Tyrolean Termination Tool[⋆]

Nao Hirokawa and Aart Middeldorp

Institute of Computer Science
University of Innsbruck
6020 Innsbruck, Austria
{nao.hirokawa,aart.middeldorp}@uibk.ac.at

1 Introduction

This paper describes the Tyrolean Termination Tool (T┬T in the sequel), the successor of the Tsukuba Termination Tool [12]. We describe the differences between the two and explain the new features, some of which are not (yet) available in any other termination tool, in some detail. T┬T is a tool for automatically proving termination of rewrite systems based on the dependency pair method of Arts and Giesl [3]. It produces high-quality output and has a convenient web interface. The tool is available at

`http://cl2-informatik.uibk.ac.at/ttt`

T┬T incorporates several new improvements to the dependency pair method. In addition, it is now possible to run the tool in *fully automatic mode* on a *collection* of rewrite systems. Moreover, besides ordinary (first-order) rewrite systems, the tool accepts simply-typed applicative rewrite systems which are transformed into ordinary rewrite systems by the recent method of Aoto and Yamada [2].

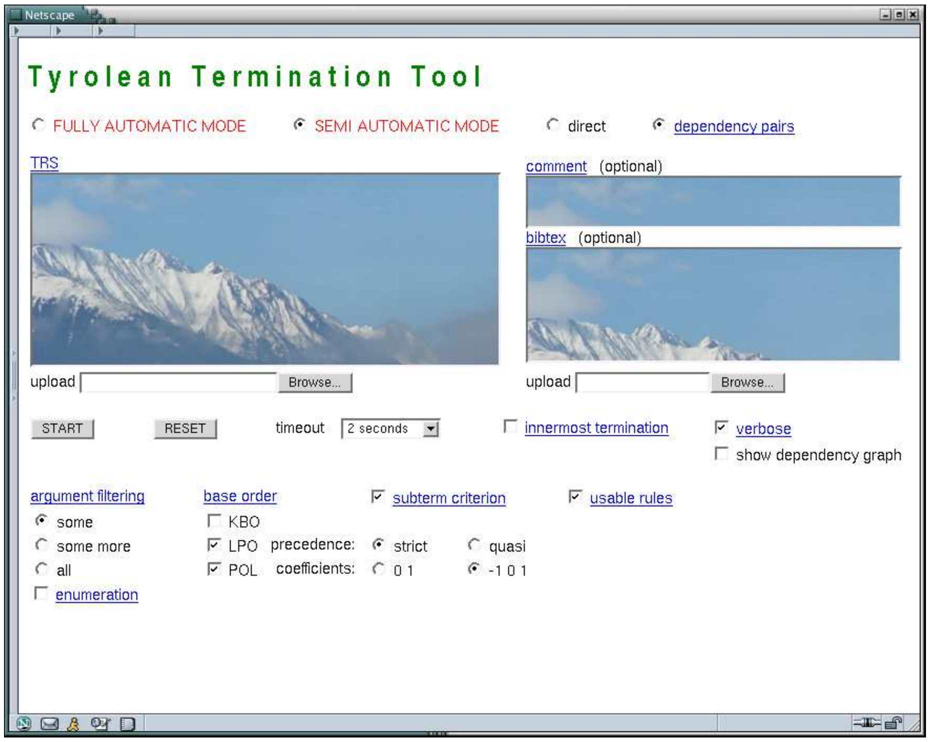
In the next section we describe the differences between the semi automatic mode and the Tsukuba Termination Tool. Section 3 describes the fully automatic mode. In Section 4 we show a termination proof of a simply-typed applicative system obtained by T┬T. In Section 5 we describe how to input a collection of rewrite systems and how to interpret the resulting output. Some implementation details are given in Section 6. The final section contains a short comparison with other tools for automatically proving termination.

2 Semi Automatic Mode

Figure 1 shows the web interface.

This menu corresponds to the options that were available in the Tsukuba Termination Tool. A first difference is that we now support the dependency pair method for innermost termination [3]. A second difference is that dependency

[⋆] A preliminary description of the Tyrolean Termination Tool appeared in the proceedings of the 7th International Workshop on Termination, Technical Report AIB-2004-07, RWTH Aachen, pages 249–268, 2004.



FULLY AUTOMATIC MODE [✓ SEMI AUTOMATIC MODE]
direct [✓ dependency pairs]

timeout 2 seconds
innermost termination
[✓ verbose]
show dependency graph

argument filtering: [✓ some] some more all
enumeration

base order: KBO [✓ LPO] [✓ POL]
precedence: [✓ strict] quasi
coefficients: 0 1 [✓ -1 0 1]

[✓ subterm criterion]
[✓ usable rules]

Fig. 1. A screen shot of the semi automatic mode of T$_\mathsf{T}$T.

pairs that are covered by the subterm criterion of Dershowitz [7] are excluded. The other differences are described in the following paragraphs.

First of all, when approximating the (innermost) dependency graph the original estimations of [3] are no longer available since the approximations described in [15] generally produce smaller graphs while the computational overhead is negligible.

Secondly, the user can no longer select the cycle analysis method (all cycles separately, all strongly connected components separately, or the recursive SCC algorithm of [15]). Extensive experiments reveal that the latter method outperforms the other two, so this is now the only supported method in T$_\mathsf{T}$T.

Finally, the default method to search for appropriate argument filterings has been changed from *enumeration* to the *divide and conquer* algorithm of [15]. By using dynamic programming techniques, the divide and conquer method has been improved (cf. [15]) to the extent that for most examples it is more efficient than the straightforward enumeration method. Still, there are TRSs where enumeration is more effective, so the user has the option to change the search strategy (by clicking the enumerate box).

New features include (1) a very useful criterion based on the subterm relation to discard SCCs of the dependency graph without considering any rewrite rules and (2) a very powerful modularity criterion for termination inspired by the *usable rules* of [3] for innermost termination. These features are described in detail in [13]. The first one is selected by clicking the *subterm criterion* box and the second by clicking the *usable rules* box. In addition, linear polynomial interpretations with coefficients from $\{-1, 0, 1\}$ can be used as base order. In [14] it is explained how polynomial interpretations with negative coefficients, like $x - 1$ for a unary function symbol or $x - y$ for a binary function symbol, can be effectively used in connection with the dependency pair method.

3 Fully Automatic Mode

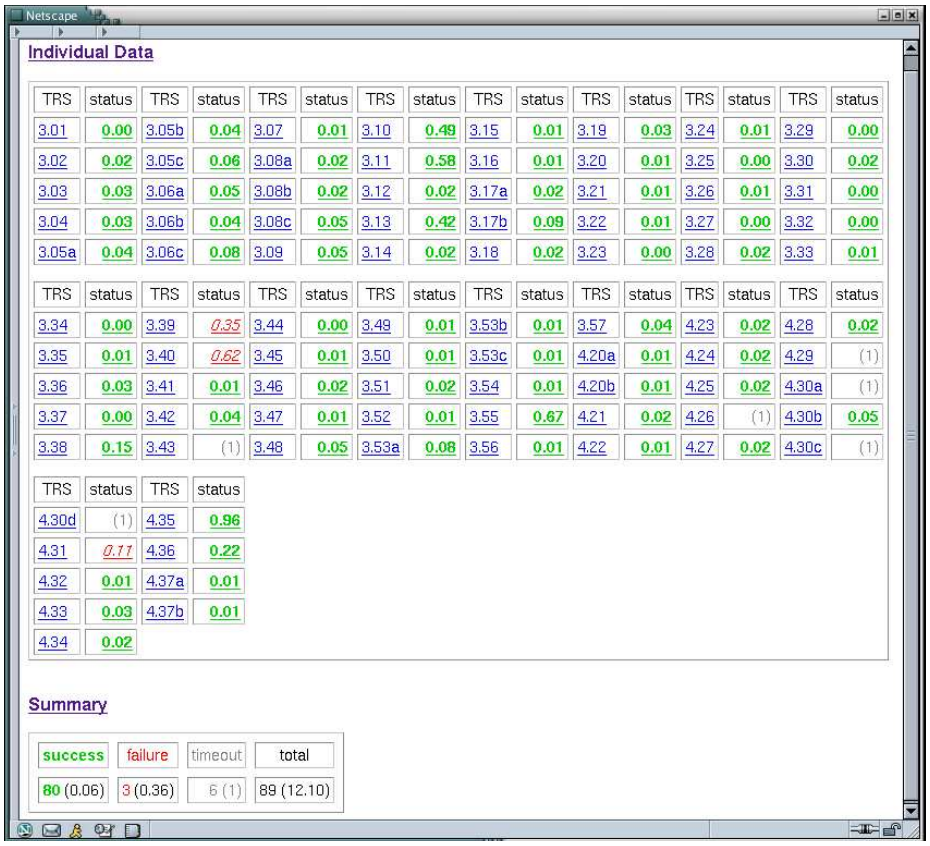
In this mode T$\mathsf{_T}$T uses a simple strategy to (recursively) solve the ordering constraints for each SCC of the approximated dependency graph. The strategy is based on the new features described in the previous section and uses LPO (both with strict and quasi-precedence) with *some* argument filterings [15] and linear polynomial interpretations with coefficients from $\{-1, 0, 1\}$ as base orders.

After computing the SCCs of the approximated (innermost) dependency graph, the strategy subjects each SCC to the following algorithm:

1. First we check whether the new *subterm criterion* is applicable.
2. If the subterm criterion was unsuccessful, we compute the *usable rules*.
3. The resulting (usable rules and dependency pairs) constraints are subjected to the *natural* (see [14]) polynomial interpretation with coefficients from $\{0, 1\}$.
4. If the constraints could not be solved in step 3, we employ the *divide and conquer* algorithm for computing suitable argument filterings with respect to the *some* heuristic [15] and LPO with *strict* precedence.
5. If the previous step was unsuccessful, we repeat step 3 with *arbitrary* polynomial interpretations with coefficients from $\{0, 1\}$.
6. Next we repeat step 4 with the variant of LPO based on *quasi-precedences* and a small increase in the search space for argument filterings (see below).
7. If the constraints could still not be solved, we try linear polynomial interpretations with coefficients from $\{-1, 0, 1\}$.

If only part of an SCC could be handled, we subject the resulting new SCCs recursively to the same algorithm.

If the current set of constraints can be solved in step 3 or 4, then they can also be solved in step 5 or 6, respectively, but the reverse is not true. The sole reason for adopting LPO and polynomial interpretations in alternating layers is efficiency; the search space in steps 3 and 4 is significantly smaller than in steps 5 and 6. The reason for putting the subterm criterion first is that with this criterion many SCCs can be eliminated very quickly, cf. the third paragraph of Section 6. The extension of the search space for argument filterings mentioned in step 6 is obtained by also considering the full *reverse* argument filtering $[n, \ldots, 1]$



Individual Data

TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status
3.01	0.00	3.05b	0.04	3.07	0.01	3.10	0.49	3.15	0.01	3.19	0.03	3.24	0.01	3.29	0.00
3.02	0.02	3.05c	0.06	3.08a	0.02	3.11	0.58	3.16	0.01	3.20	0.01	3.25	0.00	3.30	0.02
3.03	0.03	3.06a	0.05	3.08b	0.02	3.12	0.02	3.17a	0.02	3.21	0.01	3.26	0.01	3.31	0.00
3.04	0.03	3.06b	0.04	3.08c	0.05	3.13	0.42	3.17b	0.09	3.22	0.01	3.27	0.00	3.32	0.00
3.05a	0.04	3.06c	0.08	3.09	0.05	3.14	0.02	3.18	0.02	3.23	0.00	3.28	0.02	3.33	0.01

TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status
3.34	0.00	3.39	*0.35*	3.44	0.00	3.49	0.01	3.53b	0.01	3.57	0.04	4.23	0.02	4.28	0.02
3.35	0.01	3.40	*0.62*	3.45	0.01	3.50	0.01	3.53c	0.01	4.20a	0.01	4.24	0.02	4.29	(1)
3.36	0.03	3.41	0.01	3.46	0.02	3.51	0.02	3.54	0.01	4.20b	0.01	4.25	0.02	4.30a	(1)
3.37	0.00	3.42	0.04	3.47	0.01	3.52	0.01	3.55	0.67	4.21	0.02	4.26	(1)	4.30b	0.05
3.38	0.15	3.43	(1)	3.48	0.05	3.53a	0.08	3.56	0.01	4.22	0.01	4.27	0.02	4.30c	(1)

TRS	status	TRS	status
4.30d	(1)	4.35	0.96
4.31	*0.11*	4.36	0.22
4.32	0.01	4.37a	0.01
4.33	0.03	4.37b	0.01
4.34	0.02		

Summary

success	failure	timeout	total
80 (0.06)	3 (0.36)	6 (1)	89 (12.10)

Fig. 2. Output produced by TT.

for every n-ary function symbol. The advantage of this extension is that there is no need for a specialized version of LPO with right-to-left status.

The effectiveness of the automatic strategy can be seen from the data presented in Figure 2, which were obtained by running TT in fully automatic mode on the 89 terminating TRSs (66 in Section 3 and 23 in Section 4) of [4]. An explanation of the data is given in Section 5.

Our automatic strategy differs from the "Meta-Combination Algorithm" described in [11]; we avoid transforming SCC constraints using techniques like narrowing and instantiation because they tend to complicate the produced termination proofs. Instead, we rely on techniques (subterm criterion and polynomial interpretations with negative coefficients) that lead to termination proofs that are (relatively) easy to understand.

4 Simply-Typed Applicative Rewrite Systems

Besides ordinary first-order TRSs, TT accepts *simply-typed applicative rewrite systems* (STARSs) [1]. Applicative terms are built from variables, constants, and a single binary operator $\cdot$, called application. Constants and variables are equipped with a simple type such that the rewrite rules typecheck. A typical example is provided by the following rules for the map function

$$\begin{aligned}(\mathsf{map} \cdot f) \cdot \mathsf{nil} &\to \mathsf{nil} \\ (\mathsf{map} \cdot f) \cdot ((\mathsf{cons} \cdot x) \cdot y) &\to (\mathsf{cons} \cdot (f \cdot x)) \cdot ((\mathsf{map} \cdot f) \cdot y)\end{aligned}$$

with the type declaration $\mathsf{nil}\colon \alpha$, $\mathsf{cons}\colon \beta \to \alpha \to \alpha$, $\mathsf{map}\colon (\beta \to \beta) \to \alpha \to \alpha$, $f\colon \beta \to \beta$, $x\colon \beta$, and $y\colon \alpha$. Here α is the list type and β the type of elements of lists. STARSs are useful to model higher-order functions in a first-order setting. As usual, the application operator $\cdot$ is suppressed in the notation and parentheses are removed under the "association to the left" rule. The above rules then become

$$\begin{aligned}\mathsf{map}\ f\ \mathsf{nil} &\to \mathsf{nil} \\ \mathsf{map}\ f\ (\mathsf{cons}\ x\ y) &\to \mathsf{cons}\ (f\ x)\ (\mathsf{map}\ f\ y)\end{aligned}$$

This corresponds to the syntax of STARSs in TTT. The types of constants must be declared by the keyword `TYPES`. The types of variables is automatically inferred when typechecking the rules, which follow the `RULES` keyword. So the above STARS would be inputted to TTT as

```
TYPES
 nil : a                                ;
cons : b => (a => a)                    ;
 map : (b => b) => a => a               ;

RULES
map f nil        -> nil                 ;
map f (cons x y) -> cons (f x) (map f y) ;
```

In order to prove termination of STARSs, TTT uses the two-phase transformation developed by Aoto and Yamada [2]. In the first phase all head variables (e.g. `f` in `f x`) are removed by the *head variable instantiation* technique. The soundness of this phase relies on the *ground term existence condition*, which basically states that all simple types are inhabited by at least one ground term. Users need not be concerned about this technicality as TTT automatically adds fresh constants of the appropriate types to the signature so that the ground term existence condition is satisfied. (Moreover, the termination status of the original STARS is not affected by adding fresh constants.) After the first phase an ordinary TRS is obtained in which the application symbol is the only non-constant symbol. Such TRSs are not easily proved terminating since the root symbol of every term that has at least two symbols is the application symbol and thus provides no information which could be put to good use. In the second phase applicative terms are transformed into ordinary terms by the *translation to functional form* technique. This technique removes all occurrences of the application symbol. We refer to [2] for a complete description of the transformation. We contend ourselves with showing the Postscript output (in Figure 3) produced by T_TT on the following variation of combinatory logic (inspired by a recent question posted on the TYPES Forum by Jeremy Dawson):

```
TYPES
I : o => o                                ;
W : (o => o => o) => o => o               ;
S : (o => o => o) => (o => o) => o => o ;

RULES
I x      -> x         ;
W f x    -> f x x     ;
S x y z -> x z (y z) ;
```

Note that the types are crucial for termination; the untyped version admits the cyclic rewrite step W W W $\to$ W W W.

5 A Collection of Rewrite Systems as Input

Single TRSs (or STARSs) are inputted by typing (the type declarations and) the rules into the upper left text area or by uploading a file via the browse button. Besides the original TTT syntax (which is obtained by clicking the TRS link), TTT supports the official format[1] of the Termination Problems Data Base. The user can also upload a zip archive. All files ending in `.trs` are extracted from the archive and the termination prover runs on each of these files in turn. The results are collected and presented in two tables. The first table lists for each TRS the execution time in seconds together with the status: **bold green** indicates success, *red italics* indicates failure, and gray indicates timeout. By clicking **green** (*red*) entries the user can view the termination proof (attempt) in HTML or high-quality Postscript format. The second table gives the number of successes and failures, both with the average time spent on each TRS, the number of timeouts, and the total number of TRSs extracted from the zip archive together with the total execution time. Figure 2 shows the two tables for the 89 terminating TRSs in Sections 3 and 4 of [4]. Here we used TTT's fully automatic mode with a timeout of 1 second (for each TRS). The experiment was performed on a PC equipped with a 2.20 GHz Mobile Intel Pentium 4 Processor - M and 512 MB of memory, using native-compiled code for Linux/Fedora.

[1] `http://www.lri.fr/~marche/wst2004-competition/format.html`

Termination Proof Script[a]

Consider the simply-typed applicative TRS

$$\begin{aligned} \mathsf{I}\ x &\to x \\ \mathsf{W}\ f\ x &\to f\ x\ x \\ \mathsf{S}\ x\ y\ z &\to x\ z\ (y\ z) \end{aligned}$$

over the signature $\mathsf{I}\colon \mathsf{o} \Rightarrow \mathsf{o}$, $\mathsf{W}\colon (\mathsf{o} \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}) \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}$, and $\mathsf{S}\colon (\mathsf{o} \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}) \Rightarrow (\mathsf{o} \Rightarrow \mathsf{o}) \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}$. In order to satisfy the ground term existence condition we extend the signature by $\mathsf{c}\colon \mathsf{o} \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}$ and $\mathsf{c}'\colon \mathsf{o}$. Instantiating all head variables yields the following rules:

$$\begin{aligned} \mathsf{I}\ x &\to x \\ \mathsf{W}\ \mathsf{c}\ x &\to \mathsf{c}\ x\ x \\ \mathsf{S}\ \mathsf{c}\ \mathsf{I}\ z &\to \mathsf{c}\ z\ (\mathsf{I}\ z) \\ \mathsf{S}\ \mathsf{c}\ (\mathsf{W}\ w)\ z &\to \mathsf{c}\ z\ (\mathsf{W}\ w\ z) \\ \mathsf{S}\ \mathsf{c}\ (\mathsf{S}\ w\ v)\ z &\to \mathsf{c}\ z\ (\mathsf{S}\ w\ v\ z) \\ \mathsf{S}\ \mathsf{c}\ (\mathsf{c}\ w)\ z &\to \mathsf{c}\ z\ (\mathsf{c}\ w\ z) \end{aligned}$$

By transforming terms into functional form the TRS

$$\begin{aligned} 1:&\quad & \mathsf{I}_1(x) &\to x \\ 2:&\quad & \mathsf{W}_2(\mathsf{c}, x) &\to \mathsf{c}_2(x, x) \\ 3:&\quad & \mathsf{S}_3(\mathsf{c}, \mathsf{I}, z) &\to \mathsf{c}_2(z, \mathsf{I}_1(z)) \\ 4:&\quad & \mathsf{S}_3(\mathsf{c}, \mathsf{W}_1(w), z) &\to \mathsf{c}_2(z, \mathsf{W}_2(w, z)) \\ 5:&\quad & \mathsf{S}_3(\mathsf{c}, \mathsf{S}_2(w, v), z) &\to \mathsf{c}_2(z, \mathsf{S}_3(w, v, z)) \\ 6:&\quad & \mathsf{S}_3(\mathsf{c}, \mathsf{c}_1(w), z) &\to \mathsf{c}_2(z, \mathsf{c}_2(w, z)) \end{aligned}$$

is obtained. There are 3 dependency pairs:

$$\begin{aligned} 7:&\quad & \mathsf{S}_3^\sharp(\mathsf{c}, \mathsf{I}, z) &\to \mathsf{I}_1^\sharp(z) \\ 8:&\quad & \mathsf{S}_3^\sharp(\mathsf{c}, \mathsf{W}_1(w), z) &\to \mathsf{W}_2^\sharp(w, z) \\ 9:&\quad & \mathsf{S}_3^\sharp(\mathsf{c}, \mathsf{S}_2(w, v), z) &\to \mathsf{S}_3^\sharp(w, v, z) \end{aligned}$$

The approximated dependency graph contains one SCC: $\{9\}$.

- Consider the SCC $\{9\}$. By taking the simple projection π with $\pi(\mathsf{S}_3^\sharp) = 2$, the dependency pair simplifies to

  $$9:\quad \mathsf{S}_2(w, v) \to v$$

  and is compatible with the proper subterm relation.

[a] Tyrolean Termination Tool (0.03 seconds) — November 18, 2004

Fig. 3. Example output.

6 Some Implementation Details

The web interface of T_TT is written in Ruby[2] and the termination prover underlying TTT is written in Objective Caml (OCaml),[3] using the third-party libraries[4] `findlib`, `extlib`, and `pcre-ocaml`. We plan to make the OCaml source code available in the near future.

The termination prover consists of about 13,000 lines of OCaml code. About 20% is used for the manipulation of terms and rules. Another 15% is devoted to graph manipulations. This part of the code is not only used to compute dependency graph approximations, but also for precedences in KBO and LPO, and for the dependency relation which is used to compute the usable rules. The various termination methods that are provided by TTT account for less than 10% each. Most of the remaining code deals with I/O: parsing the input and producing HTML and Postscript output. For the official Termination Problems Data Base format we use parsers written in OCaml by Claude Marché. A rich OCaml library for the manipulation of terms (or rose trees) and graphs would have made our task much easier!

It is interesting to note that two of the original techniques that make TTT fast, the recursive SCC algorithm and the subterm criterion, account for just 13 and 20 lines, respectively. Especially the latter should be the method of first choice in any termination prover. To wit, of the 628 (full) termination problems for pure first-order term and string rewrite systems in the Termination Problems Data Base, 215 are proved terminating by the subterm criterion; the total time to check the whole collection is a mere 32 seconds (on the architecture mentioned in the previous section). Several of these 215 rewrite systems cannot be proved terminating by the latest release of C*i*ME [5]. (See the next section for a comparison between TTT and other termination provers.)

Concerning the implementation of simply-typed applicative rewrite systems, we use the Damas-Milner type reconstruction algorithm (see e.g. [17]) to infer the types of variables.

We conclude this section with some remarks on the implementation of base orders in TTT. The implementation of LPO follows [12] but we first check whether the current pair of terms can be oriented by the embedding order in every recursive call to LPO. This improves the efficiency by about 20%. The implementation of KBO is based on [16]. We use the "method for complete description" [8] to compute a suitable weight function. The implementation of polynomial interpretations with coefficients from $\{0, 1\}$ is based on [6, Figure 1] together with the simplification rules described in Section 4.4.1 of the same paper. The current implementation of polynomial interpretations with coefficients from $\{-1, 0, 1\}$ in TTT is rather naive. We anticipate that the recent techniques of [6] can be extended to handle negative coefficients.

[2] `http://www.ruby-lang.org/`
[3] `http://www.ocaml.org/`
[4] `http://caml.inria.fr/humps/`

7 Comparison

Needless to say, TT is not the only available tool for proving termination of rewrite systems. In this final section we compare our tool with the other systems that participated in the TRS category[5] of the termination competition that was organized as part of the 7th International Workshop on Termination.[6]

- We start our discussion with C*i*ME [5], the very first tool for automatically proving termination of rewrite systems that is still available. C*i*ME is a tool with powerful techniques for finding termination proofs based on polynomial interpretations in connection with the dependency pair method. Since C*i*ME does not support (yet) the most recent insights in the dependency pair method, it is less powerful than AProVE (described below) or TT. In contrast to TT, C*i*ME can handle rewrite systems with AC operators. As a matter of fact, termination is only a side-issue in C*i*ME. Its main strength lies in completing equational theories modulo theories like AC and C.
- CARIB*OO* [9] is a tool specializing in termination proofs for a particular evaluation strategy, like innermost evaluation or the strategies used in OBJ-like languages. The underlying proof method is based on an inductive process akin to narrowing, but its termination proving power comes from C*i*ME, which is used as an external solver. TT supports innermost termination, but no other strategies.
- Matchbox [19] is a tool that is entirely based on methods from formal language theory. These methods are especially useful for proving termination of string rewrite systems. Matchbox tries to establish termination or non-termination by using recent results on match-bounded rewriting [10]. Matchbox is not intended as a general-purpose termination prover (as its author writes in [19]).
- AProVE is the most powerful tool. Besides ordinary TRSs, it can handle logic programs, conditional rewrite systems, context-sensitive rewrite systems, and it supports rewriting modulo AC. Version 1.0 of AProVE is described in [11]. Of all existing tools, AProVE supports the most base orders and even offers several different algorithms implementing these. It incorporates virtually all recent refinements of the dependency pair method. AProVE has several methods that are not available in any other tool. We mention here the size-change principle [18], transformations for dependency pairs like narrowing and instantiation, and a modular refinement where the set of usable rules is determined after a suitable argument filtering has been computed. Despite all this, last year's termination competition version of AProVE, which further includes the methods derived from match-bounded rewriting, could handle only a few more systems than TT.

We conclude the paper by listing what we believe to be the main attractions of TT (in no particular order):

[5] `http://www.lri.fr/~marche/wst2004-competition/webform.cgi?command=trs`
[6] `http://www-i2.informatik.rwth-aachen.de/WST04/`

– T$_\top$T comes equipped with a very user-friendly web interface,
– T$_\top$T produces readable and beautifully typeset proofs,
– T$_\top$T is a very fast termination tool,
– T$_\top$T is a very powerful tool based on relatively few techniques.

References

1. T. Aoto and T. Yamada. Termination of simply typed term rewriting by translation and labelling. In *Proc. 14th RTA*, volume 2706 of *LNCS*, pages 380–394, 2003.
2. T. Aoto and T. Yamada. Termination of simply-typed applicative term rewriting systems. In *Proc. 2nd HOR*, Technical Report AIB-2004-03, RWTH Aachen, pages 61–65, 2004.
3. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
4. 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.
5. E. Contejean, C. Marché, B. Monate, and X. Urbain. C*i*ME version 2, 2000. Available at `http://cime.lri.fr/`.
6. E. Contejean, C. Marché, A.-P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. Research Report 1382, LRI, 2004.
7. N. Dershowitz. Termination by abstraction. In *Proc. 20th ICLP*, volume 3132 of *LNCS*, pages 1–18, 2004.
8. J. Dick, J. Kalmus, and U. Martin. Automating the Knuth-Bendix ordering. *Acta Infomatica*, 28:95–119, 1990.
9. O. Fissore, I. Gnaedig, and H. Kirchner. CARIBOO: An induction based proof tool for termination with strategies. In *Proc. 4th PPDP*, pages 62–73. ACM Press, 2002.
10. Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Match-bounded string rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 15:149–171, 2004.
11. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In *Proc. 15th RTA*, volume 3091 of *LNCS*, pages 210–220, 2004.
12. N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In *Proc. 14th RTA*, volume 2706 of *LNCS*, pages 311–320, 2003.
13. N. Hirokawa and A. Middeldorp. Dependency pairs revisited. In *Proc. 15th RTA*, volume 3091 of *LNCS*, pages 249–268, 2004.
14. N. Hirokawa and A. Middeldorp. Polynomial interpretations with negative coefficients. In *Proc. 7th AISC*, volume 3249 of *LNAI*, pages 185–198, 2004.
15. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 2005. To appear. A preliminary version appeared in *Proc. 19th CADE*, volume 2741 of *LNAI*, pages 32–46, 2003.
16. K. Korovin and A. Voronkov. Orienting rewrite rules with the Knuth-Bendix order. *Information and Computation*, 183:165–186, 2003.
17. B.C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
18. R. Thiemann and J. Giesl. Size-change termination for term rewriting. In *Proc. 14th RTA*, volume 2706 of *LNCS*, pages 264–278, 2003.
19. J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proc. 15th RTA*, volume 3091 of *LNCS*, pages 85–94, 2004.