,s}$ such that $t \to^{!}_{R/B} t'$.

More on Sufficient Completeness

If Σ is kind-complete, then the above requirement that for each $t \in T_{\Sigma,s}$ there is a $t' \in T_{\Omega,s}$ such that $t \to^{!}_{R/B} t'$ should apply only to the sorts $s \in [s]$ in each connected component, but not to the kinds $\top_{[s]}$. I.e., the sufficient completeness for R modulo B should required for a signature Σ before kind-completing it to $\widehat{\Sigma}$.

This is because, since terms that have a kind $[s]$ but not a sort s, correspond to undefined or error expressions, such as $p(0)$ for p the predecessor function on natural numbers, it is perfectly possible that a completely well-defined function on the right sorts cannot be simplified away when applied to arguments of wrong sorts.

More on Sufficient Completeness (II)

If (Σ, B, E) has $\Omega \subseteq \Sigma$ as a constructor subsignature with E confluent and weakly terminating modulo B, we say that the constructors Ω are free modulo B in (Σ, B, E) iff for each sort s which is not a kind we have $Can_{\Sigma/E,B,s} = T_{\Omega/B,s}$.

Therefore, if we have identified for our rewrite theory (Σ, B, R) a subsignature of Ω of constructors, a fifth and last requirement should be:

(5) the rules R are sufficiently complete modulo B.

Examples of Sufficient Completeness Modulo B

For example, consider the reverse function in the list module

```
fmod MY-LIST is protecting NAT .
  sorts NeList List .
  subsorts Nat < NeList < List .
  op _;_ : List List -> List [assoc] .
  op _;_ : NeList NeList -> NeList [assoc ctor] .
  op nil : -> List [ctor] .
  op rev : List -> List .
  eq rev(nil) = nil .
  eq rev(N:Nat ; L:List) = rev(L:List) ; N:Nat .
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 completness can be automatically checked under reasonable assumptions.

Examples of Sufficient Completeness Modulo B (III)

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

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

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

Examples of Sufficient Completeness Modulo B (VI)

The following example shows an equational theory whose constructors are not free.

```
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
```