VMTL–A Modular Termination Laboratory

Felix Schernhammer[*] and Bernhard Gramlich

TU Wien, Austria
{felixs,gramlich}@logic.at

Abstract. The automated analysis of termination of term rewriting systems (TRSs) has drawn a lot of attention in the scientific community during the last decades and many different methods and approaches have been developed for this purpose. We present VMTL (Vienna Modular Termination Laboratory), a tool implementing some of the most recent and powerful algorithms for termination analysis of TRSs, while providing an open interface that allows users to easily plug in new algorithms in a modular fashion according to the widely adopted dependency pair framework. Apart from modular extensibility, VMTL focuses on analyzing the termination behaviour of conditional term rewriting systems (CTRSs). Using one of the latest transformational techniques, the resulting restricted termination problems (for unconditional context-sensitive TRSs) are processed with dedicated algorithms.

1 Introduction and Overview

During the last decade, remarkable progress has been made in the field of termination analysis of term rewriting systems. Despite termination being an undecidable property of TRSs, increasingly sophisticated methods have been developed to prove it for given systems. From these efforts several tools have emerged that are capable of proving termination (semi) automatically (a list of currently available tools can be obtained from the official website of the termination competition [18].)

The currently most powerful tools implement the dependency pair framework of [8] based on the idea of dependency pair analysis of [3]. This approach has at least two advantages. First, it seems to be the most powerful one, which is indicated by the latest results of a yearly termination competition ([18]). Second, the pure dependency pair framework is strictly modular, which means that concrete (correct) methods to prove termination within this framework can be combined, added and removed arbitrarily, without affecting the correctness of the tool. Thus, it is easy to extend tools implementing this framework by new methods (called *dependency pair processors* in the terminology of [8]).

VMTL (currently available in Version 1.1, cf. `http://www.logic.at/vmtl/`) is a new termination tool that implements the dependency pair framework and focuses on openness, modularity and extensibility. It is easily extensible by new

[*] The author has been supported by the Austrian Academy of Sciences under grant 22.361.

dependency pair processors, while providing the main technical infrastructure of termination tools such as thread and processor scheduling, timeout handling, input parsing, output formatting etc. In addition, VMTL contains implementations of several well-known methods of proving termination within the dependency pair framework. These include

- a dependency graph processor used for decomposing dependency pair problems into strongly connected components of an estimated dependency graph (the estimation described in [1, Section 4.1] is used),
- a reduction pair processor using recursive path orderings with status based on [1, Theorem 21],
- a reduction pair processor based on polynomial interpretations featuring linear and simple mixed polynomials, with coefficients from $\mathbb{N}$ and constants from $\mathbb{Z}$,
- forward and backward narrowing as well as instantiation processors (see below resp. [17, Section 6] for more details), and
- a size-change principle processor based on results from [19].

The set of implemented processors (in particular the inclusion of the size change principle processor) was chosen to optimize the power of VMTL with respect to proving termination of conditional term rewriting systems. Moreover, all of these processors are "context-restriction aware" in VMTL which means that they are sound for both context-sensitive and standard termination problems.

The two reduction pair processors are implemented via reduction to satisfiability problems and utilizing external SAT solvers, as it is state-of-the-art at the time of writing. Benchmarks of VMTL on the set of TRSs used in the latest termination competition can be found below.

Another focus during the development of VMTL was applicability to (and suitability for) conditional term rewriting systems (CTRSs). Termination of such CTRSs is usually verified by transforming them into ordinary TRSs and deriving termination of the CTRSs from termination of the transformed TRSs. VMTL provides a public interface that allows users to plug in such transformations. Moreover, it includes a recent one ([17]) that transforms CTRSs into context-sensitive (unconditional) TRSs.

VMTL is also capable of proving termination of context-sensitive term rewriting systems (CSRSs). For this task the refined context-sensitive dependency pair approach of [1] (cf. also [2]) is used. Using context-sensitive dependency pairs, and their property that they coincide with context-free (i.e., standard) dependency pairs for context-free (i.e., ordinary) term rewriting systems, allows VMTL to treat every TRS as CSRS and analogously treat every standard dependency pair problem as context-sensitive one. Still, the use of dedicated DP processors for each kind of problem is possible in VMTL (see Section 4.1 below).

2 Preliminaries

We assume familiarity with the basic concepts and notations of term rewriting and context-sensitive rewriting (cf. e.g. [4,5,12]). As we will briefly discuss

VMTL's approach to prove operational termination of deterministic conditional term rewriting systems (DCTRSs) in Section 5, we introduce some basic notions regarding conditional term rewriting.

Conditional rewrite systems consist of conditional rules $l \to r \Leftarrow c$, with c of the form $s_1 \to^* t_1, \ldots, s_n \to^* t_n$ such that l is not a variable and $Var(r) \subseteq Var(l) \cup Var(c)$. In the following we are concerned with *deterministic* 3-CTRSs (DCTRSs), which have the additional property that for each conditional rule $l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$ it holds that $Var(s_i) \subseteq Var(l) \cup \bigcup_{j=1}^{i-1} Var(t_j)$. Deterministic 3-CTRSs are arguably the most general class of CTRSs for which termination analysis makes sense, as in non-deterministic CTRSs arbitrary instantiations of extra variables usually entail that the system is not (effectively [16] / operationally [13]) terminating.

The conditional rewrite relation induced by a CTRS $\mathcal{R}$ is inductively defined as follows: $R_0 = \emptyset$, $R_{j+1} = \{\sigma l \to \sigma r \mid l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n \in \mathcal{R}, \sigma s_i \to^*_{R_j} \sigma t_i \text{ for all } 1 \leq i \leq n\}$, and $\to_{\mathcal{R}} = \bigcup_{j \geq 0} \to_{R_j}$. In contrast to ordinary term rewriting systems well-foundedness of the induced rewrite relation is not an adequate notion of termination for CTRSs. The proof-theoretic notion of *operational termination* has turned out to be adequate to guarantee finiteness of derivations in CTRSs (cf. [13,17] for details).

Given a CSRS $\mathcal{R} = (\Sigma, R)$ with replacement map μ, the relation of context-sensitive narrowing (written $\leadsto^{\mu}_{\mathcal{R}}$) is defined as $t \leadsto^{\mu}_{\mathcal{R}} s$ if there is a replacing non-variable position p in t such that $t|_p$ and l unify (where $l \to r \in R$ and where we assume that t and l do not share any variables) with mgu θ and $s = \theta(t[r]_p)$. We say that s is a *one-step, context-sensitive narrowing* of t.

2.1 The Context-Sensitive Dependency Pair Framework

The dependency pair framework of [8] resp. [1] basically reduces termination problems to problems of proving finiteness of so-called *dependency pair problems.*

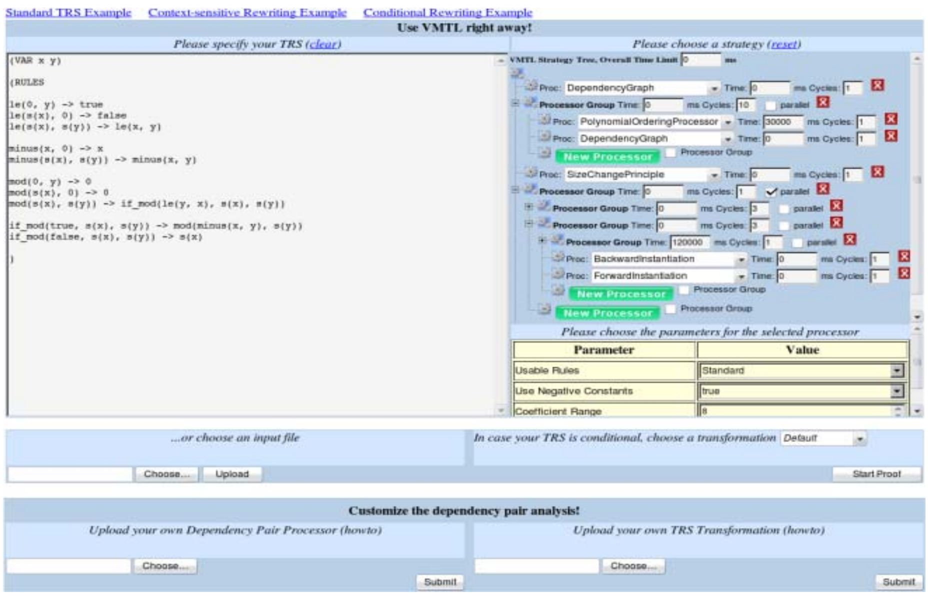
We call a triple $(P, \mathcal{R}, \mu)$, where P and $\mathcal{R}$ are TRSs and μ is a replacement map for the combined signatures of P and $\mathcal{R}$, a *(context-sensitive) dependency pair problem* (CS-DP-problem).[1] Such a problem is finite if P terminates *relative* to $(\mathcal{R}, \mu)$ where $\to_P$ steps may only occur as root steps and $\to_{\mathcal{R},\mu}$ steps may only occur strictly below the root.

Within the dependency pair framework these problems are analyzed by so-called *dependency pair processors.*

A *CS-dependency pair processor* is a function *Proc* that takes as input a CS-DP-problem and returns either a set of CS-dependency pair problems or "no". We call a CS-DP-processor *sound* if finiteness of all CS-DP-problems in *Proc*(d) implies finiteness of the input CS-DP-problem d. A CS-DP-processor is *complete* if for all CS-DP-problems d, d is infinite whenever *Proc*(d) is "no" or *Proc*(d) contains an infinite CS-DP-problem.

For further details regarding the dependency pair framework we refer to [8].

[1] Initially, P consists of the (context-sensitive) dependency pairs of $(\mathcal{R}, \mu)$ or $\mathcal{R}$, respectively, cf. [1,3].



Standard TRS Example Context-sensitive Rewriting Example Conditional Rewriting Example

Use VMTL right away!

Please specify your TRS (clear)

```
(VAR x y)

(RULES

le(0, y) -> true
le(s(x), 0) -> false
le(s(x), s(y)) -> le(x, y)

minus(x, 0) -> x
minus(s(x), s(y)) -> minus(x, y)

mod(0, y) -> 0
mod(s(x), 0) -> 0
mod(s(x), s(y)) -> if_mod(le(y, x), s(x), s(y))

if_mod(true, s(x), s(y)) -> mod(minus(x, y), s(y))
if_mod(false, s(x), s(y)) -> s(x)

)
```

Please choose a strategy (reset)

```
VMTL Strategy Tree, Overall Time Limit 0 ms
  Proc: DependencyGraph  Time: 0 ms Cycles: 1
  Processor Group Time: 0 ms Cycles: 10  parallel
    Proc: PolynomialOrderingProcessor  Time: 30000 ms Cycles: 1
    Proc: DependencyGraph  Time: 0 ms Cycles: 1
    New Processor  Processor Group
  Proc: SizeChangePrinciple  Time: 0 ms Cycles: 1
  Processor Group Time: 0 ms Cycles: 1  ✓ parallel
    Processor Group Time: 0 ms Cycles: 3  parallel
    Processor Group Time: 0 ms Cycles: 3  parallel
      Processor Group Time: 120000 ms Cycles: 1  parallel
      Proc: BackwardInstantiation  Time: 0 ms Cycles: 1
      Proc: ForwardInstantiation  Time: 0 ms Cycles: 1
      New Processor  Processor Group
    New Processor  Processor Group
```

Please choose the parameters for the selected processor

Parameter	Value
Usable Rules	Standard
Use Negative Constants	true
Coefficient Range	8

...or choose an input file Choose... Upload

In case your TRS is conditional, choose a transformation Default

Start Proof

Customize the dependency pair analysis!

Upload your own Dependency Pair Processor (howto) Choose... Submit

Upload your own TRS Transformation (howto) Choose... Submit

Fig. 1. Snapshot of the VMTL web interface after specifying a particular proof strategy

3 User Interface

VMTL provides a command line interface for batch execution and a web interface that eases configuration and extension. Figure 1 shows a screenshot of the web interface, after specifying a particular proof strategy. It contains fields for entering a TRS or alternatively uploading a file. VMTL exclusively accepts the input format specified for the termination competition. On the right-hand side the user can define the strategy, i.e., the order in which processors are applied, together with time constraints to be satisfied. At the bottom there are two fields allowing the user to upload new customized dependency pair processors and transformations, respectively.

3.1 User Defined Strategies

Since dependency pair processors often modify and simplify dependency pair problems, it is crucial to apply them in a reasonable order. Moreover, some processors are more time consuming than others. Thus, in order to achieve the goal of proving termination as quickly as possible, it is important to have a good strategy for processor application. VMTL allows the user to fully configure this strategy. It provides the following degrees of freedom in its specification.

– Dependency pair processors can be arbitrarily ordered.
– Time limits can be imposed on dependency pair processors.

– Dependency pair processors can be executed repeatedly (which can be helpful for instance for *narrowing* processors).
– Dependency pair processors can be hierarchically grouped to an arbitrary depth. Each group can be given a time limit. In case of contradicting time limits the shortest one is used.
– Execution of (groups of) dependency pair processors can be parallelized.

The semantics of parallelism in VMTL is that if one of the parallel branches finishes the termination (or non-termination) proof, the whole execution stops immediately and the proof is presented to the user. Otherwise, if all parallel branches fail to prove termination, the termination proof continues according to the strategy with the dependency pair problems derived *before* the start of the parallel execution or with any of the problems derived in the parallel paths, depending on which of these problems is the "most simple" one. This is determined by a configurable measure function on dependency pair problems. By default, this function compares the size of the problems to determine the "most simple" one.

Graphically, a strategy is represented in VMTL as a tree, where each node can either be a dependency pair processor, or a "Group node". Group nodes are used to build groups of processors or other groups, cf. Figure 1.

VMTL provides a default strategy that turned out to be a reasonable compromise between power and efficiency, which allows users to test systems for termination without manually specifying a proof strategy. This standard strategy was also used in the benchmarks below.

4 VMTL API

VMTL provides a public java programming interface that allows a user to easily build extensions. There are three basic functionalities that can be extended.

– New dependency pair processors can be added.
– New transformations, from (conditional/context-sensitive) TRSs to (context-sensitive) TRSs, can be added.
– New plug-ins for output formatting can be added.

4.1 Adding New Dependency Pair Processors

VMTL provides two interfaces *DPProcessor* and *ContextSensitiveDpProcessor* from which one has to be implemented by the user depending on whether the processor takes context-restrictions into account or not. In case DPProcessor is implemented, VMTL will make sure that the processor is not applied to context-sensitive dependency pair problems (even if the processor occurs in the strategy). Note that each context-sensitive DP processor is trivially a context-free one (as context-free DP problems can be seen as special cases of context-sensitive ones), thus ContextSensitiveDpProcessor is a "subinterface" (in an object oriented sense) of DPProcessor. See e.g. [10, Example 3] for a justification that context-free dependency pair processors (for instance using *usable rules*) may in general not be applied to context-sensitive dependency pair problems.

Technically, a user-defined processor is given a dependency pair problem (that has possibly been processed by other processors before) and a set of (processor) parameters from VMTL and is required to return a set of derived dependency pair problems. In VMTL, the datastructure of a dependency pair problem consists of two sets of rewrite rules and a replacement map according to the definition in Section 2.1. Additionally, in VMTL it contains a subsignature (cf. [17, Section 6]) and several additional flags (in particular one for position-based strategies such as *innermost rewriting* and one for completeness indicating whether infinity of the dependency pair problem implies non-termination of the initial rewrite system).

4.2 Adding New Transformations

Adding transformations to VMTL can be accomplished by implementing the interface *TrsToTrsTransformation*. Transformations in VMTL are not restricted to ones that transform conditional systems. The interface can be used to perform arbitrary preprocessing steps, such as semantic labelling etc. (this is the reason why the interface is not called *CtrsToTrsTransformation*). However, in case termination of a conditional rewrite system is to be proved, a transformation is mandatory. If none is specified by the user, VMTL will use the context-sensitive unraveling transformation of [6,15,17].

4.3 Customizing Output Formatting

The proof information, that is accumulated by VMTL, and the used dependency pair processors are represented in a simple native markup language, providing basic structuring and formatting tags. From this intermediate representation the actual (human or machine) readable output is created by so-called *OutputWriter* objects. Out of the box, VMTL only supports HTML output. However, the user may extend VMTL by additional OutputWriters through the *OutputWriter* interface and can thereby provide additional support for proof certification (see e.g. [11]).

5 Termination of CTRSs

In ([17]) it was shown that when verifying (operational) termination of a CTRS by transformation, it is sufficient to prove termination of the transformed system only on a restricted set of terms. This idea is incorporated in VMTL by extending (context-sensitive) dependency pair problems as defined in [8] by an additional component representing a sub-signature, which identifies the set of terms for which termination is to be shown.

In VMTL the following context-sensitive version ([17]) of an unraveling transformation ([14,16]) is included.

Definition 1. *(context-sensitive unraveling of DCTRSs) Let $\mathcal{R}$ be a DCTRS ($\mathcal{R} = (\Sigma, R)$). For every rule $\alpha\colon l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$ we use n new function symbols U_i^α ($i \in \{1, \ldots, n\}$). Then α is transformed into a set of unconditional rules in the following way:*

$$\begin{aligned} l &\to U_1^\alpha(s_1, Var(l)) \\ U_1^\alpha(t_1, Var(l)) &\to U_2^\alpha(s_2, Var(l), \mathcal{E}Var(t_1)) \\ &\vdots \\ U_n^\alpha(t_n, Var(l), \mathcal{E}Var(t_1), \ldots, \mathcal{E}Var(t_{n-1})) &\to r \end{aligned}$$

Here $Var(s)$ denotes an arbitrary but fixed sequence of the set of variables of the term s. Let $\mathcal{E}Var(t_i)$ be $Var(t_i) \setminus (Var(l) \cup \bigcup_{j=1}^{i-1} Var(t_j))$. Abusing notation, by $\mathcal{E}Var(t_i)$ we mean an arbitrary but fixed sequence of the variables in the set $\mathcal{E}Var(t_i)$. Any unconditional rule of $\mathcal{R}$ is transformed into itself. The transformed system $U_{cs}(\mathcal{R}) = (U(\Sigma), U(R))$ is obtained by transforming each rule of $\mathcal{R}$ where $U(\Sigma)$ is Σ extended by all new function symbols. The replacement map $\mu_{U(\Sigma)}$ is given by $\mu_{U(\Sigma)}(f) = \{1, \ldots, ar(f)\}$ for all $f \in \Sigma$ and $\mu_{U(\Sigma)}(f) = \{1\}$ for all $f \in U(\Sigma) \setminus \Sigma$.

For this transformation it turns out that termination of the transformed TRS *on original terms*, i.e., on terms over the original signature of the conditional system, is sufficient (and indeed equivalent) to derive (operational) termination of the DCTRS (cf. [17, Theorems 4, 5 and Corollary 3]). In particular, this means that potential infinite reduction sequences issuing from (non-original) terms built over the extended signature of the transformed system may be ignored.

This generalizes previous results of [6] and [15]. In [6] general termination of $U_{cs}(\mathcal{R})$ was used as a sufficient condition for operational termination of $\mathcal{R}$ (however, there the transformation dealt with a wider class of conditional systems). In [15] it was shown that derivations (from original terms to original terms) in the transformed system correspond to derivations in the original conditional systems if the reduction in the transformed system follows a certain reduction strategy. We refer to [17] for a more thorough discussion of our above characterization result for operational termination of DCTRSs.

VMTL takes advantage of these facts by adapting its narrowing processors. Inside the dependency pair framework a narrowing processor basically exploits the fact that in a dependency pair chain the reduction sequence between two dependency pairs must be non-empty if the right-hand side of the first pair does not unify with the left-hand side of the second one. The (forward) narrowing processor then anticipates the possible first steps of this reduction sequence that affect the right-hand side of the first dependency pair $l \to r$ and replaces the latter by new pairs $\theta_1 l \to r_1, \ldots, \theta_n l \to r_n$ where $\{r_1, \ldots, r_n\}$ is the set of (context-sensitive) one-step narrowings of r and θ_i are the involved mgu's.

According to [17, Definitions 10–13, Theorems 7–10], only a subset of these narrowings need to be considered. Although heuristics are needed to approximate the relevant terms here, this helps to counter the explosion of the number of dependency pairs one often has to deal with when using narrowing processors.

6 Implementation Details and Benchmarks

VMTL (version 1.1) is entirely written in Java. The core module (i.e., without user interface) contains approximately 11.500 lines of code. MiniSat ([7]) is used as SAT solver.

Currently, there is no dedicated category for conditional TRSs in the termination problem database. Still, a few conditional TRSs are contained in it. Table 1 shows the benchmarks of VMTL and AProVE (Version 1.2)[2] on these examples, extended by further examples taken from [9,14,16]. Numbers in parentheses of the "Successful Proofs" column mean successful disproofs of termination. All experiments had a time limit of 60 seconds and were performed on an Intel E8500 (3.16 GHz) with 4GB of RAM under Ubuntu Hardy Heron (32 Bit) using java 6 Build 13.

Table 1. Benchmarks on conditional TRSs

Tool	**Successful Proofs**	**Number of Systems**
VMTL	19(3)	24
AProVE	14(2)	24

Table 2. Benchmarks on standard TRSs from the TPDB

Tool	**Successful Proofs**	**Number of Systems**
VMTL	629(106)	1391
AProVE	1226(231)	1391
TTT2	970(178)	1391
Jambox	810(60)	1391

Table 3. Benchmarks on context-sensitive TRSs from the TPDB

Tool	**Successful Proofs**	**Number of Systems**
VMTL	72(2)	109
AProVE	94(0)	109
MU-TERM	82(0)	109

For several examples used in the table, proving termination heavily depends on the methods described in [17, Section 6].

Tables 2 and 3 show the performance of VMTL on the set of standard TRSs and the set of context-sensitive ones, respectively, from the TPDB. More details regarding all benchmarks, including execution times and actual termination proofs can be found on the VMTL homepage[3].

[2] Newer versions of AProVE as well as other termination tools do not seem to support proper deterministic conditional rewrite systems (i.e., DCTRS with extra variables).

[3] `http://www.logic.at/vmtl/`

Note that apart from the restricted set of proof methods available in VMTL, the inferior performance for standard and context-sensitive TRSs is due to the strict modularity. Strategies in VMTL cannot be history-aware, which for instance prevents using methods like *safe narrowing* from [8]. In addition, the proof strategy cannot be adapted to certain classes of TRSs such as applicative ones.

7 Conclusion, Related and Future Work

We introduced the termination tool VMTL that provides an easily extensible implementation of the dependency pair framework. In particular, interfaces for dependency pair processors, preprocessing steps (e.g. transformations) and output preparation are available. VMTL is supposed to support researchers in the field of termination analysis, who do not have direct access to one of the existing termination tools. Using VMTL they may relatively easily try out and evaluate their ideas without having to build their own implementation from scratch. The system also comprises implementations of some standard termination analysis methods, that should make it even easier to obtain useful benchmarks. Moreover, VMTL provides implementations of state-of-the-art methods for the termination analysis of conditional and context-sensitive rewrite systems.

Regarding conditional rewrite systems, VMTL provides the infrastructure to follow the transformational approach of [17], that reduces the problem of proving (operational) termination of CTRSs to the problem of proving termination of a (transformed context-sensitive) TRS *on a restricted set of terms*. A simple method to exploit these results is realized in terms of two generalized narrowing processors, that enable VMTL to prove termination on original terms even if general termination does not hold.

Many of the currently developed termination tools follow the dependency pair framework. Although VMTL is not yet competitive for standard and context-sensitive TRSs, we see at least two points that distinguish VMTL from other existing tools:

- VMTL provides open and easily accessible interfaces for extensions.
- VMTL provides dedicated methods for the termination analysis of conditional rewrite systems that go beyond reduction to standard termination problems.

Future extensions of VMTL may provide means for an even more fine-grained guidance of proof attempts through a more powerful strategy language (adding for instance non-determinism). Moreover, an extension to new classes of rewrite systems and termination problems (e.g. higher-order rewrite systems and innermost/outermost or relative termination) and programming languages (e.g. Haskell) as input is desirable.

References

1. Alarcón, B., Emmes, F., Fuhs, C., Giesl, J., Gutiérrez, R., Lucas, S., Schneider-Kamp, P., Thiemann, R.: Improving context-sensitive dependency pairs. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS, vol. 5330, pp. 636–651. Springer, Heidelberg (2008)
2. Alarcón, B., Gutiérrez, R., Lucas, S.: Context-sensitive dependency pairs. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 297–308. Springer, Heidelberg (2006)
3. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoretical Computer Science 236(1–2), 133–178 (2000)
4. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press, New York (1998)
5. Bezem, M., Klop, J.W., de Vrijer, R. (eds.): Term rewriting systems. Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press, Cambridge (2003)
6. Durán, F., Lucas, S., Meseguer, J., Marché, C., Urbain, X.: Proving operational termination of membership equational programs. Higher-Order and Symbolic Computation 21, 59–88 (2008)
7. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
8. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. Journal of Automated Reasoning 37(3), 155–203 (2006)
9. Gmeiner, K., Gramlich, B.: Transformations of conditional rewrite systems revisited. In: Corradini, A., Montanari, U. (eds.) Recent Trends in Algebraic Development Techniques (WADT 2008) – Selected Papers. LNCS. Springer, Heidelberg (to appear, 2009)
10. Gutiérrez, R., Lucas, S., Urbain, X.: Usable rules for context-sensitive rewrite systems. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 126–141. Springer, Heidelberg (2008)
11. Koprowski, A.: Termination of rewriting and its certification. PhD thesis, Eindhoven University of Technology (2008)
12. Lucas, S.: Context-sensitive computations in functional and functional logic programs. Journal of Functional and Logic Programming 1998(1) (January 1998)
13. Lucas, S., Marché, C., Meseguer, J.: Operational termination of conditional term rewriting systems. Inf. Process. Lett. 95(4), 446–453 (2005)
14. Marchiori, M.: Unravelings and ultra-properties. In: Hanus, M., Rodríguez-Artalejo, M. (eds.) ALP 1996. LNCS, vol. 1139, pp. 107–121. Springer, Heidelberg (1996)
15. Nishida, N., Sakai, M., Sakabe, T.: Partial inversion of constructor term rewriting systems. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 264–278. Springer, Heidelberg (2005)
16. Ohlebusch, E.: Advanced topics in term rewriting. Springer, London (2002)
17. Schernhammer, F., Gramlich, B.: Characterizing and proving operational termination of deterministic conditional term rewriting systems. Technical Report E1852-2009-01, TU Wien (March 2009), `http://www.logic.at/vmtl/`
18. The termination competition, `http://termination-portal.org/wiki/Termination_Competition`
19. Thiemann, R., Giesl, J.: The size-change principle and dependency pairs for termination of term rewriting. Applicable Algebra in Engineering, Communication and Computing 16(4), 229–270 (2005)