The Maude Termination Assistant

Raúl Gutiérrez[1], José Meseguer[2] and Stephen Skeirik[2]

[1] Universitat Politècnica de València
[2] University of Illinois at Urbana-Champaign

Abstract. We present the Maude Termination Assistant (MTA). Its purpose is to complement the fully automatic Maude Termination Tool (MTT) [2] in two ways: (i) to help users find a proof when MTT may not find one; and (ii) for teaching purposes. The main features supported by MTA are: (1) termination proofs of order-sorted equational theories, which may include conditional rules without extra variables; (2) such proofs can be *modulo* any combination of associativity and/or commutativity and/or identity axioms; (3) identity axioms are automatically transformed into rules by the semantics-preserving theory transformation in [4]; (4) support for user-specified *recursive path orders* modulo any combination of the above axioms; and (5) support for user-specified *linear polynomial orders* modulo any combination of the above axioms.
Keywords: RPO termination modulo, polynomial termination modulo, term rewriting, Maude.

1 Introduction

Termination proofs are required for many formal verification purposes. Maude has automated support for termination proofs thanks to its MTT tool [2]. MTT works as follows: (1) First, the input module/theory is *transformed* in suitable ways [3] to deal with issues such as subsorts, memberships, equations versus rules, and conditional equations/rules. *Non-termination-preserving* transformations are applied that guarantee termination of the original specification if the transformed one is terminating. (2) Then, termination tools such as AProVE [5] or MU-TERM [1] are invoked as backends. These backend state-of-the-art tools are fully automated and use sophisticated *dependency pair* techniques [10]. But, since termination is undecidable and the tools have timeouts and other restrictions, some specifications cannot be proved terminating this way.

At present, if that happens (e.g., for `IDEMPOTENT-SEMIGROUP` in Section 3, or for the example in Appendix A), no further assistance can be provided by Maude tools: the user has to find other ways of proving the specification terminating. But since some of the best automated tools have already been tried, more direct user intervention is often needed. Could some tool assistance be provided? And if so, what should such assistance be like? An almost *sine qua non* requirement is support for termination proofs *modulo* any combination of associativity and/or commutativity and/or identity axioms: such axioms are heavily used in many Maude specifications, so that a tool not supporting them would be of marginal interest. Another requirement in a more interactive mode is placing *limited demands* on the user's expertise about termination techniques. For example, dependency pairs and their termination proofs are quite complex, and it would be hard for an average user to use them. A third *desideratum* is support for conditional rules, which are used quite often in Maude specifications. The MTA is of course not the most powerful such tool possible; but it meets these three requirements reasonably well: (i) it supports termination proofs of order-sorted Maude functional modules or theories modulo *any* combination of associativity and/or commutativity and/or identity axioms; (ii) it does so for two well-known termination proof methods that are quite easy to use, yet, remarkably effective in proving termination for many examples, namely, RPO termination orders modulo the above axioms, and polynomial termination orders also modulo the above axioms; and (iii) supports proofs of conditional termination in the most common case, which is also the easiest for a user to handle, namely, rules without extra variables in their righthand sides or conditions. One additional advantage —which actually stimulated us to develop MTA— is its usefulness for *teaching* termination techniques to students in formal verification courses. Since RPO and polynomial termination methods are among the most basic, a first introduction to rewriting termination techniques should cover them. But there are two practical problems: first, without some suitable tool no *hads-on* experience can be provided to students; second, since students, at least in the courses of this kind provided at the University of Illinois, become familiar with Maude and its rewriting modulo axioms, such experience should allow them to prove terminating specifications that use such axioms. The teaching experience with two prototype versions of MTA in the last two years has been quite encouraging and has motivated the present, more mature, general, and user-friendly version.

2 Theoretical Foundations

Termination Modulo $A \vee C \vee U$ Axioms. The Maude specifications handled by MTA are functional modules or theories of the form `fmod` $(\Sigma, E \uplus B)$ `endfm`, or `fth` $(\Sigma, E \uplus B)$ `endfth`, which are regarded as rewrite theories $(\Sigma, B, \vec{E})$ with the equations E oriented as rewrite rules, and where the axioms B can be any combination of associativity (A), commutativity (C) or identity (U) axioms. E may contain conditional equations of the form $t = t'$ *if* $\bigwedge_i u_i = v_i$, but such that $vars(t') \cup vars(\bigwedge_i u_i = v_i) \subseteq vars(t)$. However, conditional rules are *transformed* into unconditional ones as explained below. Therefore, we need only focus on explaining the kinds of *unconditional* termination proofs supported by MTA. All of them are based on using a reduction ordering $>$ [10] on terms such that $t > t'$ for each rule $t \to t'$ in $\vec{E}$, where $>$ is *compatible* with the axioms B. This exactly means that the termination order is also defined on B-equivalence classes as, $[t]_B > [t']_B$. Identity axioms U are well-known to be problematic for reduction orders. However, using the transformation in [4], the rewrite theory $(\Sigma, B, \vec{E})$ can be transformed into a *semantically equivalent* one of the form $(\Sigma, B_0, \vec{\hat{E}} \cup \vec{U})$, where $B = B_0 \uplus U$ is a decomposition into $A \vee C$-axioms B_0 and U-axioms U, which are now oriented as rules $\vec{U}$. This tranformation is applied at the outset by MTA, so that the reduction orders used need only be $A \vee C$-compatible. Two kinds of orders can be defined by an MTA user: $A \vee C$-compatible *RPO* orders, or an $A \vee C$-compatible *linear polynomial* orders.

$A \vee C$ **RPO**. RPO is a well-know *simplification order* parametric on an order $f > g$ on function symbols [10]. Various versions of an AC-compatible RPO order are well-known. What is perhaps less well-known, and is used crucially by MTA, is that for *any* combination B_0 of A and/or C axioms, and an ordering on symbols in a finite Σ, a B_0-compatible RPO order can be defined, which furthermore becomes a *total* order on ground terms modulo B_0, i.e., on T_{Σ/B_0}. The definition of all such $A \vee C$-compatible RPO orders and the proof of all these properties is one of the important contributions of A. Rubio's Ph.D. thesis [8].

Linear Polynomial Termination. For B_0 any $A \vee C$ axioms, call $(\Sigma, B_0, \vec{E})$ $\mathbb{N}$-*polynomially terminating* iff each $f \in \Sigma$ can be associated a polynomial p_f with natural coefficients and exactly as many distinct variables as arguments in f such that there exists a bound $b \in \mathbb{N}$, $b \geqslant 2$, such that: (i) for all constants a in Σ, $p_a \geqslant b$, (ii) for each $u = v$ in E $p_u > p_v$ as multi-argument functions on $\mathbb{N}_{\geqslant b}$, where p_t is the homomorphic extension to a term t of the map $f \mapsto p_f$, and (iii) if $f \in \Sigma$ satisfies some A and/or C axiom in B_0, then p_f does so too as a polynomial function on $\mathbb{N}_{\geqslant b}$. Under such conditions the assignment $f \mapsto p_f$ defines a B_0-compatible reduction order (see [10] for the case $B_0 = \varnothing$). There is, however, one problem blocking full automation and needing heuristics, namely, that given $(\Sigma, B_0, \vec{E})$ it is in general *undecidable* whether a polynomial order of this kind exists or not. This is because the problem can be reduced to a general one of Diophantine equation solvability (see, e.g., [10]), which is well-known to be undecidable by Matijasevich's Theorem on Hilbert's 10^{th} Problem. MTA adopts a simple approach to such an automation problem. First of all note that a terminating $\mathbb{N}$-polynomial order exists iff the formula

$$(\exists z) \bigwedge_{a \in \Sigma} z \geqslant p_a \wedge (\forall \overline{x}) \bigwedge_{i} x_i \geqslant z \Rightarrow \bigwedge_{u=v \in E} p_u > p_v$$

is valid in $\mathbb{N}$, where $\overline{x} = x_1 \ldots x_k$ are the variables appearing in the equations E, and we assume that different equations in E have *disjoint* variables. But if the p_f are all *linear polynomials*, then the above formula is in Presburger arithmetic and its validity is therefore *decidable*. MTA does not use a decision procedure for Presburger arithmetic (which in this case would require quantifier elimination). It uses instead a simple special-purpose decision procedure written in Maude to decide validity of the above formula for the given polynomial interpretation. Finally, $A \vee C$-compatibility is easy to check for linear polynomials (more later).

Termination of Conditional Rules. Conditional equations $t = t'$ *if* $\bigwedge_i u_i = v_i$ with the above-mention requirement on variables and oriented as rules with their conditions interpreted as *joinability* conditions can be proved terminating modulo B_0 if there is a B_0-compatible simplification order $>$ such that $t > t'$, $t > u_i$ and $t > v_i$ for each i [7]. This is achieved in MTA by: (i) transforming the above rule into the rules $t \to t'$, $t \to u_i$ and $t \to v_i$ for each i, and (ii) ensuring $>$ is a B_0-compatible *simplification* order, which holds automatically for $A \vee C$

RPO, and does hold for linear polynomial orders if for each unary f, $p_f = ax + b$ is such that either $a > 1$ or if $a = 1$ then $b \geqslant 1$. MTA checks this last condition if a polynomial order is used for a module with conditional equations.

3 MTA Syntax, Interaction, and Implementation

The *syntax* required for using the MTA is quite simple. It has two aspects: (i) *metadata* notation (more later) for a user to specify for each function symbol or constant in module or theory, say `FOO`, the information needed to *define* either an $A \vee C$ RPO order or a linear polynomial order; and (ii) the respective check command syntax to be given after `FOO` and the tool have been loaded into Maude.

The user interaction is also quite simple. The user: (i) loads his/her module or theory `FOO` in Maude with the desired meta-data annotations (we assume `FOO` *makes no use of built-in modules*); (ii) loads the file `mta.maude` containing the MTA tool implementation and providing a simple user interface as a Full Maude extension; (iii) types the command `(check-AvCrpo FOO .)` if the defined order is $A \vee C$ RPO, or `(check-poly FOO .)` if it is a linear polynomial order; and (iv) then receives output from the tool, either confirming that the termination check was successful or listing the equations that could not be proved terminating. If the check is unsuccessful, the user can easily modify the given order by changing some metadata annotations or try a different kind of order.

Proving $A \vee C$ RPO Termination. Consider the following module specifying both lists and multisets and a `l2m` function transforming a list into a multiset. To avoid default inclusion of the `BOOL` built-in module, note the command `set include BOOL off .`, which should always precede any module.

```
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
```

An $A \vee C$ RPO order is here defined by a function *level* : $\Sigma \rightarrow \mathbb{N}$ and each symbol's lexicographic status, if any. *level* defines an order on symbols by

$f > g \Leftrightarrow \mathit{level}(f) > \mathit{level}(g)$. For each $f \in \Sigma$, $\mathit{level}(f) = k$ is specified by adding to f the declaration: `metadata` "k". If f does not satisfy any $A{\vee}C$ axioms, a *lexicographic* status [10] can be specified by adding: `lex`$(i_1 \ldots i_n)$, where $(i_1 \ldots i_n)$ is a permutation of f's n arguments, as done above for the $\_;\_$ symbol. After loading `LIST+MSET` and `mta.maude`, the command `(check-AvCrpo LIST+MSET .)` is answered with output: `Module is terminating by AvC-RPO order`.

Proving $A \vee C$ Polynomial Termination. The theory of idempotent semigroups is conjectured impossible to specify by *unconditional* confluent and terminating rules in [9]. The authors give a locally confluent (modulo $A \vee C$) *conditional* specification expressible in Maude as the following *functional theory*:

```
set include BOOL off .

fth IDEMPOTENT-SEMIGROUP is
  sorts Semigroup Set .
  op _ _ : Semigroup Semigroup -> Semigroup [assoc metadata "1 1 10"] .
  op _,_ : Set Set -> Set  [assoc comm metadata "1 1 0"] .
  op {_} : Semigroup -> Set  [metadata "1 1"].
  var S : Set . vars L P Q : Semigroup .
  eq L L   = L .
  ceq L P Q = L Q if {L} = {Q}  /\ {L P} = {L} .
  eq S , S = S .
  eq {L P} = {L} , {P} .
endfth
```

The key idea is to use the function `{_}` to turn semigroup elements into set elements and use it in the second equation's condition, so that the original idempotency equation *plus* the added conditional equation make the theory locally confluent. To show that the `IDEMPOTENT-SEMIGROUP` specification makes the theory *decidable* we need to prove its *operational termination.* This can be done in MTA using a linear polynomial order. The map $p_{\_} : f \mapsto p_f = a_1x_1 + \ldots a_nx_n + a_{n+1}$ for each $f \in \Sigma$ with n arguments is specified in Maude by declaring the operator f with the metadata attribute: `metadata` "$a_1 \ldots a_n\, a_{n+1}$" subject to the following requirements: (i) all $a_1, \ldots, a_n$ are nonzero, (ii) for a constant c we must have $p_c = a_1 \geqslant 2$, (iii) for f a binary symbol with $p_f = a_1x_1 + a_2x_2 + a_3$, if f is commutative, p_f must be a *symmetric* polynomial, so that $a_1 = a_2$, and if f is *associative* (with or without commutativity) we must have $a_1 = a_2 = 1$. Furthermore, if, as in this case, some equations are conditional, to make the order a *simplification order* [10], (iv) if f is unary with $p_f = a_1x_1 + a_2$ we must either have $a_1 > 1$ or if $a_1 = 1$ then $a_2 \geqslant 1$. Note that all these requirements are met by the above meta-data annotations. After loading into Maude the above theory and `mta.maude` and giving the command `(check-poly IDEMPOTENT-SEMIGROUP .)` we get the output: `Module is terminating by polynomial order`.

Reflective Maude Implementation. Since MTA manipulates *terms* and *theories*, its most natural implementation is a *reflective* one extending Maude's `META-LEVEL`, where terms and theories are available as data elements of respective sorts `Term` and `Module`. The MTA implementation extends Full Maude and has three components: (i) an $A \vee C$ RPO *library*, which is of independent interest for resolution theorem proving modulo $A \vee C$, e.g., [8], or for Knuth-Bendix completion modulo $A \vee C$, (ii) a *decision procedure* for linear polynomial termination; and (iii) a simple *user interface* accepting user commands to check termination and reporting answers and errors. The MTA implementation with some examples can be found in `http://maude.cs.illinois.edu/tools/mta/`.

4 Conclusions and Future Work

MTA complements the fully automatic MTT Tool [2] to assist users when MTT cannot prove terminating a functional module or theory having $A \vee C \vee U$ axioms and possibly with conditional equations without extra variables in conditions or righthand sides. It also complements AGES [6], a quite general tool that can prove polynomial termination of conditional Maude modules with subsorts, but without axioms. MTA can also be used as a teaching tool to give students a hands-on experience in defining RPO or polynomial orders modulo axioms. After more experimentation, MTA could be extended in several ways, including: (i) support for termination proofs of Maude *system* modules and theories; (ii) support for polynomial termination proofs using *non-linear* polynomials; and (iii) integration with MTT and other tools within the Maude Formal Environment.

References

1. Alarcón, B., Gutiérrez, R., Lucas, S., Navarro-Marset, R.: Proving Termination Properties with MU-TERM. In: Johnson, M., Pavlovic, D. (eds.) Proc. AMAST'10. LNCS, vol. 6486, pp. 201–208. Springer (2011)
2. Durán, F., Lucas, S., Meseguer, J.: MTT: The Maude Termination Tool (system description). In: IJCAR 2008. Lecture Notes in Computer Science, vol. 5195, pp. 313–319. Springer (2008)
3. Durán, F., Lucas, S., Meseguer, J.: Methods for proving termination of rewriting-based programming languages by transformation. Electr. Notes Theor. Comput. Sci. 248, 93–113 (2009)
4. Durán, F., Lucas, S., Meseguer, J.: Termination modulo combinations of equational theories. In: Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009, Trento, Italy, September 16-18, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5749, pp. 246–262. Springer (2009)
5. Giesl, J., Aschermann, C., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Hensel, J., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Analyzing program termination and complexity automatically with AProVE. Journal of Automated Reasoning 58(1), 3–31 (2017)
6. Gutiérrez, R., Lucas, S., Reinoso, P.: A tool for the automatic generation of logical models of order-sorted first-order theories (2016)
7. Kaplan, S.: Simplifying conditional term rewriting systems: Unification, termination and confluence. J. Symb. Comput. 4(3), 295–334 (1987)
8. Rubio, A.: Automated Deduction with Constrained Clauses. Ph.D. thesis, Universitat Politècnica de Catalunya (1994)
9. Siekmann, J., Szabó, P.: A Noetherian and confluent rewrite system for idempotent semigroups. Semigroup Forum 25, 83–110 (1982)
10. TeReSe: Term Rewriting Systems. Cambridge University Press (2003)

A An Example

As mentioned in the Introduction, the IDEMPOTENT-SEMIGROUP theory in Section 3 cannot be proved terminating by MTT. This is because MTT does not handle associative but non-commutative axioms. Likewise, MTT cannot prove the ABSTRACT-BAKERY module below terminating, but MTA can prove its $A{\vee}C$-RPO termination (in this second case MTT seems to have no problem with axioms).

Abstract Bakery. This functional module specifies an equational abstraction for a two-process version of Lamport's Bakery Mutual Exclusion Protocol. Bakery is an infinite-state system; so it cannot be model checked by explicit-state techniques. The ABSTRACT-BAKERY functional module below:

```
set include BOOL off .

fmod NAT> is
  sorts Zero NzNat Nat Bool .
  subsorts Zero NzNat < Nat .
  op tt : -> Bool [ctor metadata "1"] .  *** true
  op ff : -> Bool [ctor metadata "2"] .  *** false
  op 0 : -> Zero [ctor metadata "3"] .
  op 1 : -> NzNat [ctor metadata "4"] .
  op _+_ : Nat Nat -> Nat [ctor assoc comm id: 0 metadata "12"] .
  op _+_ : NzNat Nat -> NzNat [ctor assoc comm id: 0 metadata "12"] .
  op _+_ : Nat NzNat -> NzNat [ctor assoc comm id: 0 metadata "12"] .
  op _+_ : NzNat NzNat -> NzNat [ctor assoc comm id: 0 metadata "12"] .
  op _>_ : Nat Nat -> Bool [metadata "13"] .

  vars n m : Nat .  vars n' m' : NzNat .

  eq n + n' > n = tt .
  eq n > n + m = ff .
endfm

fmod BAKERY is
  protecting NAT> .

  sorts Mode BState .

  op sleep     :                   -> Mode   [ctor metadata "5"] .
  op wait      :                   -> Mode   [ctor metadata "6"] .
  op crit      :                   -> Mode   [ctor metadata "7"] .

  op <_,_,_,_> : Mode Nat Mode Nat -> BState [ctor metadata "14"] .
endfm

fmod BAKERY-PREDS is
  protecting BAKERY .
  sort State Prop .
  subsort BState < State .
  ops 1wait    :                   -> Prop   [ctor metadata "8"] .
  ops 2wait    :                   -> Prop   [ctor metadata "9"] .
  ops 1crit    :                   -> Prop   [ctor metadata "10"] .
  ops 2crit    :                   -> Prop   [ctor metadata "11"] .
  op _|=_      : State Prop        -> Bool   [frozen ctor metadata "15"] .

  vars P Q : Mode .
  vars X Y : Nat .

  eq < wait, X, Q, Y > |= 1wait = tt .
  eq < sleep, X, Q, Y > |= 1wait = ff .
  eq < crit, X, Q, Y > |= 1wait = ff .
  eq < P , X, wait, Y > |= 2wait = tt .
  eq < P , X, sleep, Y > |= 2wait = ff .
  eq < P , X, crit, Y > |= 2wait = ff .
  eq < crit , X, Q, Y > |= 1crit = tt .
  eq < sleep, X, Q, Y > |= 1crit = ff .
  eq < wait, X, Q, Y > |= 1crit = ff .
  eq < P , X, crit, Y > |= 2crit = tt .
  eq < P , X, sleep, Y > |= 2crit = ff .
  eq < P , X, wait, Y > |= 2crit = ff .
endfm

fmod ABSTRACT-BAKERY is
  including BAKERY-PREDS .

  vars P Q : Mode .  vars X Y : Nat .  vars X' Y' : NzNat .

  eq < P, 0, Q, 1 + Y' > = < P, 0, Q, 1 >  .
  eq < P, 1 + X', Q, 0 > = < P, 1 , Q, 0 >  .
  eq < P, 1, Q, 1 + Y' + X' > = < P, 1, Q, 1 + 1 >  .
  eq < P, 1 + X' + Y', Q, 1 > = < P, 1 + 1, Q, 1 >  .
  eq < P, X' + 1, Q, Y' + X' + 1 > = < P, 1, Q, 1 + 1 > .
  eq < P, X' + Y' + 1, Q, Y' + 1 > = < P, 1 + 1, Q, 1 >  .
  eq < P, X' + Y' + 1, Q, X' + Y' + 1 > = < P, 1 + 1, Q, 1 + 1 > .
endfm
```

defines the semantics of state predicates using the equations in the `BAKERY-PREDS` submodule, and uses the last seven equations to identify system states (4-tuples with ticket number and a mode for each process) so as to make the system finite-state. This enables LTL model checking (with rewrite rules for system transitions in a Maude system module not shown here, and with the above-defined state predicates). The correctness of the associated LTL verification depends on checking several crucial proof obligations, including local confluence (done using Maude's CRC Tool), *and* termination of `ABSTRACT-BAKERY`. However, MTT does not succeed in finding a termination proof. But MTA can prove `ABSTRACT-BAKERY` terminating: `Module is terminating by AvC-RPO order.`

Note the somewhat interesting fact that the axioms for + in submodule `NAT>` include the identity axiom, which is outside the scope of $A \vee C$-RPO orders. Before $A \vee C$-RPO termination is checked for `ABSTRACT-BAKERY`, the module is transformed by MTA into a semantically equivalent module where such an identity axioms has been transformed into the equation $n + 0 = n$ and "variants" of all other equations using the rule $n + 0 \rightarrow n$ modulo AC have been computed according to the semantics-preserving transformation described in [4].