

# Program Verification: Lecture 9

José Meseguer

Computer Science Department  
University of Illinois at Urbana-Champaign

## Termination

We need methods to check termination of an equational theory  $(\Sigma, E)$ . 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  $\Sigma$ -monotonicity: for each  $f \in \Sigma$ , whenever  $f(t_1, \dots, t_n), f(t_1, \dots, t_{i-1}, t'_i, t_{i+1}, \dots, t_n) \in T_\Sigma(V)$  with  $t_i > t'_i$ , then,  $f(t_1, \dots, t_n) > f(t_1, \dots, t_{i-1}, t'_i, t_{i+1}, \dots, t_n)$
- closure under substitution: if  $t > t'$ , then, for any substitution  $\theta : V \longrightarrow T_\Sigma(V)$  we have,  $t\theta > t'\theta$ .

$>$  is called a **simplification order** if, furthermore,  $f(t_1, \dots, t_n) > t_i$ ,  $1 \leq i \leq n$ .

## Reduction Orderings (II)

Theorem: For  $(\Sigma, E)$  an equational theory,  $\vec{E}$  is terminating iff there exists a reduction order  $>$  s.t.  $\forall (u = v) \in E, u > v$ .

Proof:  $(\Rightarrow)$  follows from: (i)  $\vec{E}$  terminating makes  $\longrightarrow_E^+$  **irreflexive** and well-founded **order**, (ii) the **Context Lemma** implies  $\longrightarrow_E^+$  strictly monotonic, and (iii) the **Substitution Lemma** implies  $\longrightarrow_E^+$  closed under substitutions.

To see  $(\Leftarrow)$ , need to prove  $t \rightarrow_E t' \Rightarrow t > t'$ . By  $t \rightarrow_{\vec{E}} t'$ , there is a position  $p$ , equation  $(u = v) \in E$ , and substitution  $\theta$  s.t.  $t = t[u\theta]_p$ , and  $t' = t[v\theta]_p$ . We reason by induction on the length  $|p|$  of  $p$ .  
Base Case:  $p = \epsilon$ , i.e.,  $t = u\theta > v\theta = t'$ , by  $>$  substit. closed. Induction Step:  $p = i.q$  has length  $n + 1$ , with  $t = f(w_1, \dots, w_i, \dots, w_n)$ ,  $t' = f(w_1, \dots, w'_i, \dots, w_n)$ , with  $w_i \rightarrow_E w'_i$  at position  $q$ . By Ind. Hyp.  $w_i > w'_i$ , and by strict monotonicity  $t > t'$ . 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  $\Sigma$ , which is then extended to a **reduction ordering** on all terms. Since if  $\Sigma$  is finite the number of possible orderings between function symbols in  $\Sigma$  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:  $\_*\_ > \_+\_ > \mathbf{s}$ ) tends to work quite well, and can yield a reduction ordering containing the equations. Furthermore each symbol  $f$  in  $\Sigma$  is given a **status**  $\tau(f)$  equal to either:  $\tau(f) = \text{lex}(\pi)$  (lexicographic), or  $\tau(f) = \text{mult}$  (multiset).  $\tau(f)$  indicates how the **arguments** of  $f$  should be compared in the order.

## RPO (II)

Given a finite signature  $\Sigma$ , and an ordering  $>$  and a status function  $\tau$  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:

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

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

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  $\pi$  a permutation on  $n$  elements, then  $\langle u_1, \dots, u_n \rangle >_{rpo}^{\tau(f)} \langle t_1, \dots, t_n \rangle$  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  $\langle u_1, \dots, u_n \rangle >_{rpo}^{\tau(f)} \langle t_1, \dots, t_n \rangle$  iff we have  $\{u_1, \dots, u_n\} >_{rpo}^{mult} \{t_1, \dots, t_n\}$

where, given any order  $>$  on a set  $A$ , its 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 and Baader-Nipkow books cited later) that for a finite signature  $\Sigma$  RPO is a **simplification 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  $\tau(f) = \text{lex}(id)$  for each symbol  $f$ . Indeed, it is then trivial to check that  $n + 0 >_{rpo} n$  and  $n + s(m) >_{rpo} s(n + m)$ .



## 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  $f(a, b, c, d)$ . In his Ph.D. thesis,<sup>a</sup> Albert Rubio has further generalized *AC.RPO* to the *A ∨ C.RPO* order, where some symbols can be associative and/or commutative.

---

<sup>a</sup>A. Rubio, “Automated Deduction with Constrained Clauses,” Ph.D. thesis, Universitat Politècnica de Catalunya, 1994.

## Proving Termination with $A \vee C.RPO$

The Maude Termination Assistant (MTA) can prove termination modulo  $A \vee C$  axioms using an  $A \vee C.RPO$  reduction order.

To prove a functional module `F00` (preceded by: `set include BOOL off .`)  $A \vee C.RPO$ -terminating:

1. Choose a number  $n_f$  for each  $f \in \Sigma$  ( $f > g$  iff  $n_f > n_g$ ) using Maude's `metadata` attribute to specify  $n_f$  and `lex` in `F00`.
2. Load functional module `F00` in Maude; then load `mta.maude`.
3. Give the command (`check-AvCrpo F00 .`) that will check whether each  $u = v$  in `F00` satisfies  $u >_{A \vee C.rpo} v$ . It will reply: `Module is terminating by AvC-RPO order` or display those  $u = v$  in `F00` not provable with chosen order  $>$ .

MTA proves module `LIST+MSET` is  $AC.RPO$ -terminating:

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

```
set include BOOL off .
```

```
fmod LIST+MSET is
```

```
  sorts Element List MSet .  subsorts Element < List .
```

```
  subsorts Element < MSet .
```

```
  op a : -> Element [ctor metadata "1"] .
```

```
  op b : -> Element [ctor metadata "2"] .
```

```
  op c : -> Element [ctor metadata "3"] .
```

```
  op nil : -> List [ctor metadata "4"] .
```

```
  op _;_ : List List -> List [metadata "5 lex(2 1)"] .
```

```
  op _;_ : List Element -> List [ctor metadata "5 lex(2 1)"] .
```

```
  op _,_ : MSet MSet -> MSet [ctor assoc comm metadata "4"] .
```

```
  op null : -> MSet [ctor metadata "3"] .
```

```
  op l2m : List -> MSet [ctor metadata "5"] .
```

```
  vars L P Q : List .  var M : MSet .  var E : Element .
```

```
  eq L ; (P ; Q) = (L ; P) ; Q .  eq L ; nil = L .
```

```
  eq nil ; L = L .  eq M , null = M .  eq l2m(nil) = null .
```

```
  eq l2m(E) = E .  eq l2m(L ; E) = l2m(L) , E .
```

```
endfm
```

## 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**  $p_{\mathbb{N}_{\geq k}} : \mathbb{N}_{\geq k}^n \longrightarrow \mathbb{N}_{\geq k}$  (where  $k \geq 3$  and  $\mathbb{N}_{\geq k} = \{n \in \mathbb{N} \mid n \geq k\}$ ), just by evaluating the polynomial on a given tuple of numbers greater or equal to  $k$ . For  $p$  the polynomial above we have for example,  $p_{\mathbb{N}_{\geq k}}(3, 3, 3) = 383$ .

## Polynomial Orderings (II)

Note also that we can order the set  $[\mathbb{N}_{\geq k}^n \rightarrow \mathbb{N}_{\geq k}]$  of functions from  $\mathbb{N}_{\geq k}^n$  to  $\mathbb{N}_{\geq k}$  by defining  $f > g$  iff for each  $(a_1, \dots, a_n) \in \mathbb{N}_{\geq k}^n$   $f(a_1, \dots, a_n) > g(a_1, \dots, a_n)$ . 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  $(a_1, \dots, a_n) \in \mathbb{N}_{\geq k}^n$  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  $\Sigma$  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  $p_b \in \mathbb{N}_{\geq k}$ .

Suppose that in our set  $E$  of equations we have used variables  $Y = vars(E)$  (such variables need not be numbered at all). Then our assignment of a polynomial  $p_f$  on variables  $x_1, \dots, x_n$  to each function symbol of  $n$  arguments, and a number  $p_a$  to each constant  $a$  extends to a function:

## Polynomial Orderings (IV)

$$p_- : T_{\Sigma^u(Y)} \longrightarrow \mathbb{N}[Y]$$

where  $\Sigma^u(Y)$  is the unsorted version of  $\Sigma(Y)$ ,  $\mathbb{N}[Y]$  denotes the polynomials with natural number coefficients in the variables  $Y$ , and where  $p_-$  is defined inductively as follows:

- $p_b = p_b$
- $p_y = y$  for each  $y \in Y$
- $p_{f(t_1, \dots, t_n)} = p_f \{ x_1 \mapsto p_{t_1}, \dots, x_n \mapsto p_{t_n} \}$

## Polynomial Orderings (V)

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

$$t >_p t' \iff p_{t_{\mathbb{N}_{\geq k}}} > p_{t'_{\mathbb{N}_{\geq k}}}$$

where if  $Y = \text{vars}(E)$  and  $|Y| = m$ , linearly ordering  $Y$  we interpret  $p_{t_{\mathbb{N}_{\geq k}}}$  and  $p_{t'_{\mathbb{N}_{\geq k}}}$  as functions in  $[\mathbb{N}_{\geq k}^m \rightarrow \mathbb{N}_{\geq k}]$ . The relation  $>_p$  is clearly an irreflexive and transitive relation on terms in  $T_{\Sigma(X)} \subseteq T_{\Sigma^u(X)}$ , therefore a strict ordering, and is clearly well-founded, because otherwise we would have an infinite descending chain of polynomial functions in  $[\mathbb{N}_{\geq k}^m \rightarrow \mathbb{N}_{\geq k}]$ , which is impossible.



## Polynomial Orderings (VI)

We now need to check that this ordering is furthermore: (i) strictly  $\Sigma$ -monotonic, and (ii) closed under substitution. Condition (i) follows from  $+$  and  $*$  strictly monotonic on  $\mathbb{N}_{\geq k}$ , plus each function symbol  $f : s_1 \dots s_n \longrightarrow s$  in  $\Sigma$  the polynomial  $p_f$  involving **exactly** the variables  $x_1, \dots, x_n$  ( $p_f$  does not drop any variables and all coefficients are non-zero). Therefore,  $p_{f_{\mathbb{N}_{\geq k}}}$ , 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, left as an exercise, (where  $vars(t) = \{y_1, \dots, y_n\}$ ):

$$p(t\{y_1 \mapsto u_1, \dots, y_n \mapsto u_n\}) = p_t\{y_1 \mapsto p_{u_1}, \dots, y_n \mapsto p_{u_n}\}.$$

This then easily yields that if  $t >_p t'$  then  $t\{y_1 \mapsto u_1, \dots, y_n \mapsto u_n\} >_p t'\{y_1 \mapsto u_1, \dots, y_n \mapsto u_n\}$ , as desired.

## Polynomial Orderings (VII)

Therefore, polynomial interpretations of this kind define **reduction orders** and can be used to prove termination. One can also easily show them to be **simplification orders**. 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, e.g., the following polynomial interpretation:

- $p_f = x_1^3$
- $p_g = 2x_1$
- $p_a = 3$

since we have the following strict inequality of functions:

$$((2x)^3)_{\mathbb{N}_{\geq k}} > (2(x^3))_{\mathbb{N}_{\geq k}}, \text{ showing that } f(g(x)) >_p g(f(x)).$$

## Polynomial Termination Modulo Axioms

Some polynomial interpretations are compatible with certain axioms. For example, a **symmetric** polynomial, i.e., 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 associativity and commutativity axioms have a simple characterization: they must be of the form  $axy + b(x + y) + c$  with  $ac + b - b^2 = 0$ .

## Proving Polynomial Termination with MTA

The MTA tool can be used to prove polynomial termination of a module `F00` using **linear** polynomials. That is, we associate to each  $n$ -argument operator  $f \in \Sigma$  a linear polynomial of the form:

$$p_f = a_1x_1 + \dots + a_nx_n + a_{n+1}$$

where  $a_i \neq 0$  for  $1 \leq i \leq n$ . For constants  $c \in \Sigma$  we require  $p_c = a_1 \geq 2$ .

Using the `metadata` attribute, we express each  $p_f$  as the **string** " $a_1 \dots a_{n+1}$ ".

To prove polynomial termination we: (1) load into Maude `F00` with `metadata` annotations; then load `mta.maude`; then (2) give the command: `(check-poly F00 .)` MTA then replies with either `Module is terminating by polynomial order` or the list of equations failing the given order. Let us see an example:

## Proving Polynomial Termination with MTA (II)

```
set include BOOL off .
```

```
fmod LIST+MSET is
```

```
  sorts Element List MSet .      subsorts Element < List .
```

```
  subsorts Element < MSet .
```

```
  op a      : -> Element [ctor metadata "3"] .
```

```
  op b      : -> Element [ctor metadata "3"] .
```

```
  op c      : -> Element [ctor metadata "3"] .
```

```
  op nil    : -> List [ctor metadata "2"] .
```

```
  op _;_    : List List -> List [metadata "2 1 1"] .
```

```
  op _;_    : Element List -> List [ctor metadata "2 1 1"] .
```

```
  op _,_    : MSet MSet -> MSet [ctor assoc comm metadata "1 1 1"] .
```

```
  op null   : -> MSet [ctor          metadata "2"] .
```

```
  op l2m    : List -> MSet [ctor          metadata "1 1"] .
```

```
  vars L P Q : List .  var M : MSet .  var E : Element .
```

```
  eq (L ; P) ; Q = L ; (P ; Q) .          eq L ; nil = L .
```

```
  eq nil ; L = L .          eq M , null = M .          eq l2m(nil) = null .
```

```
  eq l2m(E) = E .          eq l2m(E ; L) = E , l2m(L) .
```

```
endfm
```

## Proving Polynomial Termination with MTA (III)

For an `assoc comm` (or `assoc comm id:`) symbol  $f$ , recall that the corresponding polynomial  $p_f$  must itself be `assoc comm` and therefore must have the form:  $axy + b(x + y) + c$  with  $ac + b - b^2 = 0$ . But since in MTA  $p_f$  must be **linear**, this forces  $a = 0$  and  $b = b^2$ . Therefore,  $p_f = 1x + 1y + c$ . That is why we have declared:

```
op _,_ : MSet MSet -> MSet [ctor assoc comm metadata "1 1 1"] .
```

Note that if  $f$  is only `assoc` (or `assoc id:`) it is OK for  $p_f$  to be `assoc comm`, since in particular  $p_f$  **is** `assoc`. Therefore, for an `assoc` symbol  $f$  we must also choose  $p_f = 1x + 1y + c$ .

Note: We do not need to worry about  $p_f$  satisfying `id:` axioms: MTT automatically generates a **semantically equivalent** module where `id:` axioms become **rules**, so  $p_f$  need only be `assoc comm`.

## 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 the Maude 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 argument 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  $(\Sigma, E)$  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 unconditional 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 screenshot of a tool interaction is given in the next page.

**File Edit**

**Typing desugaring** **InputText Maude**

Complete
  No Kinds
  No Sorts

AND optimization

CS-TRS Transf.: **Eliminate CS information** ▼

AProVE handles conditions

Timeout:  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(x01) -> s(x01)])
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**. This is so because a Turing machine can be specified as a term rewriting system (TRS) (see Ölveczky's, and Baader and Nipkow's books in next slide) and the **halting** (=termination) problem is undecidable for Turing machines. However, there are some TRS classes (e.g., **ground** TRSs) for which termination **is** decidable (see Baader and Nipkow's book in next slide).

## Where to Go from Here

Besides RPO and polynomials, other orderings and termination methods 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.

P. Ölveczky, “Designing Reliable Distributed Systems,” Springer Verlag, 2017.