ISR: Lecture 2

José Meseguer

University of Illinois at Urbana-Champaign (USA)

Terms

Given an order-sorted signature $\Sigma = ((S, <), F)$, the Σ-terms are, intuitively, the well-typed expressions that can be formed using the symbols in Σ. Each Σ-term can be typed with some sort in S, so that Σ-terms form an S-indexed family of terms:

$$T_\Sigma = \{T_{\Sigma,s}\}_{s \in S}$$

Assuming that we use a prefix notation for the function declarations in F, this family is defined inductively as follows:

- If $a :\rightarrow s$ is a constant declaration in F, then $a \in T_{\Sigma,s}$
- if $t \in T_{\Sigma,s}$ and $s < s'$, then $t \in T_{\Sigma,s'}$
- if $f : s_1 \dots s_n \rightarrow s$ is a function declaration in F and $t_1 \in T_{\Sigma,s_1}, \dots, t_n \in T_{\Sigma,s_n}$, then $f(t_1, \dots, t_n) \in T_{\Sigma,s}$.

Note that, as defined, Σ-terms do not have variables. That is they are so-called ground terms.

An Example of Terms

Consider the following order-sorted signature $\Sigma = ((S, <), F)$ of data constructors for the natural numbers:

```
sorts Nat NzNat .
subsorts NzNat < Nat .
op 0 : -> Nat [ctor] .
op s : Nat -> NzNat [ctor] .
```

Then we have:

$$T_{\Sigma,NzNat} = \{s(0), s(s(0)), \ldots, s^n(0), \ldots\}$$

and

$$T_{\Sigma,Nat} = \{0\} \cup T_{\Sigma,NzNat}.$$

Terms with Variables

We can reduce the notion of term with variables to that of ground term as follows. Given a signature $\Sigma = ((S, <), F)$ and an S-indexed family of sets of typed variables $X = \{X_s\}_{s \in S}$ such that:

- all variables are different from the constants in Σ, and
- for $s, s' \in S$, $s \neq s' \Rightarrow X_s \cap X_{s'} = \emptyset$,

we can then "add the variables in X as new constants" to obtain the signature $\Sigma(X) = ((S, <), F(X))$, where, by definition, $F(X) = F \cup \{x :\rightarrow s \mid x \in X_s \wedge s \in S\}$.

Then a Σ-term t with variables in X is just a ground $\Sigma(X)$-term.

Notation: We denote by T_Σ the union $\bigcup_{s \in S} T_{\Sigma,s}$. Likewise, $T_{\Sigma(X)}$ denotes the union $\bigcup_{s \in S} T_{\Sigma(X),s}$.

Subterms

In a Σ-term $f(t_1, \ldots, t_n)$, the $t_1, \ldots, t_n$ are called its immediate subterms, denoted $t_i \lhd f(t_1, \ldots, t_n)$, $1 \leq i \leq n$. Note that the inverse relation $\lhd^{-1} = \rhd$ is *well-founded* (see *STAC* 11.1).

A term u is called a subterm of t iff $t \rhd^* u$, and a proper subterm of t iff $t \rhd^+ u$. Note that the relation $\rhd^+$ is also well-founded and a strict order.

Given a term $t \in T_{\Sigma(X)}$, we denote by $vars(t)$ the set of its variables, that is, $vars(t) = \{x \in X \mid t \rhd^* x\}$.

A term t may contain different occurrences of the same subterm u. For example, the subterm $g(a)$ appears twice in the term $f(b, h(g(a)), g(a))$.

Context-Subterm Decomposition of a Term

To indicate where a subterm is located we can replace it by a *hole*, a new constant [], added at the kind level to the signature Σ, marking where the subterm u was before we removed it.

For example, we can indicate the two places where $g(a)$ occurs in $f(b, h(g(a)), g(a))$ by $f(b, h([]), g(a))$ and $f(b, h(g(a)), [])$. A term with a single occurrence of a hole is called a context.

We write $C[]$ to denote a context. Given a context $C[]$ and a term u, we can obtain a new term, denoted $C[u]$, by replacing the hole [] by the term u. For example, if $C[] = f(b, h([]), g(a))$ and $u = k(b, y)$, then $C[u] = f(b, h(k(b, y)), g(a))$.

Context-Subterm Decomposition of a Term (II)

Of course, if $C[]$ is the context obtained from a term t by placing a hole $[]$ were subterm u occurred, then we have the term identity $t = C[u]$.

That is, we can always decompose a term t into a context and a chosen subterm, where if $t = C[u]$, then the decomposition of t into the context-subterm pair $(C[], u)$ is succinctly indicated by the more compact notation $C[u]$.

For example, we have, among others, the following decompositions of our term $f(b, h(g(a)), g(a))$:

$$f(b, h([g(a)]), g(a)) = f(b, [h(g(a))], g(a)) = [f(b, h(g(a)), g(a))]$$

where the last decomposition has an "empty context" $[]$.

Equations and Equational Theories

Given an order-sorted signature $\Sigma = ((S, <), F)$, a Σ-equation is an atomic formula $t = t'$, where $t, t' \in T_{\Sigma(X)}$, and where we require that $t = t'$ is well typed, i.e., there are sorts $s, s' \in S$ such that $t \in T_{\Sigma(X),s}$, $t' \in T_{\Sigma(X),s'}$, and $[s] = [s']$. Recall that $[s]$ denotes the $\equiv_{\leq}$-equivalence class (connected component) of $s \in S$.

An equational theory is then a pair (Σ, E), with Σ and order-sorted signature, and E a set of Σ-equations.

In an equational theory (Σ, E) all equations $t = t' \in E$ are implicitly assumed to be universally quantified as

$$(\forall x_1 : s_1, \ldots, x_n : s_n)\ \ t = t'$$

with $vars(t = t') = \{x_1 : s_1, \ldots, x_n : s_n\}$, where, by definition, $vars(t = t') = vars(t) \cup vars(t')$.

Equational Deduction: Replacing Equals by Equals

Equational deduction is the systematic replacement of equals by equals using the given equations E.

For example, we may use ring theory equations such as:
(1) $x + y = y + x$, (2) $x * y = y * x$,(3) $(x + y) + z = x + (y + z)$,
(4) $x + 0 = x$, (5) $x * 1 = x$, (6) $x * (y + z) = (x * y) + (x * z)$, to prove the polynomial equality $y + (z + (0 + (1 * x)) = (y + z) + x$ by the following sequence of replaments of equals by equals:

$$(\ddagger)\quad y+(z+[0+(1*x)]) = y+(z+[(1*x)+0]) = y+(z+[1*x]) =$$

$$y + (z + [x * 1]) = [y + (z + x)] = (y + z) + x$$

where at each point the subterm where an equation is applied is marked by the term decomposition.

Equational Deduction: Replacing Equals by Equals (II)

We can make the above proof of equality ($\ddagger$) more informative by giving a name, say ALG, to the above set (1)–(6) of equations, and indicating a proof step:

- applying an equation from left to right by $t \rightarrow_{ALG} t'$,
- a proof step from right to left by $t \leftarrow_{ALG} t'$, and
- a proof step in either direction by $t \leftrightarrow_{ALG} t'$.

With this notation we obtain the more informative proof:

$$y+(z+[0+(1*x)]) \leftrightarrow_{ALG} y+(z+[(1*x)+0]) \rightarrow_{ALG} y+(z+[1*x]) \leftrightarrow_{ALG}$$

$$y + (z + [x * 1]) \rightarrow_{ALG} [y + (z + x)] \leftarrow_{ALG} (y + z) + x.$$

Term Rewriting

Certain equations, for example equations (3)–(6) in ALG, can be applied from left to right as algebraic simplification rules, because their righthand side is clearly simpler, so that applying them leads to a simpler expressions.

Algebraic simplification produces a special type of equational proofs, called algebraic simplification proofs, where equations are always applied from left to right. Here is an algebraic simplification proof with equations in ALG for a polynomial expression:

$$([x{+}0]{*}(y{+}(z{*}1))){+}x' \to_{ALG} (x{*}(y{+}[z{*}1])){+}x' \to_{ALG} [x{*}(y{+}z)]{+}x' \to_{ALG}$$

$$[((x * y) + (x * z)) + x'] \to_{ALG} (x * y) + ((x * z) + x')$$

This process is called term rewriting, or term reduction.

Rewrite Rules and Term Rewriting Systems

We can make term rewriting explicit by choosing and orientation for an equation: we can orient an equation $t = t'$ from left to right as a so-called rewrite rule $t \to t'$, and from right to left as the rewrite rule $t' \to t$.

Definition

(Rewrite Rules and Term Rewiting Systems). Given an order-sorted signature $\Sigma = ((S, <), F)$, a Σ-rewrite rule is a sequent $t \to t'$, where $t, t' \in \bigcup T_{\Sigma(X)}$, and where we require that the rule $t \to t'$ is well typed, in the sense that there are sorts $s, s' \in S$ such that $t \in T_{\Sigma(X),s}$, $t' \in T_{\Sigma(X),s'}$, and $[s] = [s']$.

A term rewriting system is then a pair (Σ, R), with Σ and order-sorted signature, and R a set of Σ-rewrite rules.

Substitutions

Substitutions are mappings instantiating variables to terms. Since they have to preserve sorts, a substitution θ is actually an S-indexed mapping

$$\theta = \{\theta_s : X_s \to T_{\Sigma(Y),s}\}_{s \in S}$$

The application $t\theta$ of θ to a term $t \in T_{\Sigma(X)}$ is the term $t\theta \in T_{\Sigma(Y)}$ defined inductively as follows:

- for $x \in X_s$, $x\theta = \theta_s(x) \in T_{\Sigma(Y)}$
- for $f(t_1, \ldots, t_n) \in T_{\Sigma(X)}$, $n \geq 0$, $f(t_1, \ldots, t_n)\theta = f(t_1\theta, \ldots, t_n\theta) \in T_{\Sigma(Y)}$.

Example: For $\theta = \{x \mapsto x' + z', y \mapsto 0 + s(y')\}$ and $t = s(x) * y$, $t\theta = s(x' + z') * (0 + s(y'))$.

The Rewrite Relation

Definition

Let $\Sigma = ((S, <), F)$ be a kind-complete signature, (Σ, R) a term rewriting system, and $Y = \{Y_s\}_{s \in S}$ be an S-indexed set of variables. Then an R-rewrite step is a pair (u, v), denoted $u \to_R v$, such that $u \in T_{\Sigma(Y)}$ and there is a rewrite rule $t \to t' \in R$, a substitution $\theta : vars(t \to t') \longrightarrow T_{\Sigma(Y)}$, and a term decomposition $u = C[t\theta]$ such that $v = C[t'\theta]$, where, by definition, $vars(t \to t') = vars(t) \cup vars(t')$.

Since Σ is kind-complete, if $t \to t' \in R$ and $u = C[t\theta] : [s]$, then we must have $v = C[t'\theta] : [s]$, that is, $\to_R$ never produces ill-formed terms.

We denote by $\to_R^+$ the transitive closure of $\to_R$, and by $\to_R^*$ the reflexive-transitive closure of $\to_R$.

Rewrite Proofs

Definition

A (Σ, R)-rewrite proof is, by definition, either:

- a term $t \in \bigcup T_{\Sigma(Y)}$, witnessing a 0-step rewrite $t \to_R^* t$, or
- a sequence of R-rewrite steps of the form

$$t_0 \to_R t_1 \to_R t_2 \ldots t_{n-1} \to_R t_n$$

with $n \geq 1$, witnessing $t_0 \to_R^+ t_n$.

The Equality Relation and Equational Proofs

The notion of an equational proof, that is, a sequence of steps of replacement of equals by equals using equations E, is a trivial instance of the notion of a rewrite proof.

Given an equational theory (Σ, E), all we need to do is to consider proofs in the term rewriting system $(\Sigma, \overrightarrow{E} \cup \overleftarrow{E})$, where, by definition:

- $\overrightarrow{E}$ is the set of left-to-right orientations $\overrightarrow{E} = \{t \to t' \mid t = t' \in E\}$; and

- $\overleftarrow{E}$ is the set of right-to-left orientations $\overleftarrow{E} = \{t' \to t \mid t = t' \in E\}$.

The Equality Relation and Equational Proofs (II)

Definition

Given an equational theory (Σ, E) with Σ kind-complete and with nonempty sorts (i.e., $\forall s \in S\ T_{\Sigma,s} \neq \emptyset$) an *E-equality step* is, by definition, a $(\overrightarrow{E} \cup \overleftarrow{E})$-rewrite step $u \rightarrow_{(\overrightarrow{E} \cup \overleftarrow{E})} v$, denoted $u \leftrightarrow_E v$, where $u, v \in T_{\Sigma(Y)}$ for some variables Y.

$\leftrightarrow_E^+$ denotes the transitive closure of $\leftrightarrow_E$; and $\leftrightarrow_E^*$ the reflexive-transitive closure of $\leftrightarrow_E$. $\leftrightarrow_E^*$ is called the E-equality relation, and is often abbreviated to $=_E$. It is also called the relation of equality modulo E.

A (Σ, E)-equality proof is by, definition, either a term $t \in \bigcup T_{\Sigma(Y)}$, witnessing a 0-step E-equality $t \leftrightarrow_E^* t$, or a sequence of E-equality steps of the form $t_0 \leftrightarrow_E t_1 \leftrightarrow_E t_2 \ldots t_{n-1} \leftrightarrow_E t_n$, with $n \geq 1$, witnessing $t_0 \leftrightarrow_E^+ t_n$.

Term Rewriting Modulo Axioms

Certain equations are intrinsically problematic for term rewriting. For example, the commutativity equation $x + y = y + x$ is intrinsically problematic for rewriting because:

- we do not obtain a simpler term, but only a "mirror image" of the original term; for example, $(x * 7) + (0 * y)$ is rewritten to $(0 * y) + (x * 7)$; and
- even worse, we can easily *loop* when applying this equation, as in the infinite, alternating sequence

$$(x*7)+(0*y) \rightarrow_{ALG} (0*y)+(x*7) \rightarrow_{ALG} (x*7)+(0*y) \rightarrow_{ALG} \cdots$$

The solution to this problem is to build in certain, commonly occurring equational axioms, such as the above commutativity axioms, so that rewriting takes place modulo such axioms.

Term Rewriting Modulo Axioms (II)

For example, we can decompose our equations ALG into a built-in, commutative part $C = \{x + y = y + x, x * y = y * x\}$ and the rest, say, $ALG_0 = \{(x + y) + z = x + (y + z), x + 0 = x, x * 1 = x, x * (y + z) = (x * y) + (x * z)\}$, and then rewrite with the equations in ALG_0 from left to right applying them, not just to the given term t, but to any other term t' which is provably equal to t by the equations C.

This, more powerful rewrite relation is called rewriting modulo C, and is denoted $\rightarrow_{ALG_0/C}$. For example, we can simplify the expression $((0 + x) * ((1 * y) + 7)) + z$ to $(x * y) + ((x * 7) + z)$ in just four steps with $\rightarrow_{ALG_0/C}$ as follows:

$$((0+x)*((1*y)+7))+z \rightarrow_{ALG_0/C} (x*((1*y)+7))+z \rightarrow_{ALG_0/C} (x*(y+7))+z \rightarrow_{ALG_0/C} ((x*y)+(x*7))+z \rightarrow_{ALG_0/C} (x*y)+((x*7)+z)$$

Term Rewriting Modulo Axioms (III)

But why stopping with commutativity? How about associativity? An associativity (A) equation such as $(x+y)+z = x+(y+z)$ has no looping problems; but parentheses around associative operators are a nuisance and can block the application of equations.

For example, we can simplify to 0 the term $((x+y)+z)+\ -(y+(z+x))$ in one step of rewriting modulo the following set AC of associativity and commutativity axioms for $\_+\_$ and $\_*\_$, $AC = \{x+y = y+x, x*y = y*x, (x+y)+z = x+(y+z), (x*y)*z = x*(y*z)\}$, using the single equation $ALG_1 = \{x+\ -x = 0\}$ oriented as the rule $x+\ -x \to 0$.

$$((x+y)+z)+\ -(y+(z+x)) \to_{ALG_1/AC} 0.$$

That is, when rewriting modulo AC: (i) the order of the arguments does not matter (because of commutativity, C), and (ii) parentheses do not matter (because of associativity, A).

Rewrite Theories

Likewise, we could also build in the unit element axioms $U = \{x + 0 = x, x * 1 = x\}$. Or any combination of C, and/or A, and/or U axioms could be built in.

In fact, the idea of building in a set B of equational axioms, so that we rewrite with a set of rules R modulo B, is entirely general, and is associated to the notion of a rewrite theory.

Definition

Let Σ be a sensible order-sorted signature. A rewrite theory is a triple (Σ, B, R), where B is a set of Σ-equations, and R is a set of Σ-rewrite rules.

Rewriting with R modulo B can then be formalized as follows.

Rewriting Modulo B

Definition

Let (Σ, B, R) be a rewrite theory such that Σ is sensible and kind-complete. Then an R-rewrite step modulo B is a pair $(u, v) \in T^2_{\Sigma(Y)}$, denoted $u \to_{R/B} v$, such that there are terms $u', v' \in T_{\Sigma(Y)}$ with $u =_B u'$, $u' \to_R v'$, and $v' =_B v$, that is, we have $u =_B u' \to_R v' =_B v$.

We call $\to_{R/B}$ the *one-step* R-rewrite relation modulo B, and denote by $\to^0_{R/B}$ the relation $=_B$, called the 0-step R-rewrite relation modulo B, by $\to^+_{R/B}$ the transitive closure of $\to_{R/B}$, and by $\to^*_{R/B}$ the relation $\to^+_{R/B} \cup (=_B)$.

Rewrite Proofs Modulo B

Definition

An R-rewrite proof modulo B is either:

- a pair $(u, v) \in T^2_{\Sigma(Y)}$, with $u =_B v$, witnessing a *0-step* R-rewrite modulo B, or
- a sequence of R-rewrite steps modulo B of the form

$$v_0 \to_{R/B} v_1 \to_{R/B} v_2 \ldots v_{n-1} \to_{R/B} v_n,$$

$n \geq 1$, witnessing $v_0 \to^+_{R/B} v_n$.

Examples of Equational Simplification Modulo *B*

Lists modulo associativity and identity, with membership:

```
fmod LIST-AID is
  protecting NAT .
  sort List .
  subsort Nat < List .
  op nil : -> List [ctor] .
  op _;_ : List List -> List [assoc id: nil ctor] .
  op _in_ : Nat List -> Bool .
  var N : Nat .  vars L L' : List .
  eq N in L ; N ; L' = true .
  eq N in L = false [owise] .
endfm
reduce in LIST-AID : 7 in 3 ; 4 ; 9 .
result Bool: false
==========================================
reduce in LIST-AID : 7 in 4 ; 3 ; 7 .
result Bool: true
```

Examples of Equational Simplification Modulo B (II)

Lists modulo associativity with membership. More patterns are need.

```
fmod LIST-A is
  protecting NAT .  sort List .  subsort Nat < List .
  op nil : -> List [ctor] .
  op _;_ : List List -> List [assoc ctor] .
  op _in_ : Nat List -> Bool .
  var N : Nat .  vars L L' : List .
  eq nil ; L = L .
  eq L ; nil = L .
  eq N in N = true .
  eq N in N ; L = true .
  eq N in L ; N = true .
  eq N in L ; N ; L' = true .
  eq N in L = false [owise] .
endfm

reduce in LIST-A : 7 in 4 ; 3 ; 7 .
result Bool: true
```

Examples of Equational Simplification Modulo B (III)

Multisets modulo associativity, commutativity, and identity.

```
fmod MSET-ACU is
  protecting NAT .
  sort MSet .
  subsort Nat < MSet .
  op nil : -> MSet [ctor] .
  op _;_ : MSet MSet -> MSet [assoc comm id: nil ctor] .
  op _in_ : Nat MSet -> Bool .
  var N : Nat .  var S : MSet .
  eq N in N ; S = true .
  eq N in S = false [owise] .
endfm
reduce in MSET-ACID : 7 in 3 ; 4 ; 9 .
result Bool: false
==========================================
reduce in MSET-ACID : 7 in 4 ; 3 ; 7 .
result Bool: true
```

Examples of Equational Simplification Modulo B (IV)

Multisets modulo associativity and commutativity: more patterns needed.

```
fmod MSET-AC is
  protecting NAT .
  sort MSet .              subsort Nat < MSet .
  op nil : -> MSet [ctor] .
  op _;_ : MSet MSet -> MSet [assoc comm ctor] .
  op _in_ : Nat MSet -> Bool .
  var N : Nat .  var S : MSet .
  eq nil ; S = S .
  eq N in N = true .
  eq N in N ; S = true .
  eq N in S = false [owise] .
endfm
reduce in MSET-AC : 7 in 3 ; 4 ; 9 .
result Bool: false
==========================================
reduce in MSET-AC : 7 in 4 ; 3 ; 7 .
result Bool: true
```

Examples of Equational Simplification Modulo B (V)

Sets of natural numbers using identity and idempotency equations.

```
fmod NAT-SET is protecting NAT .
  sort NatSet .
  subsort Nat < NatSet .
  op mt : -> NatSet [ctor] .
  op _ _ : NatSet NatSet -> NatSet [ctor assoc comm] . *** set union
  op _/\ _ : NatSet NatSet -> NatSet [assoc comm] .    *** intersection
  vars X Y : NatSet .       var N : Nat .
  eq mt X = X .                                        *** identity
  eq X X = X .                                         *** idempotency
  eq N /\ N = N .
  eq N /\ (N X) = N .
  eq (N X) /\ (N Y) = N (X /\ Y) .
  eq X /\ Y = mt [owise] .
endfm
Maude> red (1 2 3 4 5) /\ (3 4 5 6 7) .
result NatSet: 3 4 5
```

Caveats on Equational Simplification Modulo B

Equational simplification modulo identity is trickier. For example, the innocent-looking idempotency equation in

```
fmod NAT-SET' is protecting NAT .
  sort NatSet .
  subsort Nat < NatSet .
  op mt : -> NatSet [ctor] .
  op _ _ : NatSet NatSet -> NatSet [ctor assoc comm id: mt] .
  var X : NatSet .
  eq X X = X .
endfm
```

is nonterminating, since we have,

$$\texttt{mt} =_{ACI} \texttt{mt mt} \longrightarrow_{E} \texttt{mt} =_{ACI} \texttt{mt mt} \longrightarrow_{E} \ldots$$

Caveats on Equational Simplification Modulo B (II)

Nontermination can be avoided by giving instead a more careful equation, where we restrict idempotency to pairs of elements (yet, with the same effect, sice this ensures that all repeated elements will be eliminated) by means of the (now terminating) equation,

```
var N : Nat .
eq N N = N .
```

Another alternative is to declare:

```
sort NatSet NeNatSet .
subsort Nat < NeNatSet < NatSet .
op mt : -> NatSet [ctor] .
op _ _ : NatSet NatSet -> NatSet [ctor assoc comm id: mt]
op _ _ : NeNatSet NeNatSet -> NeNatSet [ctor assoc comm
var X : NeNatSet .
eq X X = X .
```

Readings

All the theoretical aspects of the material presented in this lecture are covered in detail in *STAC* 13.1 and 13.2.