Foundations of System Development

Martin Wirsing

in cooperation with
Axel Rauschmayer

WS 05/06

Goals

Get to know

- constructor subsignatures
- equational simplification modulo equations

Constructor Subsignature

Let T = (Σ,E) be an equational theory with equations E that are ground confluent and terminating.
Then a subsignature Ω is called the **constructor subsignature** for T = (Σ,E) iff:

1. $\Omega \subset \Sigma$
2. Ω has the same poset of sorts (S,≤) as Σ, and
3. Ω is the smallest signature satisfying (1) and (2) above and such that its terms contain all canonical forms:

   $Can_{\Sigma,E}|_{\Omega} \subset T_{\Omega}$.

We call Ω a subsignature of **(absolutely) free constructors** if the equality $Can_{\Sigma,E}|_{\Omega} = T_{\Omega}$ holds.

Constructor Subsignature

- In Maude, we indicate the constructor subsignature by declaring the **ctor** attribute.
- **Example**

```
fmod NAT-MIXFIX is
    sort Natural .
    op 0 : -> Natural [ctor] .
    op s_ : Natural -> Natural [ctor] .
    op _+_ : Natural Natural -> Natural .
    vars N M : Natural .
    eq N + 0 = N .
    eq N + s M = s(N + M) .
endfm
```

The annotations indicate the, in this case (absolutely) free, constructor subsignature.

Constructor Subsignature

The subsignature of constructors is **not always (absolutely) free**.

Example

```
fmod NAT-3 is
   sort Natural .
   op 0 : -> Natural [ctor] .
   op s_ : Natural -> Natural [ctor] .
   op _+_ : Natural Natural -> Natural .
   vars N M : Natural .
   eq N + 0 = N .
   eq N + s M = s(N + M) .
   eq s s s 0 = 0 .
endfm
```

$Can_{\Sigma,E,Natural}$ = {0, s 0, s s 0}. Therefore, $Can_{\Sigma,E}|_W$ is different from T_Ω.

Constructor Subsignature

Note that

a given subsort overloaded operator may be a constructor, while another subsort overloaded version of the same operator may not be.

Example

```
fmod INTEGER is
   sorts Zero Natural NzNatural Negative NzNegative Integer .
   subsorts Zero NzNatural < Natural < Integer .
   subsorts Zero NzNegative < Negative < Integer .
   op 0 : -> Zero [ctor] .
   op s_ : Natural -> NzNatural [ctor] .
   op s_ : Integer -> Integer .
   op p_ : Negative -> NzNegative [ctor] .
   op p_ : Integer -> Integer .
   var I : Integer .
   eq p s I = I .   eq s p I = I .
endfm
```

defines a free constructor subsignature for the integers.

Equational Simplification modulo Equations

- **Equations as attributes of Operators**
  In Maude, we allow certain equations, namely
  associativity, commutativity, and identity
  to be declared as **attributes of operators** by means of
  assoc, comm, and **id**: _
- **Mathematically**, this means that the alg. specification is of the form
  $(\Sigma, E \cup A)$ with
  - E the equations explicitly given in the module, and
  - A the equations implicitly declared by attributes such as
    assoc, comm, and id:_.
- **Operationally,** this means is that
  we can apply the equations in E modulo the axioms A.

Equational Simplification modulo Equations

Example

The equation N + 0 = N applies to

- the term s(0)+(0+s(0)) modulo associativity of + ; and to
- the term 0+s(0) modulo commutativity of + .

Example of Simplification modulo Equations

Lists modulo associativity and identity, with membership:

```
fmod LIST-AID is
  protecting NAT .
  sort List .
  subsort Nat < List .
  op nil : -> List .
  op _;_ : List List -> List [assoc id: nil] .
  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

Example of Simplification modulo Equations

Lists modulo associativity, with membership:
More patterns need to be considered without the identity attribute.

```
fmod LIST-A is
  protecting NAT . sort List . subsort Nat < List .
  op nil : -> List .
  op _;_ : List List -> List [assoc] .
  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
```

Example of Simplification modulo Equations

```
reduce in LIST-A : 7 in 3 ; 4 ; 9 .
result Bool: false
==========================================
reduce in LIST-A : 7 in 4 ; 3 ; 7 .
result Bool: true
```


Examples of Simplification modulo Equations

Multisets modulo associativity, commutativity, and identity.

```
fmod MSET-ACID is
  protecting NAT .
  sort MSet .
  subsort Nat < MSet .
  op nil : -> MSet .
  op _;_ : MSet MSet -> MSet [assoc comm id: nil] .
  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 Simplification modulo Equations

Multisets modulo associativity and commutativity.

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


Examples of Simplification modulo Equations

```
reduce in MSET-ACID : 7 in 3 ; 4 ; 9 .
result Bool: false
==========================================
reduce in MSET-ACID : 7 in 4 ; 3 ; 7 .
result Bool: true
```

Equational Simplification modulo A

The above examples and reduce commands illustrate equational simplification modulo A, for A any combination of associativity, commutativity, and identity axioms.

Let T = $(\Sigma, E \cup A)$ be a theory whose equations E are admissible as equational simplification rules.
Then, we can define a binary relation on terms in T, denoted $\rightarrow_{E/A}$, and
called **one-step equational simplification modulo A**, by as follows:

$$t \rightarrow_{E/A} t' \quad \text{if, and only, if} \quad t \equiv_A u \rightarrow_E v \equiv_A t'.$$

Equational Simplification modulo A

- Note that, denoting equivalence classes modulo A by [t], simplification modulo A defines also a relation (with the same notation) on T/A as follows:

  $$[t] \rightarrow_{E/A} [t'] \text{ if, and only, if } t \rightarrow_{E/A} t'.$$

- Conceptually, this is the best way of thinking of this form of equational simplification:
  - we think of equivalence classes [t] modulo A as
    abstract data structures
    (e.g., strings for A associativity,
    and multisets for A associativity and commutativity)
  - we think of $\rightarrow_{E/A}$ as acting not on terms, but on such abstract data stuctures (for example, string rewriting, and multiset rewriting).

Another Example of Simplification modulo A

Sets of natural numbers by simplifying
multisets of natural numbers modulo associativity and commutativity,
using identity and idempotency equations.

```
fmod NAT-SET is  protecting NATURAL .
  sort NatSet .
  subsort Natural < NatSet .
  op empty : -> NatSet [ctor] .
  op _ _ : NatSet NatSet -> NatSet [ctor assoc comm label set
union] .
  var X : NatSet .
  eq empty X = X  [label  identity] .
  eq X X = X  [label idempotency] .
endfm
```


Caveats of Simplification modulo A

Equational simplification modulo identity is trickier:

Example

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

The innocent-looking idempotency equation is nonterminating, since,
denoting by $\equiv_{ACI}$ the congruence modulo associativity, commutativity, and identity, we have,

empty $\equiv_{ACI}$ empty empty $\to_E$ empty $\equiv_{ACI}$ empty empty $\to_E$ . . .

Caveats of Simplification modulo A

We can avoid this nontermination problem by giving instead a 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 : Natural .
eq N N = N .
```


All Results Generalize Modulo

Under reasonable assumptions on A,
all the concepts on equational simplification generalize in a natural way to equational simplification modulo A.
We define the relation

$\rightarrow_{E/A}{}^*$ as the reflexive and transitive closure of $\rightarrow_{E/A}$.

The definitions of confluence and termination are the same, replacing $\rightarrow_E$ by $\rightarrow_{E/A}$: we have

- soundness, and
- for confluent equations completeness,

of equational simplification modulo A.

All Results Generalize Modulo

- The concepts of **canonical term algebra** and of **constructor subsignature** also generalize,

  except that now canonical forms are equivalence classes modulo A, (Notation: $Can_{\Sigma,E/A}$) and if Ω are the constructors we have,

  $Can_{\Sigma,E/A}|_{\Omega} \subset T_{\Omega/A}$.
- We call the constructors **free modulo A** if we in fact have,

  $Can_{\Sigma,E/A}|_{\Omega} = T_{\Omega/A}$.
- **Example**

  The constructors in NAT-SET are not free modulo associativity and commutativity;

  e.g. the multiset 0 0 0 is not in canonical form.

All Results Generalize Modulo

Functional modules in Maude are of the form

fmod (Σ, E $\cup$ A) endfm,

where we assume E confluent and terminating modulo A.

- **Mathematical semantics:**

  initial algebra $T_{\Omega/E \cup A}$.
- **Operational semantics:**

  equational simplification with E modulo A.

- Both semantics coincide in the canonical term algebra, since we have the Σ–isomorphism

  $T_{\Omega/E \cup A} \cong Can_{\Sigma,E/A}$

Summary

- In Maude, we indicate the constructor subsignature by declaring the **ctor** attribute.
- Maude supports equational simplification modulo all combinations of
  - **associativity, commutativity, identity**.
- All concepts such as **canonical term algebra, soundness, and completeness** generalize in a natural way to **equational simplification modulo**.