Program Verification: Lecture 8

José Meseguer

Computer Science Department
University of Illinois at Urbana-Champaign

Termination

We need methods to check termination of an equational theory (Σ, E). For unconditional equations E this means proving that the rewriting relation $\longrightarrow_E$ (or, more generally, $\longrightarrow_{E/B}$ for $(\Sigma, E \cup B)$) is well-founded.

The key observation is that, if we exhibit a well-founded ordering $>$ on terms such that

$$(\clubsuit) \quad t \longrightarrow_E t' \quad \Rightarrow \quad t > t',$$

then we have obviously proved termination, since nontermination of $\longrightarrow_E$ would make the order $>$ non-well-founded.

Reduction Orderings

To show ($\clubsuit$) we need to consider an, infinite number of rewrites $t \longrightarrow_E t'$. We would like to reduce this problem to checking ($\clubsuit$) only for the equations in E. We need:

Definition: A well-founded ordering $>$ on $\cup_{s \in S} T_\Sigma(V)$ is called a reduction ordering iff it satisfies the following two conditions:

- **strict Σ-monotonicity**: for each $f \in \Sigma$, whenever $f(t_1, \ldots, t_n), f(t_1, \ldots, t_{i-1}, t'_i, t_{i+1}, \ldots, t_n) \in T_\Sigma(V)$ with $t_i > t'_i$, we have,

$$f(t_1, \ldots, t_n) \;>\; f(t_1, \ldots, t_{i-1}, t'_i, t_{i+1}, \ldots, t_n)$$

- **closure under substitutuion**: if $t > t'$, then, for any substitution $\theta : V \longrightarrow T_\Sigma(V)$ we have, $t\theta > t'\theta$.

Reduction Orderings (II)

Theorem: Let (Σ, E) be an (unconditional) equational theory. Then, E is terminating iff there exists a reduction order $>$ such that for each equation $u = v$ in E we have, $u > v$.

Proof: The $(\Rightarrow)$ part follows from the observation that, if E is terminating, the transitive closure $\overset{+}{\longrightarrow}_E$ of the relation $\longrightarrow_E$ is a reduction order satisfying this requirement.

To see $(\Leftarrow)$, it is enough to show that a reduction order with the above property satisfies the implication $(\clubsuit)$. Let $t \longrightarrow_E t'$ this means that there is a position π in t, an equation $u = v$ in E, and a substitution θ such that $t = t[\pi \leftarrow \overline{\theta}(u)]$, and $t' = t[\pi \leftarrow \overline{\theta}(v)]$. But by closure under substitution we have, $\overline{\theta}(u) > \overline{\theta}(v)$ and by repeated application of strict Σ-monotonicity we then have, $t > t'$. q.e.d.

Recursive Path Ordering (RPO)

The recursive path ordering (RPO) is based on the idea of giving an ordering on the function symbols in Σ, which is then extended to a reduction ordering on all terms. Since if Σ is finite the number of possible orderings between function symbols in Σ is also finite, checking whether a proof of termination exists this way can be automated.

The intuitive idea that functions that are more complex should be bigger in the ordering (for example: $\_*\_ > \_+\_ > \mathtt{s}$) tends to work quite well, and can yield a reduction ordering containg the equations. Furthermore each symbol f in Σ is given a status $\tau(f)$ equal to either: $\tau(f) = lex(\pi)$ (lexicographic), or $\tau(f) = mult$ (multiset). $\tau(f)$ indicates how the arguments of f should be compared in the order.

RPO (II)

Given a finite signature Σ and an ordering $>$ and a status function τ on its symbols, the recursive path ordering $>_{rpo}$ on $\cup_{s \in S} T_\Sigma(V)$ is defined recursively as follows. $u >_{rpo} t$ iff:

$u = f(u_1, \ldots, u_n)$, and either:

1. $u_i \geq_{rpo} t$ for some $1 \leq i \leq n$, or
2. t $= g(t_1, \ldots, t_m)$, $u >_{rpo} t_j$ for all $1 \leq j \leq m$, and either:
    - $f > g$, or
    - $f = g$ and $\langle u_1, \ldots, u_n \rangle >_{rpo}^{\tau(f)} \langle t_1, \ldots, t_n \rangle$

where the extension of $>_{rpo}$ to an order $>_{rpo}^{\tau(f)}$ on lists of terms is explained below.

RPO (III)

The extension of $>_{rpo}$ to an order $>_{rpo}^{\tau(f)}$ on lists of terms is defined as follows:

- If f has n arguments and $\tau(f) = lex(\pi)$ with π a permutation on n elements, then $\langle u_1, \ldots, u_n \rangle >_{rpo}^{\tau(f)} \langle t_1, \ldots, t_n \rangle$ iff there exists i, $1 \leq i \leq n$ such that for $j < i$ $u_{\pi(j)} = t_{\pi(j)}$, and $u_{\pi(i)} > t_{\pi(i)}$.

- if $\tau(f) = mult$, then $\langle u_1, \ldots, u_n \rangle >_{rpo}^{\tau(f)} \langle t_1, \ldots, t_n \rangle$ iff we have $\{u_1, \ldots, u_n\} >_{rpo}^{mult} \{t_1, \ldots, t_n\}$

where, given any order $>$ on a set A, it extension to an order $>^{mult}$ on the set $Mult(A)$ of multisets on A is the transitive closure of the relation $>_{elt}^{mult}$ defined by $M \cup a >_{elt}^{mult} M \cup S$ iff $(\forall x \in S)\ a > x$, where S can be $\emptyset$.

RPO (IV)

It can be shown (for a detailed proof see the Terese book cited later) that for a finite signature Σ RPO is a reduction order. We can therefore use it to prove termination.

Consider for example the usual equations for natural number addition: $n+0=n$ and $n+s(m)=s(n+m)$. We can prove that they are terminating by using the RPO associated to the ordering $+ > s > 0$ with $\tau(f) = lex(id)$ for each symbol f. Indeed, it is then trivial to check that $n+0 >_{lpo} n$ and $n+s(m) >_{lpo} s(n+m)$.

Termination Modulo Axioms

To prove that rewriting modulo axioms B are terminating, we need a reduction order that is compatible with the axioms B. That is, if $u > t$, $u =_B u'$ and $t =_B t'$, then we must always have $u' > t'$. This means that $>$ defines also an order on the set, $\cup_{s \in S} T_{\Sigma/B}(X)$. For example, RPO is compatible with commutativity axioms if we specify $\tau(f) = mult$ for each commutative symbol f.

To make RPO compatible with associative and commutative symbols it has been generalized to the $AC.RPO$ order by a method of flattening AC symbols. E.g., for f AC, $f(f(a,b), f(c,d))$ flattens to $f(a,b,c,d)$. $AC.RPO$ can be further generalized to the $A \vee C.RPO$ order, where symbols can be associative and/or commutative.

Proving Termination with $A \vee C.RPO$

A simple tool can be used to prove termination modulo any $A \vee C$ axioms using the $A \vee C.RPO$ reduction order. The tool supports two orders: (i) $AC.RPO$, which contains RPO as a special case; and (ii) the more general $A \vee C.RPO$.

The tool works as follows:

1. The user defines a liner order on the function symbols of Σ using Maude's `metadata` attribute, and enters Σ (with the axioms and the metadata) to the tool.
2. Then for each equations $u = v$ in the program, asks the tool whether $u >_{AC.rpo} v$ or $u >_{A \vee C.rpo} v$, depending on whether the axiom are only C or AC, or also include A.

Let us see some examples.

Proving Termination with $A \vee C.RPO$ (II)

Want to prove termination, with RPO and the given symbol order, of addition, multiplication and exponentiation in:

```
fmod NATU is sort Natu .
  op 0     : -> Natu [ctor metadata "1"] .
  op s     : Natu -> Natu [ctor metadata "2"] .
  op _+_   : Natu Natu -> Natu [comm metadata "3"] .
  op _*_   : Natu Natu -> Natu [metadata "4"] .
  op _^_   : Natu Natu -> Natu [metadata "5"] .

  eq X:Natu + 0 = X:Natu .
  eq X:Natu + s(Y:Natu) = s(X:Natu + Y:Natu) .
  eq X:Natu * 0 = (0).Natu .
  eq X:Natu * s(Y:Natu) = (X:Natu * Y:Natu) + X:Natu .
  eq X:Natu ^ 0 = (s(0)).Natu .
  eq X:Natu ^ s(Y:Natu) = X:Natu * (X:Natu ^ Y:Natu) .
endfm
```

For this load in the tool the module:

Proving Termination with $A \vee C.RPO$ (III)

```
load ../tuple.maude
load ../module-util.maude
load ../aacrpo.maude

set include BOOL off .

fmod NATU is sort Natu .
  op 0     : -> Natu [ctor metadata "1"] .
  op s     : Natu -> Natu [ctor metadata "2"] .
  op _+_   : Natu Natu -> Natu [comm metadata "3"] .
  op _*_   : Natu Natu -> Natu [metadata "4"] .
  op _^_   : Natu Natu -> Natu [metadata "5"] .
endfm

fmod AvC-RPO-EXT is
  pr AC-RPO .
  pr AAC-RPO .
  pr NATU .
endfm
```

Proving Termination with $A \vee C.RPO$ (IV)

Then execute the reduce commands (all yield true):

```
red in AvC-RPO-EXT : upTerm(X:Natu + 0) >AC{upModule('NATU,true)}
                                                  upTerm(X:Natu) .

red in AvC-RPO-EXT : upTerm(X:Natu + s(Y:Natu))>AC{upModule('NATU,true)}
                                              upTerm(s(X:Natu + Y:Natu)) .

red in AvC-RPO-EXT : upTerm(X:Natu * 0) >AC{upModule('NATU,true)}
                                               upTerm((0).Natu) .

red in AvC-RPO-EXT : upTerm(X:Natu * s(Y:Natu)) >AC{upModule('NATU,true)}
                                     upTerm((X:Natu * Y:Natu) + X:Natu) .

red in AvC-RPO-EXT : upTerm(X:Natu ^ 0) >AC{upModule('NATU,true)}
                                                upTerm((s(0)).Natu) .

red in AvC-RPO-EXT : upTerm(X:Natu ^ s(Y:Natu)) >AC{upModule('NATU,true)}
                                      upTerm(X:Natu * (X:Natu ^ Y:Natu)) .
```

Proving Termination with $A \vee C.RPO$ (V)

Want to prove $A \vee C.RPO$-termination of list (associative) and sets (associative-commutative) of naturals:

```
fmod NATU-LS is sorts Natu NatuList  NatuSet .
  subsorts Natu < NatuList  NatuSet .
  op 0      : -> Natu [ctor metadata "1"] .
  op mt     : -> NatuSet [ctor metadata "2"] .
  op nil    : -> NatuList [ctor metadata "3"] .
  op s      : Natu -> Natu [ctor metadata "4"] .
  op _+_    : Natu Natu -> Natu [comm metadata "5"] .
  op _*_    : Natu Natu -> Natu [metadata "6"] .
  op _^_    : Natu Natu -> Natu [metadata "7"] .
  op _,_    : NatuSet NatuSet -> NatuSet [ctor assoc comm metadata "7"] .
  op __     : NatuList NatuList -> NatuList [ctor assoc  metadata "8"] .
  op length   : NatuList -> Natu [metadata "9"] .
  op rev    : NatuList -> NatuList [metadata "10"] .
  op list2set : NatuList -> NatuSet [metadata "11"] .
  eq X:Natu + 0 = X:Natu .
  eq X:Natu + s(Y:Natu) = s(X:Natu + Y:Natu) .
  eq X:Natu * 0 = (0).Natu .
  eq X:Natu * s(Y:Natu) = (X:Natu * Y:Natu) + X:Natu .
  eq X:Natu ^ 0 = (s(0)).Natu .
  eq S:NatuSet , S:NatuSet = S:NatuSet .
  eq X:Natu ^ s(Y:Natu) = X:Natu * (X:Natu ^ Y:Natu) .
  eq length((nil).NatuList) = (0).Natu .
  eq length(X:Natu) = (s(0)).Natu .
  eq length(X:Natu L:NatuList) = s(length(L:NatuList)) .
  eq rev((nil).NatuList) = (nil).NatuList .
  eq rev(X:Natu) = X:Natu .
  eq rev(X:Natu L:NatuList) = rev(L:NatuList) X:Natu .
  eq list2set((nil).NatuList) = (mt).NatuSet .
  eq list2set(X:Natu) = X:Natu .
  eq list2set(X:Natu L:NatuList) = X:Natu , list2set(L:NatuList) .
endfm
```

For this load in the tool the module:

Proving Termination with $A \vee C.RPO$ (VI)

```
load ../tuple.maude
load ../module-util.maude
load ../aacrpo.maude

set include BOOL off .

fmod NATU-LS is
  sorts Natu NatuList  NatuSet .
  subsorts Natu < NatuList  NatuSet .
  op 0     : -> Natu [ctor metadata "1"] .
  op mt    : -> NatuSet [ctor metadata "2"] .
  op nil   : -> NatuList [ctor metadata "3"] .
  op s     : Natu -> Natu [ctor metadata "4"] .
  op _+_   : Natu Natu -> Natu [comm metadata "5"] .
  op _*_   : Natu Natu -> Natu [metadata "6"] .
  op _^_   : Natu Natu -> Natu [metadata "7"] .
  op _,_   : NatuSet NatuSet -> NatuSet [ctor assoc comm metadata "7"] .
  op __    : NatuList NatuList -> NatuList [ctor assoc  metadata "8"] .
  op length   : NatuList -> Natu [metadata "9"] .
  op rev    : NatuList -> NatuList [metadata "10"] .
  op list2set : NatuList -> NatuSet [metadata "11"] .
endfm

fmod AvC-RPO-EXT is
  pr AC-RPO .
  pr AAC-RPO .
  pr NATU-LS .
endfm
```

Since list concatenation is associative, we now need to use the $A \vee C.RPO$ order (denoted `>AAC` in the tool). For this we give the following reduce commands for each equation (all come out `true`):

Proving Termination with $A \vee C.RPO$ (VII)

```
red in AvC-RPO-EXT : upTerm(X:Natu + 0)
  >AAC{upModule('NATU-LS,true)} upTerm(X:Natu) .

red in AvC-RPO-EXT : upTerm(X:Natu + s(Y:Natu))
   >AAC{upModule('NATU-LS,true)} upTerm(s(X:Natu + Y:Natu)) .

red in AvC-RPO-EXT : upTerm(X:Natu * 0)
   >AAC{upModule('NATU-LS,true)} upTerm((0).Natu) .

red in AvC-RPO-EXT : upTerm(X:Natu * s(Y:Natu))
  >AAC{upModule('NATU-LS,true)} upTerm((X:Natu * Y:Natu) + X:Natu) .

red in AvC-RPO-EXT : upTerm(X:Natu ^ 0)
   >AAC{upModule('NATU-LS,true)} upTerm((s(0)).Natu) .

red in AvC-RPO-EXT : upTerm(X:Natu ^ s(Y:Natu))
   >AAC{upModule('NATU-LS,true)} upTerm(X:Natu * (X:Natu ^ Y:Natu)) .
```

```
red in AvC-RPO-EXT : upTerm(S:NatuSet , S:NatuSet)
  >AAC{upModule('NATU-LS,true)} upTerm(S:NatuSet) .

red in AvC-RPO-EXT : upTerm(length((nil).NatuList))
  >AAC{upModule('NATU-LS,true)} upTerm((0).Natu) .

red in AvC-RPO-EXT : upTerm(length(X:Natu))
  >AAC{upModule('NATU-LS,true)} upTerm((s(0)).Natu) .

red in AvC-RPO-EXT : upTerm(length(X:Natu L:NatuList))
   >AAC{upModule('NATU-LS,true)} upTerm(s(length(L:NatuList))) .

red in AvC-RPO-EXT : upTerm(rev((nil).NatuList))
  >AAC{upModule('NATU-LS,true)} upTerm((nil).NatuList) .

red in AvC-RPO-EXT : upTerm(rev(X:Natu))
  >AAC{upModule('NATU-LS,true)} upTerm(X:Natu) .

red in AvC-RPO-EXT : upTerm(rev(X:Natu L:NatuList))
  >AAC{upModule('NATU-LS,true)} upTerm(rev(L:NatuList) X:Natu) .
```

```
red in AvC-RPO-EXT : upTerm(list2set((nil).NatuList))
  >AAC{upModule('NATU-LS,true)} upTerm((mt).NatuSet) .

red in AvC-RPO-EXT : upTerm(list2set(X:Natu))
  >AAC{upModule('NATU-LS,true)} upTerm(X:Natu) .

red in AvC-RPO-EXT : upTerm(list2set(X:Natu L:NatuList))
  >AAC{upModule('NATU-LS,true)} upTerm(X:Natu , list2set(L:NatuList)) .
```

Polynomial Orderings

Another general method of defining suitable reduction orderings is based on polynomial orderings. In its simplest form we can just use polynomials on several variables whose coefficients are natural numbers. For example,

$$p = 7x_1^3x_2 + 4x_2^2x_3 + 6x_3^2 + 5x_1 + 2x_2 + 11$$

is one such polynomial. Note that a polynomial p whose biggest indexed variable is n (in the above example $n = 3$) defines a function $p_{\mathbf{N}_+} : \mathbf{N}_+^n \longrightarrow \mathbf{N}_+$, just by evaluating the polynomial on a given tuple of positive numbers. For p the polynomial above we have for example, $p_{\mathbf{N}_+}(1, 1, 1) = 35$.

Polynomial Orderings (II)

Note also that we can order the set $\mathbf{N}_+^{\mathbf{N}_+^n}$ of functions from $\mathbf{N}_+^n$ to $\mathbf{N}_+$ by defining $f > g$ iff for each $(a_1, \dots a_n) \in \mathbf{N}_+^n$ $f(a_1, \dots a_n) > g(a_1, \dots a_n)$. Notice that this order is well-founded, since if we have an infinite descending chain of functions

$$f_1 > f_2 > \dots f_n > \dots$$

by choosing any $(a_1, \dots a_n) \in \mathbf{N}_+^n$ we would get a descending chain of positive numbers

$$f_1(a_1, \dots a_n) > f_2(a_1, \dots a_n) > \dots f_n(a_1, \dots a_n) > \dots$$

which is impossible.

Polynomial Orderings (III)

The method of polynomial orderings then consists in assigning to each function symbol $f : s_1 \dots s_n \longrightarrow s$ in Σ a polynomial p_f involving exactly the variables $x_1, \dots x_n$ (all of them, and only them must appear in p_f). If f is subsort overloaded, we assign the same p_f to all such overloadings. Also, to each constant symbol b we likewise associate a positive number $p_b \in \mathbf{N}_+$.

Suppose, to simplify notation, that in our set E of equations we have used exactly k different variables, denoted $x_1, \dots x_k$, each declared with its corresponding sort. Let us denote $X = \{x_1, \dots x_k\}$. Then our assignment of a polynomial to each function symbol and a number to each constant extends to an S-sorted family of functions

Polynomial Orderings (IV)

$$p_{\_} : T_{\Sigma(X)} \longrightarrow \mathbf{N}[X]$$

where $\mathbf{N}[X]$ denotes the polynomials with natural number coefficients in the variables X, and where $p_{\_}$ is defined in the obvious, homomorphic way:

- $p_b = p_b$
- $p_{x_i} = x_i$
- $p_{f(t_1,\ldots,t_n)} = p_f(x_1/p_{t_1}, \ldots, x_n/p_{t_n})$

Polynomial Orderings (V)

Note that the the polynomial interpretation p induces a well-founded ordering $>_p$ on the terms of $T_{\Sigma(X)}$ as follows:

$$t >_p t' \quad \Leftrightarrow \quad p_{t_{\mathbf{N}_+}} > p_{t'_{\mathbf{N}_+}}$$

where if $X = \{x_1, \ldots x_k\}$, we interpret $p_{t_{\mathbf{N}_+}}$ and $p_{t'_{\mathbf{N}_+}}$ as functions in $\mathbf{N}_+^{\mathbf{N}_+^k}$. The relation $>_p$ is clearly an irreflexive and transitive relation on terms in $T_{\Sigma(X)}$, therefore a strict ordering, and is clearly well-founded, because otherwise we would have an infinite descending chain of polynomial functions in $\mathbf{N}_+^{\mathbf{N}_+^k}$, which is impossible.

Polynomial Orderings (VI)

We now need to check that this ordering is furthermore: (i) strictly Σ-monotonic, and (ii) closed under substitution. Condition (i) follows easily from the fact that for each function symbol $f : s_1 \dots s_n \longrightarrow s$ in Σ p_f involves exactly the variables $x_1, \dots x_n$ and has all its coefficients in $\mathbf{N}_+$. Therefore, $p_{f_{\mathbf{N}_+}}$, viewed as a function of n arguments, is strictly monotonic in each of its arguments. Condition (ii) follows from the following general property of the $p\_$ function, which is left as an excercise:

$$p_{t(x_1/u_1,\dots,x_n/u_n)} = p_t(x_1/p_{u_1}, \dots, x_n/p_{u_n}).$$

This then easily yields that if $t >_p t'$ then $t(x_1/u_1, \dots, x_n/u_n) >_p t'(x_1/u_1, \dots, x_n/u_n)$, as desired.

Polynomial Orderings (VII)

Therefore, polynomial interpretations of this kind define reduction orderings and can be used to prove termination. Consider for example the single equation $f(g(x)) = g(f(x))$ in an unsorted signature having also a constant a. Is this equation terminating? We can prove that it is so by the following polynomial interpretation:

- $p_f = {x_1}^3$
- $p_g = 2x_1$
- $p_a = 1$

since we have the following strict inequality of functions: $((2x)^3)_{\mathbf{N}_+} > (2(x^3))_{\mathbf{N}_+}$, showing that $f(g(x)) >_p g(f(x))$.

Polynomial Termination Modulo Axioms

Some polynomial interpretations are compatible with certain axioms. For example, a symmetric polynomial such that $p(x, y) = p(y, x)$ is compatible with commutativity and can therefore be used to interpret a commutative symbol. For example, $2x + 2y$ is symmetric. Similarly, a polynomial $p(x, y)$ which is symmetric ($p(x, y) = p(y, x)$) and furthermore satisfies the associativity equation $p(x, p(y, z)) = p(p(x, y), z)$ can be used to interpret an associative-commutative symbol. As shown by Bencheriffa and Lescanne the polynomials satisfying these two conditions have a simple characterization: they must be of the form $axy + b(x + y) + c$ with $ac + b - b^2 = 0$.

The MTT Tool

The Maude Termination Tool (MTT) is a tool that can be used to prove the operational termination of Maude functional modules. In general, such modules can be conditional and may be not just order-sorted, but membership equational theories.

They may involve axioms like associativity and commutativity; and they may also have evaluation strategies (see Maude 2.2 manual, Section 4.4.7) indicating what arguments of a function symbol should be evaluated before applying equations for that symbol. For example, in an `if_then_else_fi` the first agument should be evaluated before equations for it are applied; and in a "lazy list cons" `_;_` the first argument is evaluated, but not the second.

The MTT Tool (II)

Features such as sorts, subsorts, memberships, and evaluation strategies may be essential for the termination of a Maude module. That is, ignoring them may result in a nonterminating module.

To preserve these features somehow, while still allowing using standard termination backend tools, the MTT implements the transformations of (Σ, E) first into an unsorted conditional theory $(\Sigma^{\circ}, E^{\circ})$, and then $(\Sigma^{\circ}, E^{\circ})$ is transformed into an unsorted unconditional theory $(\Sigma^{\bullet}, E^{\bullet})$.

If the module declares evaluation strategies, they are also transformed; but at the end evaluation strategies can either be used directly by a termination tool like Mu-Term, or a further theory transformation can eliminate such strategies.

The MTT Tool (III)

The course web page indicates where MTT has been installed. By typing: `./MTT` and carriage return the tool's GUI comes up and the user can interact with it. By using the `File` menu one can enter a Maude module into the tool.

Once a Maude module (enclosed in parentheses, and not importing any built-in modules) has been entered, the user can perform the theory transformation $(\Sigma, E) \mapsto (\Sigma^\bullet, E^\bullet)$ in one of three increasingly simpler modes: (1) **Complete**; (2) **No Kinds**; and (3) **No Sorts**. In case (2) kinds are ignored; and in case (3) both kinds and sorts are ignored. There is a tradeoff between simplicity of the transformation and its tightness. Sometimes a simpler transformation works better, and sometimes a more complete one is essentially needed.

The MTT Tool (IV)

The choice of transformation can be made by clicking the appropriate buttons (a screenshot will show this). But one also needs to choose which backend termination tool for unsorted and uncondional specifications will be used. One among the **CiME**, **MU-TERM**, and **AProVE** termination tools can be chosen.

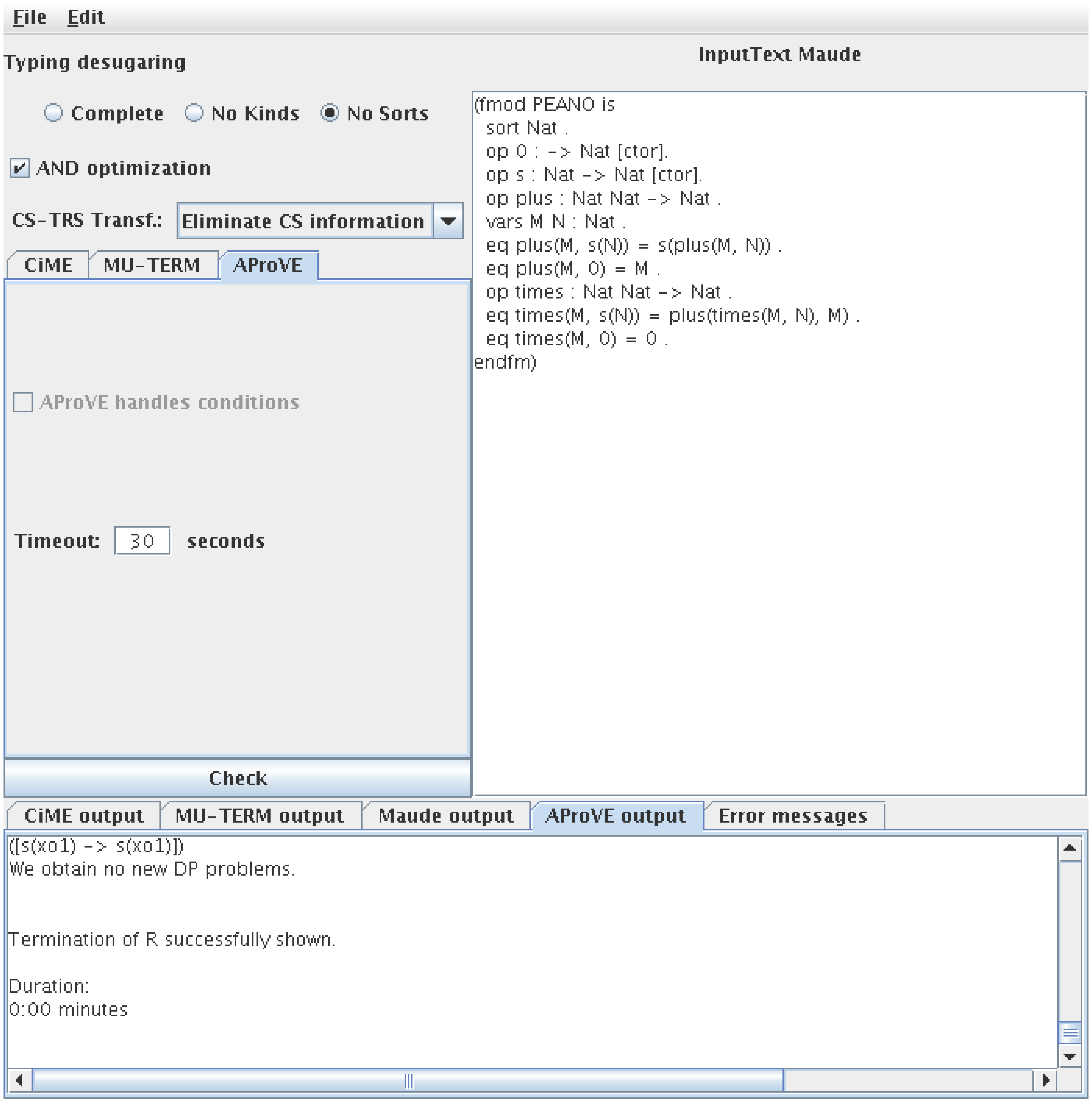
Then one can click on the `Check` bar to check the specification with the chosen tool. Some of these tools offer choices for different settings. So, we can try to prove termination using three different transformation variants, and then with one of three backend tools, sometimes customizing the particular tool choices. This maximizes the chances of obtaining a successful termination proof.

The MTT Tool (V)

What the tool then demonstrates is that the original Maude functional module is operationally terminating. The correctness of such a proof is based on:

- the correctness of the theory transformations (see paper in course web page); and
- the correctness of the chosen tools, that sometimes output a justification of how they proved termination.

A screeshot of a tool interaction is given in the next page.



( ) Complete ( ) No Kinds (●) No Sorts

☑ AND optimization

CS-TRS Transf.: Eliminate CS information

☐ AProVE handles conditions

Timeout: 30 seconds

```
(fmod PEANO is
  sort Nat .
  op 0 : -> Nat [ctor].
  op s : Nat -> Nat [ctor].
  op plus : Nat Nat -> Nat .
  vars M N : Nat .
  eq plus(M, s(N)) = s(plus(M, N)) .
  eq plus(M, 0) = M .
  op times : Nat Nat -> Nat .
  eq times(M, s(N)) = plus(times(M, N), M) .
  eq times(M, 0) = 0 .
endfm)
```

```
([s(xo1) -> s(xo1)])
We obtain no new DP problems.


Termination of R successfully shown.

Duration:
0:00 minutes
```

Termination is Undecidable

All the termination tools try to prove that a set of equations E, conditional or unconditional, is terminating by applying different proof methods; for example by trying to see if particular orderings can be used to prove the equations terminating.

But these termination proof methods are not decision procedures: in general termination of a set of equations (even if they are unconditional) is undecidable . However, termination is decidable for finite sets of unconditional equations E such that both the lefthand and the righthand sides are ground terms, or even if just the righthand sides are ground terms (see Chapter 5 in Baader and Nipkow, "Term Rewriting and All That", Cambridge U.P.).

Where to Go from Here

Besides RPO and polynomials there are various other orderings and a general “dependency pairs” method that can be used to prove termination. Good sources include:

TeReSe, “Term Rewriting Systems,” Cambridge U. P., 2003.

Baader and Nipkow, “Term Rewriting and All That”, Cambridge U.P., 1998.

N. Dershowitz and J.-P. Jouannaud, “Rewrite Systems,” in J. van Leeuwen, ed., “Handbook of Theoretical Computer Science,” Elsevier, 1990.

E. Ohlebusch, “Advanced Topics in Term Rewriting Systems,” Springer Verlag, 2002.