Tsukuba Termination Tool*

Nao Hirokawa[1] and Aart Middeldorp**[2]

[1] Master's Program in Science and Engineering
University of Tsukuba, Tsukuba 305-8573, Japan
nao@score.is.tsukuba.ac.jp

[2] Institute of Information Sciences and Electronics
University of Tsukuba, Tsukuba 305-8573, Japan
ami@is.tsukuba.ac.jp

Abstract. We present a tool for automatically proving termination of first-order rewrite systems. The tool is based on the dependency pair method of Arts and Giesl. It incorporates several new ideas that make the method more efficient. The tool produces high-quality output and has a convenient web interface.

1 Introduction

Developing termination techniques for rewrite systems that can be automated has become an important research topic in the past few years. The dependency pair method of Arts and Giesl [3] is one of the most popular methods capable of automatically proving termination of first-order term rewrite systems (TRSs) that cannot be handled by traditional simplification orders. The dependency pair method has been implemented by Arts [1] and is part of the termination toolbox of C*i*ME [5]. Tsukuba Termination Tool (T$_\text{T}$T in the sequel) is a new tool in which the dependency pair method takes center-stage. In the following sections we explain the features of T$_\text{T}$T, give some implementation details, report on some of the experiments that we performed, and provide a brief comparison with the tools described in [1, 5]. We conclude with some ideas for future extensions of the tool. Familiarity with the dependency pair method will be helpful in the sequel.

2 Interface

We describe the features of T$_\text{T}$T by means of its web interface, displayed in Fig. 1.

TRS The user inputs a TRS by typing the rules into the upper left text area or by uploading a file via the browse button. The input syntax is obtained by clicking the TRS link.

* http://www.score.is.tsukuba.ac.jp/ttt/

** Partially supported by the Grant-in-Aid for Scientific Research (C)(2) 13224006 of the Ministry of Education, Culture, Sports, Science and Technology of Japan.

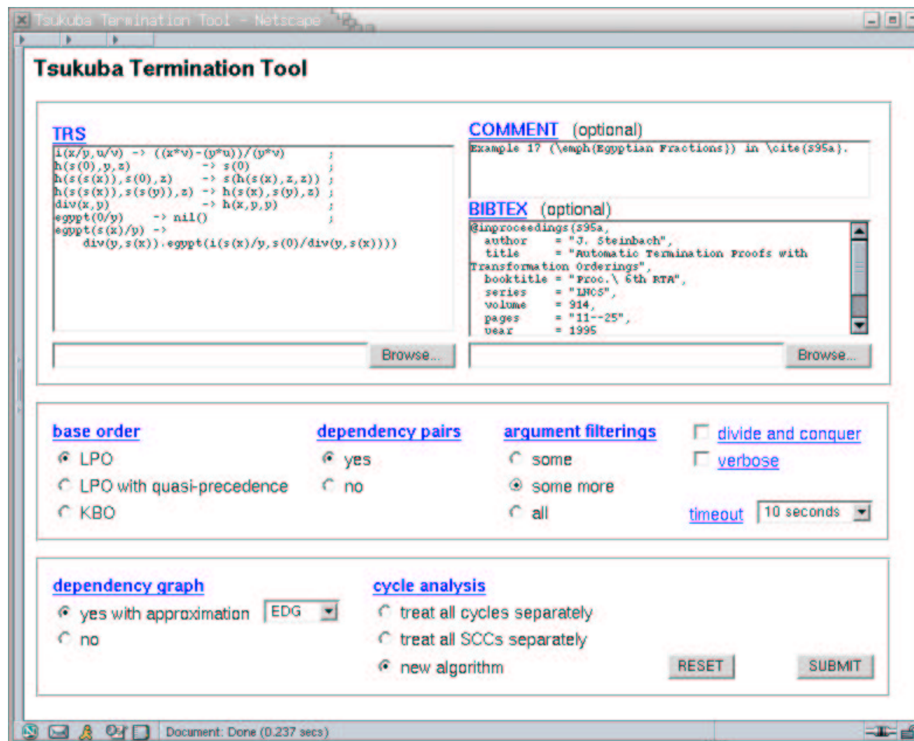


Fig. 1. A screen shot of the web interface of T$_{\rm T}$T.

Comment and Bibtex Anything typed into the upper right text area will appear as a footnote in the generated LaTeX code. This is useful to identify TRSs. LaTeX commands may be included. A typical example is a line like

```
Example 33 (\emph{Battle of Hydra and Hercules}) in \cite{D33}.
```

In order for this to work correctly, a bibtex entry for `D33` should be supplied. This can be done by typing the entry into the appropriate text area or by uploading an appropriate bibtex file via the browse button.

Base Order The current version of T$_{\rm T}$T supports the following three base orders: LPO (lexicographic path order) with strict precedence, LPO with quasi-precedence, and KBO (Knuth-Bendix order) with strict precedence. The implementation of KBO is based on the polynomial-time algorithm of Korovin and Voronkov [11]. In Section 4 we comment on the implementation of LPO precedence constraint solving.

Dependency Pairs Setting the dependency pair option activates the dependency pair technique of Arts and Giesl [2], which greatly enhances the termination proving power of the tool. The current version of T$_\text{T}$T supports the basic features of the dependency pair technique (argument filtering, dependency graph, cycle analysis) described below. Advanced features like narrowing, rewriting, and instantiation are not yet available. Also innermost termination analysis is not yet implemented.

Argument Filtering A single function symbol f of arity n gives rise to $2^n + n$ different argument filterings:

- $f(x_1, \dots, x_n) \to f(x_{i_1}, \dots, x_{i_m})$ for all $1 \leqslant i_1 < \cdots < i_m \leqslant n$,
- $f(x_1, \dots, x_n) \to x_i$ for all $1 \leqslant i \leqslant n$.

A moment's thought reveals that even for relatively small signatures, the number of possible argument filterings is huge. T$_\text{T}$T supports two simple heuristics to reduce this number.

- The *some* option considers for a function symbol f of arity n only the 'full' argument filtering $f(x_1, \dots, x_n) \to f(x_1, \dots, x_n)$ and the n 'collapsing' argument filterings $f(x_1, \dots, x_n) \to x_i$ $(1 \leqslant i \leqslant n)$.
- The *some more* option considers the argument filtering $f(x_1, \dots, x_n) \to f$ (when $n > 0$) in addition to the ones considered by the *some* option.

Dependency Graph The dependency graph determines the ordering constraints that have to be solved in order to guarantee termination. Since the dependency graph is in general not computable, a decidable approximation has to be adopted. The current version of T$_\text{T}$T supplies two such approximations:

- EDG is the original estimation of Arts and Giesl [2, latter part of Section 2.4].
- EDG$*$ is an improved version of EDG described in [12, latter half of Section 6].

We refer to [9] for some statistics related to these two approximations.

Cycle Analysis Once an approximation of the dependency graph has been computed, some kind of cycle analysis is required to generate the actual ordering constraints. T$_\text{T}$T offers three different methods:

1. The method described in [7] is to treat cycles in the approximated dependency graph separately. For every cycle $\mathcal{C}$, the dependency pairs in $\mathcal{C}$ and the rewrite rules of the given TRS must be weakly decreasing and at least one dependency pair in $\mathcal{C}$ must be strictly decreasing (with respect to some argument filtering and base order).
2. Another method, implemented in [1, 5], is to treat all strongly connected components (SCCs) separately. For every SCC $\mathcal{S}$, the dependency pairs in $\mathcal{S}$ must be strictly decreasing and the rewrite rules of the given TRS must be weakly decreasing. Treating SCCs rather than cycles separately improves the efficiency at the expense of reduced termination proving power.

3. The third method available in T$_\text{T}$T combines the termination proving power of the cycle method with the efficiency of the SCC method. It is described in [9].

Divide and Conquer The default option to find a suitable argument filtering that enables a group of ordering constraints to be solved by the selected base order is *enumeration*, which can be very inefficient, especially for larger TRSs where the number of suitable argument filterings is small. Setting the *divide and conquer* option computes all suitable argument filterings for each constraint separately and subsequently merges them to obtain the solutions of the full set of constraints. This can (greatly) reduce the execution time at the expense of an increased memory consumption. The divide and conquer option is described in detail in [9]. At the moment of writing it is only available in combination with LPO.

Verbose Setting the verbose option generates more proof details. In combination with the divide and conquer option described above, the total number of argument filterings that enable the successive ordering constraints to be solved are displayed during the termination proving process.

Timeout Every combination of options results in a finite search space for finding termination proofs. However, since it can take days to fully explore the search space, (the web version of) T$_\text{T}$T puts a strong upper bound on the permitted execution time.

3 Output

If T$_\text{T}$T succeeds in proving termination, it outputs a proof script which explains in considerable detail how termination was proved. This script is available in both HTML and LaTeX format. In the latter, the approximated dependency graph is visualized using the *dot* tool of the Graphviz toolkit [8]. Fig. 2 shows the generated output (with slightly readjusted vertical space to fit a single page) on Example 17 (*Egyptian Fractions*) in Steinbach [15] (in [15] the binary function symbol `i` is denoted by the infix operator $\bowtie$). Here T$_\text{T}$T is used with the EDG approximation of the dependency graph, the SCC approach to cycle analysis, *some more* argument filterings, and without the verbose option. As can be seen from Fig. 2, we prefer to output LPO precedences compactly like $\mathsf{div} \succ \mathsf{h} \succ \mathsf{s}$ as opposed to $\mathsf{div} \succ \mathsf{h}$; $\mathsf{h} \succ \mathsf{s}$ (or worse: $\mathsf{div} \succ \mathsf{h}$; $\mathsf{h} \succ \mathsf{s}$; $\mathsf{div} \succ \mathsf{s}$). This is achieved by an obvious topological sorting algorithm.

4 Implementation

T$_\text{T}$T is written in Objective Caml [13], which is a strongly typed functional programming language extended with object-oriented features. We make use

Termination Proof Script[a]

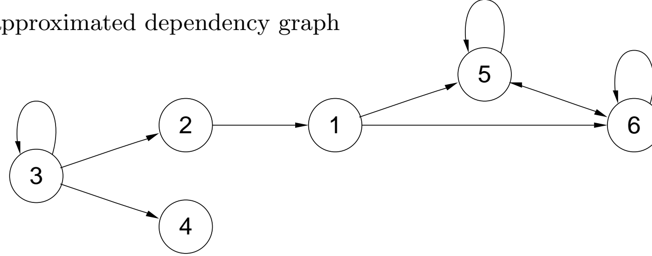
We prove that the TRS $\mathcal{R}$ consisting of the 7 rewrite rules[b]

$$\begin{aligned}
\mathsf{div}(x,y) &\rightarrow \mathsf{h}(x,y,y)\\
\mathsf{egypt}(\mathsf{0}\,/\,y) &\rightarrow \mathsf{nil}\\
\mathsf{egypt}(\mathsf{s}(x)\,/\,y) &\rightarrow \mathsf{div}(y,\mathsf{s}(x))\cdot\mathsf{egypt}(\mathsf{i}(\mathsf{s}(x)\,/\,y,\mathsf{s}(\mathsf{0})\,/\,\mathsf{div}(y,\mathsf{s}(x))))\\
\mathsf{h}(\mathsf{s}(\mathsf{0}),y,z) &\rightarrow \mathsf{s}(\mathsf{0})\\
\mathsf{h}(\mathsf{s}(\mathsf{s}(x)),\mathsf{s}(\mathsf{0}),z) &\rightarrow \mathsf{s}(\mathsf{h}(\mathsf{s}(x),z,z))\\
\mathsf{h}(\mathsf{s}(\mathsf{s}(x)),\mathsf{s}(\mathsf{s}(y)),z) &\rightarrow \mathsf{h}(\mathsf{s}(x),\mathsf{s}(y),z)\\
\mathsf{i}(x\,/\,y,u\,/\,v) &\rightarrow ((x\times v)-(y\times u))\,/\,(y\times v)
\end{aligned}$$

is terminating. There are 6 dependency pairs:

$$\begin{aligned}
1:\quad \mathsf{DIV}(x,y) &\rightarrow \mathsf{H}(x,y,y)\\
2:\quad \mathsf{EGYPT}(\mathsf{s}(x)\,/\,y) &\rightarrow \mathsf{DIV}(y,\mathsf{s}(x))\\
3:\quad \mathsf{EGYPT}(\mathsf{s}(x)\,/\,y) &\rightarrow \mathsf{EGYPT}(\mathsf{i}(\mathsf{s}(x)\,/\,y,\mathsf{s}(\mathsf{0})\,/\,\mathsf{div}(y,\mathsf{s}(x))))\\
4:\quad \mathsf{EGYPT}(\mathsf{s}(x)\,/\,y) &\rightarrow \mathsf{I}(\mathsf{s}(x)\,/\,y,\mathsf{s}(\mathsf{0})\,/\,\mathsf{div}(y,\mathsf{s}(x)))\\
5:\quad \mathsf{H}(\mathsf{s}(\mathsf{s}(x)),\mathsf{s}(\mathsf{0}),z) &\rightarrow \mathsf{H}(\mathsf{s}(x),z,z)\\
6:\quad \mathsf{H}(\mathsf{s}(\mathsf{s}(x)),\mathsf{s}(\mathsf{s}(y)),z) &\rightarrow \mathsf{H}(\mathsf{s}(x),\mathsf{s}(y),z)
\end{aligned}$$

The EDG approximated dependency graph



contains 2 SCCs: $\{5,6\}$, $\{3\}$.

- Consider the SCC $\{5,6\}$. By taking the AF $\cdot \mapsto []$ and LPO with precedence $\mathsf{div} \succ \mathsf{h} \succ \mathsf{s}$; $\mathsf{egypt} \succ \cdot$; $\mathsf{i} \succ \times$; $\mathsf{i} \succ -$; $\mathsf{i} \succ / \succ \mathsf{nil}$, the rules in $\mathcal{R}$ are weakly decreasing and the rules in $\{5,6\}$ are strictly decreasing.
- Consider the SCC $\{3\}$. By taking the AF $\times, \mathsf{i} \mapsto []$ and LPO with precedence $\mathsf{egypt} \succ \cdot$; $\mathsf{egypt} \succ \mathsf{div} \succ \mathsf{h} \succ \mathsf{s} \succ \mathsf{i} \succ \times$; $\mathsf{egypt} \succ \mathsf{nil}$; $\mathsf{i} \succ -$; $\mathsf{i} \succ /$, the rules in $\mathcal{R}$ are weakly decreasing and the rules in $\{3\}$ are strictly decreasing.

References

1. J. Steinbach. Automatic termination proofs with transformation orderings. In *Proc. 6th RTA*, volume 914 of *LNCS*, pages 11–25, 1995.

[a] `http://www.score.is.tsukuba.ac.jp/ttt`
[b] Example 17 (*Egyptian Fractions*) in [1].

Fig. 2. Output.

of the latter for the enumeration of argument filterings, but most of the code is written in a purely functional style. For instance, to compute SCCs and cycles in the approximated dependency graph the depth-first search algorithm described in [10] is used. Below we comment on the implementation of LPO precedence constraint solving.

The definition of LPO with strict precedence can be rendered as follows:

$$
\begin{aligned}
s >_{\text{lpo}} t &= s >^1_{\text{lpo}} t \lor s >^2_{\text{lpo}} t \\
x >^1_{\text{lpo}} t &= x >^2_{\text{lpo}} t = \bot \\
f(s_1,\ldots,s_n) >^1_{\text{lpo}} t &= \exists i\ (s_i = t \lor s_i >_{\text{lpo}} t) \\
f(s_1,\ldots,s_n) >^2_{\text{lpo}} f(t_1,\ldots,t_n) &= \\
&\quad (\forall i\ f(s_1,\ldots,s_n) >_{\text{lpo}} t_i) \land (s_1,\ldots,s_n) >^{\text{lex}}_{\text{lpo}} (t_1,\ldots,t_n) \\
f(s_1,\ldots,s_n) >^2_{\text{lpo}} g(t_1,\ldots,t_m) &= (\forall i\ f(s_1,\ldots,s_n) >_{\text{lpo}} t_i) \land (f > g)
\end{aligned}
$$

with $() \succ^{\text{lex}} () = \bot$ and $(s_1,\ldots,s_n) \succ^{\text{lex}} (t_1,\ldots,t_n) = s_1 \succ t_1 \lor (s_1 = t_1 \land (s_2,\ldots,s_n) \succ^{\text{lex}} (t_2,\ldots,t_n))$ for $n > 0$. Finding a precedence $>$ such that $s >_{\text{lpo}} t$ for concrete terms s and t is tantamount to solving the constraint that is obtained by unfolding the definition of $s >_{\text{lpo}} t$. The constraint involves the boolean connectives $\land$ and $\lor$, $\bot$, and atomic statements of the forms $f > g$ for function symbols f, g and $s = t$ for terms s, t. These symbols are interpreted as sets of precedences, as follows:

$$
\begin{aligned}
[\![C \lor D]\!] &= \text{mins}\,([\![C]\!] \cup [\![D]\!]) & [\![\bot]\!] &= \varnothing \\
[\![C \land D]\!] &= \text{mins}\,([\![C]\!] \otimes [\![D]\!]) & [\![f > g]\!] &= \{(f,g)\}
\end{aligned}
$$

$$
[\![s = t]\!] = \begin{cases} \{\varnothing\} & \text{if } s \text{ and } t \text{ are the same term} \\ \varnothing & \text{otherwise} \end{cases}
$$

Here $O_1 \otimes O_2$ denotes the set of all strict orders $(>_1 \cup >_2)^+$ with $>_1 \in O_1$ and $>_2 \in O_2$. The purpose of the operator mins, which removes non-minimal precedences from its argument, is to avoid the computation of redundant precedences. For instance, one readily verifies that $[\![\mathsf{f}(\mathsf{c}) >_{\text{lpo}} \mathsf{c}]\!] = \{\varnothing\}$ whereas without mins we would get $[\![\mathsf{f}(\mathsf{c}) >_{\text{lpo}} \mathsf{c}]\!] = \{\varnothing, \{(\mathsf{f},\mathsf{c})\}\}$.

Now, by encoding the above definitions one almost immediately obtains an implementation of LPO precedence constraint solving. For instance, TT contains (a slightly optimized version of) the following OCaml code fragment:

```
let bottom   = empty
let disj c d = minimal (union c d)
let conj c d = minimal (combine c d)

let rec lex rel ss ts =
  match ss, ts with
  | s :: ss', t :: ts' when s = t -> lex rel ss' ts'
  | s :: ss', t :: ts'            -> rel s t
```

```
let rec lpo s t = disj (lpo1 s t) (lpo2 s t)
and lpo1 s t =
  match s with
  | V _       -> bottom
  | F (f, ss) -> exists (fun s' -> disj (equal s' t) (lpo s' t)) ss
and lpo2 s t =
  match s, t with
  | F (f, ss), F (g, ts) when f = g
      -> conj ((for_all (fun t' -> lpo s t')) ts) (lex lpo ss ts)
  | F (f, ss), F (g, ts)
      -> conj ((for_all (fun t' -> lpo s t')) ts) (prec f g)
  | _ -> bottom
```

The point we want to emphasize is that other precedence-based syntactic orderings follow the same scenario and can thus be added very easily to T$_\mathrm{T}$T.

5 Experimental Results

We tested the various options of T$_\mathrm{T}$T on numerous examples. Here we consider 14 examples from the literature. Our findings are summarized in Tables 1 and 2. All experiments were performed on a notebook PC with an 800 MHz Pentium III CPU and 128 MB memory. The numbers in the two tables denote execution time in seconds. *Italics* indicate that T$_\mathrm{T}$T could not prove termination while fully exploring the search space implied by the options within the given time. Question marks denote a timeout of one hour. For the experiments in Table 1 we used LPO with strict precedence as base order, EDG as dependency graph approximation, and enumeration of argument filterings.

The three question marks for [6, Example 11] are explained by the fact that the dependency graph admits 4095 cycles (but only 1 strongly connected component, consisting of all 12 dependency pairs). The largest example in the collection, Example 3.11 (*Quicksort*) in [3], clearly reveals the benefits of the argument filtering heuristics as well as the new approach to cycle analysis.

From columns (1), (2), and (3) in Table 2 we infer that the divide and conquer option has an even bigger impact on this example, especially if one keeps in mind that all suitable argument filterings are computed. Here we used the new algorithm for cycle analysis, LPO with strict precedence as base order, EDG as dependency graph approximation, and *some* (1), *some more* (2), and *all* (3) argument filterings. The question marks in column (3) are largely explained by the large memory demands of the divide and conquer option. We will address this issue in the near future. Note that [6, Example 11] is the only example in the collection which can be directly handled by LPO.

In columns (4) and (5) we used KBO as base order and *some* respectively *some more* argument filterings (as well as the new algorithm for cycle analysis and EDG as dependency graph approximation). According to column (5) T$_\mathrm{T}$T

Table 1. Argument filtering heuristics in combination with cycle analysis methods.

argument filterings	some			some more			all		
cycle analysis	cycle	scc	new	cycle	scc	new	cycle	scc	new
[3, Example 3.3]	0.02	*0.08*	0.02	0.14	*0.81*	0.13	5.72	5.68	5.78
[3, Example 3.4]	*0.01*	*0.01*	*0.01*	0.02	*0.02*	0.02	0.09	0.06	0.08
[3, Example 3.9]	0.28	*0.23*	0.13	1.94	*2.32*	0.86	9.79	4.38	4.48
[3, Example 3.11]	14.29	*9.35*	5.79	152.45	*209.10*	57.51	?	?	?
[3, Example 3.15]	*0.00*	*0.00*	*0.00*	*0.00*	*0.00*	*0.00*	*0.01*	*0.01*	*0.01*
[3, Example 3.38]	*0.01*	*0.02*	*0.01*	*0.05*	*0.05*	*0.05*	4.38	0.87	0.91
[3, Example 3.44]	*0.00*	*0.01*	*0.01*	0.01	0.00	0.01	0.00	0.00	0.01
[4, Example 6]	*0.06*	*0.05*	*0.05*	0.89	0.60	0.26	0.82	*2.97*	0.24
[6, Example 11]	?	0.08	0.08	?	0.08	0.08	?	16.46	15.21
[6, Example 33]	*0.05*	*0.05*	*0.05*	*0.13*	*0.13*	*0.12*	*0.48*	*0.44*	*0.48*
[15, Example 17]	*1.77*	*1.75*	*1.63*	4.41	4.39	4.19	1904.79	805.92	1045.75
[16, Example 4.27]	*0.00*	*0.01*	*0.01*	*0.01*	*0.01*	*0.01*	*0.03*	*0.04*	*0.03*
[16, Example 4.60]	*0.17*	*0.17*	*0.16*	0.23	0.13	0.14	41.19	21.50	21.42
[17, Example 58]	*0.00*	*0.00*	*0.00*	*0.00*	*0.00*	*0.00*	0.01	0.01	0.01

requires 593.26 seconds to prove the termination of Example 33 (*Battle of Hydra and Hercules*) in [6]. This example nicely illustrates that the termination proving power of KBO, which is considered not to be very large, is increased significantly in combination with dependency pairs. The ARTS and C*i*ME columns are described in the next section.

6 Comparison

TTT is not the first tool that implements the dependency pair method. The implementation of Arts [1] offers more refinements (like narrowing and termination via innermost termination) of the dependency pair method and, via its graphical user interface, allows the user to choose a particular argument filtering (separately for each group of ordering constraints). In contrast to the latter, TTT offers improved algorithms for the automatic search for suitable argument filterings. For the ARTS column in Table 2 we used the only available automatic strategy in the distribution, which is (partly) described in [1, Section 3], and not guaranteed to terminate. Most of the successful termination proofs it generates use the refinements mentioned above.

The implementation of the dependency pair method in C*i*ME [5] uses weakly monotone polynomial interpretations as base order (which removes the need for argument filterings) and the SCC method for cycle analysis in the EDG approximated dependency graph. The search for a suitable polynomial interpretation can be restricted by specifying a certain class of simple polynomials as well as indicating a restricted range for the coefficients. Needless to say, the use of poly-

Table 2. Divide and conquer, KBO, and other tools.

	(1)	(2)	(3)	(4)	(5)	LPO	KBO	ARTS	C*i*ME
[3, Example 3.3]	0.10	3.30	492.76	269.03	?	*0.00*	*0.00*	58.92	?
[3, Example 3.4]	*0.01*	0.02	0.08	*3.13*	25.51	*0.00*	*0.00*	2.76	0.19
[3, Example 3.9]	0.25	3.43	340.99	1183.73	?	*0.00*	*0.00*	391.18	?
[3, Example 3.11]	0.47	4.42	430.72	*2.25*	*33.42*	*0.00*	*0.01*	?	?
[3, Example 3.15]	*0.01*	*0.00*	*0.01*	0.24	0.24	*0.00*	0.03	?	0.08
[3, Example 3.38]	*0.01*	*0.05*	4.50	*0.02*	?	*0.00*	*0.00*	613.32	0.19
[3, Example 3.44]	*0.00*	0.04	0.04	*6.55*	10.16	*0.00*	*0.87*	0.60	0.02
[4, Example 6]	*0.02*	3.08	80.65	*0.09*	1783.14	*0.00*	*0.00*	?	*440.39*
[6, Example 11]	2.08	92.92	?	*0.39*	*5.52*	0.00	*0.00*	4.43	*61.36*
[6, Example 33]	*0.02*	*0.12*	*0.89*	*0.18*	593.26	*0.00*	*0.00*	?	*0.78*
[15, Example 17]	*0.66*	57.85	?	67.36	?	*0.00*	*0.00*	?	15.80
[16, Example 4.27]	*0.01*	*0.05*	*0.29*	202.65	396.36	*0.00*	*5.17*	?	*1485.27*
[16, Example 4.60]	*0.04*	0.36	23.14	*0.28*	387.01	*0.00*	*0.00*	6.44	1.74
[17, Example 58]	*0.00*	*0.00*	0.02	0.43	0.45	*0.00*	0.06	1.35	0.06

nomial interpretations considerably restricts the class of terminating TRSs that can be proved terminating (automatically or otherwise). On the other hand, C*i*ME admits AC operators and supports powerful modularity criteria based on the dependency pair method (described in [18, 19]), two extensions which are not (yet) available in TTT. The data in the C*i*ME column was obtained by using the default options to restrict the search for polynomial interpretations: simple-mixed polynomials with coefficients in the range 0–6.

7 Future Extensions

In the near future we plan to add an option that makes TTT search for a termination proof using everything in its arsenal. The challenge here is to develop a strategy that finds proofs quickly without compromising the ability to find proofs at all.

Other future extensions include adding more base orders, incorporating the refinements of the dependency pair method mentioned in the first paragraph of Section 6, and implementing the powerful modularity criteria based on the dependency pair method described in [14] and [19]. We also plan to add techniques for AC-termination. In Section 5 we already mentioned the reduction of the large memory requirements of the divide and conquer option as an important topic for future research.

Another interesting idea is to generate output that allows for an independent check of the termination proof. We plan to develop some kind of formal language in which all kinds of termination proofs can be conveniently expressed.

This development may lead to cooperating rather than competing termination provers.

Acknowledgments

We thank the anonymous referees for several suggestions to improve the presentation.

References

1. T. Arts. System description: The dependency pair method. In *Proc. 11th RTA*, volume 1833 of *LNCS*, pages 261–264, 2001.
2. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
3. T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-2001-09, RWTH Aachen, 2001. Available at `http://aib.informatik.rwth-aachen.de/`.
4. C. Borralleras, M. Ferreira, and A. Rubio. Complete monotonic semantic path orderings. In *Proc. 17th CADE*, volume 1831 of *LNAI*, pages 346–364, 2000.
5. E. Contejean, C. Marché, B. Monate, and X. Urbain. C*i*ME version 2, 2000. Available at `http://cime.lri.fr/`.
6. N. Dershowitz. 33 Examples of termination. In *French Spring School of Theoretical Computer Science*, volume 909 of *LNCS*, pages 16–26, 1995.
7. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.
8. `http://www.research.att.com/sw/tools/graphviz/`.
9. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. In *Proc. 19th CADE*, LNAI, 2003. To appear.
10. D.J. King and J. Launchbury. Structuring depth-first search algorithms in Haskell. In *Proc. 22nd POPL*, pages 344–354, 1995.
11. K. Korovin and A. Voronkov. Verifying orientability of rewrite rules using the Knuth-Bendix order. In *Proc. 12th RTA*, volume 2051 of *LNCS*, pages 137–153, 2001.
12. A. Middeldorp. Approximations for strategies and termination. In *Proc. 2nd WRS*, volume 70(6) of *Electronic Notes in Theoretical Computer Science*, 2002.
13. `http://caml.inria.fr/ocaml/`.
14. E. Ohlebusch. Hierarchical termination revisited. *Information Processing Letters*, 84(4):207–214, 2002.
15. J. Steinbach. Automatic termination proofs with transformation orderings. In *Proc. 6th RTA*, volume 914 of *LNCS*, pages 11–25, 1995.
16. J. Steinbach and U. Kühler. Check your ordering – termination proofs and open problems. Technical Report SR-90-25, Universität Kaiserslautern, 1990.
17. Joachim Steinbach. Simplification orderings: History of results. *Fundamenta Informaticae*, 24:47–87, 1995.
18. X. Urbain. Automated incremental termination proofs for hierarchically defined term rewriting systems. In *Proc. IJCAR*, volume 2083 of *LNAI*, pages 485–498, 2001.
19. X. Urbain. Modular & incremental automated termination proofs. *Journal of Automated Reasoning*, 2003. To appear.