TPA: Termination Proved Automatically

Adam Koprowski

Technical University of Eindhoven
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands
A.Koprowski@tue.nl

Abstract. TPA is a tool for proving termination of term rewrite systems (TRSs) in a fully automated fashion. The distinctive feature of TPA is the support for relative termination and the use of the technique of semantic labelling with natural numbers. Thanks to the latter, TPA is capable of delivering automated termination proofs for some difficult TRSs for which all other tools fail.

1 Introduction

Termination is an important concept in term rewriting and, although in general undecidable, several techniques have been developed for proving termination. The present focus in this area is on automation of the proof searching process. A number of tools have been developed to facilitate this and they are capable of finding proofs that are often fairly complicated and unlikely to be found by hand. To stimulate those developments an annual competition is being organized in which all the participating tools compete on a set of termination problems [1].

The tool TPA developed by the author is such tool aiming at proving termination of TRSs in an automatic fashion. What makes it different from all the other tools is the support for relative termination and the use of semantic labelling with natural numbers.

TPA took part in the termination competition of 2005. Only after three months of author's work, participating among well-established tools like C*i*ME [5], TTT [11] or AProVE [8], each developed by a group of people, it got the 3rd place there. Moreover, it succeeded in providing termination proofs for some systems for which all other tools failed. Two of such systems we will see in Figure 2. In particular it can solve the well-known SUBST [10] example encoding process of substitution in combinatory categorical logic for which showing termination was considered to be difficult. For a more throughout benchmark of TPA performance compared to other tools we refer to results of [1].

The following techniques are used in TPA:

- polynomial interpretations [14],
- recursive path order [6] (also over infinite signatures [13]),
- semantic labelling [16] (also with natural numbers [13]),
- dependency pairs [3],

– dummy elimination [7] and
– reduction of right hand sides [18].

TPA is freely available and can be downloaded from its web page:

`http://www.win.tue.nl/tpa`

It is written in Objective Caml (OCaml)[1] and is available in native code for Linux platforms and as a byte code which can be run on any platform for which OCaml interpreter is available.

2 Preliminaries

For a signature Σ and a set of variables $\mathcal{V}$ we denote the set of terms over Σ and $\mathcal{V}$ by $\mathcal{T}(\Sigma, \mathcal{V})$. We denote by $\mathsf{Var}(t)$ a set of variables occurring in term t. A *rewrite rule* is a pair (ℓ, r), written $\ell \to r$ with $\ell, r \in \mathcal{T}(\Sigma, \mathcal{V})$, $\ell \notin \mathcal{V}$, $\mathsf{Var}(r) \subseteq \mathsf{Var}(l)$. A *term rewriting system* (TRS) is a set of rewrite rules. The *rewrite relation* $\to_R$ for a TRS R is defined as $s \to_R t$ if there exists a rewrite rule $\ell \to r \in R$, a substitution σ and a context C such that $s = C[l\sigma]$ and $t = C[r\sigma]$. A TRS R is called *terminating* ($\mathsf{SN}(R)$) is there is no infinite reduction $t_1 \to_R t_2 \to_R \ldots$, that is when $\to_R$ is a well-founded relation. For two relations R, S we define $R/S \equiv S^* \cdot R \cdot S^*$ and we say that *relative termination* between R and S holds if $\mathsf{SN}(\to_R / \to_S)$. We will then say that R terminates relative to S. Note that this is equivalent to lack of infinite reduction in $R \cup S$ with infinitely many R-steps. For relative termination problems we will refer to the rules from R as *strict rules* and to the rules from S as *non-strict rules*.

3 Motivation

With that many available tools for proving termination automatically it seems natural to ask why creating yet another one? There are three main reasons why TPA has been developed and they are listed below in no particular order.

- **Semantic labelling with natural numbers** Semantic labelling is a transformational technique for proving termination of TRSs [16]. Its variant with the model over two or three element sets is used in some tools. However the infinite model variants were considered not to be suitable for automation so far. Indeed they pose some difficulties as the labelled TRS and its signature are usually infinite.
  The author's hope was that automation of semantic labelling over infinite sets (natural numbers in particular) can be accomplished and that it will be a fruitful technique for proving termination of TRSs. In fact TPA started as a prototype to verify this conjecture. The experiment turned out to be rather successful [13] and the prototype grew into a tool on its own. We will treat semantic labelling with natural numbers in some more detail in Section 4.4.

[1] `http://www.ocaml.org`

- **Relative termination** In our opinion the notion of relative termination is very natural. First of all termination is the special case of relative termination as we have $\mathsf{SN}(R) \equiv \mathsf{SN}(R/\emptyset)$. The concept of relative termination is also very closely related to the top termination and hence to the dependency pairs method [3].
  Giesl and Zantema proposed a method for verification of liveness properties using rewriting techniques [9]. In continuation of this line of research Koprowski and Zantema extended the framework to deal with fairness and there relative termination turned out to be a very natural and needed concept [12].
  Lucas and Meseguer conducted research on termination of concurrent systems under fairness assumptions [15]. In their setting they again use the notion of relative termination to establish the property of fair-termination. They used TPA for proving relative termination.
  Still up till now the support for relative termination was very limited. TORPA [17] incorporates this notion but is limited to string rewriting. TEPARLA is another tool capable of proving relative termination but it is not actively developed anymore. So being able to deal with relative termination problems for TRSs was one of the motivations for developing TPA.
- **CoLoR** CoLoR[2] stands for *Coq library on rewriting and termination*. It is an initiative aiming at proving theoretical results from term rewriting in the theorem prover Coq. The ultimate goal is to (automatically) transform the termination proof candidates produced by termination tools into formal Coq proofs certifying termination.
  This will involve some cooperation between tool authors and CoLoR developers. Presently the output produced by termination tools is just an informal, textual description of the termination proof. Certification of such results will require more explicit output. The idea within the CoLoR project is to provide a description of the proof output format in XML. Every tool being able to generate output in this common proof format would potentially be able to employ CoLoR to certify its proofs.
  The author is involved in the CoLoR project and having his own tool is helpful in bridging the gap between those two communities and propagating the idea of formal verification of termination proofs. Hence it is the author's hope that TPA will be the first tool capable of producing formally certified termination proofs.

4 Overview

We begin with a few remarks about the implementation of TPA in Section 4.1. Then in Section 4.2 we give a description of our approach to proving termination. Finally in the following sections we briefly present the theory behind techniques used in TPA as well as some details of the way in which they are used.

[2] `http://color.loria.fr`

4.1 Implementation

TPA is implemented in Objective Caml (OCaml). It is written almost fully in a functional programming style with only few fragments using imperative features of OCaml. It does not use any third-party libraries and the source code of TPA consists of about 10,000 lines of OCaml code. It is equipped with command line interface. It is available in native code for Linux platform and as OCaml byte code that can be run on any platform supported by OCaml (that includes all the major platforms).

4.2 TPA Approach to Proving Termination

First we introduce the transformation used by TPA to eliminate all the function symbols of arity greater than 2.

Definition 1. *Given TRS R over Σ we define the transformed TRS $\overline{R}$ over signature*

$$\Sigma' = \{f_i \mid f \in \Sigma,\ 1 \le i \le \mathsf{arity}(f) - 1\} \cup \{f \mid f \in \Sigma, \mathsf{arity}(f) \le 2\}$$

to consist of the rules $\overline{R} = \{\overline{\ell} \to \overline{r} \mid \ell \to r \in R\}$, where $\overline{\cdot}$ is defined as follows:

$$\begin{array}{rll} \overline{x} = x & & \\ \overline{f(t_1, \ldots, t_n)} = f(\overline{t_1}, \ldots, \overline{t_n}) & & \text{if } \mathsf{arity}(f) \le 2 \\ \overline{f(t_1, \ldots, t_n)} = f_1(\overline{t_1}, f_2(\overline{t_2}, \ldots, f_{n-1}(\overline{t_{n-1}}, \overline{t_n}))) & & \text{if } \mathsf{arity}(f) > 2 \end{array}$$

The main result concerning this transformation states that $\mathsf{SN}(R) \iff \mathsf{SN}(\overline{R})$. The ($\Leftarrow$) is crucial in this application and is easy to show. The ($\Rightarrow$), implying completeness, of the transformation is more involved but can be shown using Aoto's theorem [2].

Note that in the transformed system only constants, unary and binary symbols may occur; a fact that is heavily used in TPA as we shall see. On the other hand the original structure of the TRS is somehow obscured by the transformation which may in some cases make termination arguments more difficult.

TPA tries to use a modular approach to proving termination. Given an input TRS it first applies the transformation from Definition 1 and then it tries to apply different tactics according to the *prover configuration* that we will describe later on. As soon as one of the tactics succeeds, resulting in simplification of the TRS, the whole procedure is repeated until finally termination is proved, all tactics fail or the maximum specified search time is used and TPA stops due to timeout. For this modular approach the following theorem from [17] is essential.

Theorem 1. *Let R, S, R' and S' be TRSs such that:*

- $R \cup S = R' \cup S'$ *and* $R \cap S = R' \cap S' = \emptyset$,
- $\mathsf{SN}(R'/S')$ *and* $\mathsf{SN}((R \cap S')/(S \cap S'))$.

Then $\mathsf{SN}(R/S)$.

The *prover configuration* is a description of the way TPA should employ different tactics in order to prove termination. In the present version of the tool the configuration is embedded in the source code but it is only a matter of providing the user interface to let the user specify custom configurations. Below is a snippet from the OCaml source code of TPA with a slightly simplified description of the prover configuration.

```
type proverStep = ElimDummy
                | ReduceRHS
                | PolyInt of PolyInterp.configuration * bool
                | SemLabBool of SemLabBool.semLabCfg
                | SemLabNat of PolyInterp.configuration * proverConfig
                | Rpo of Rpo.configuration
                | DP of proverConfig
                | Parallel of proverConfig list
and proverConfig = proverStep list
```

proverConfig corresponds to the prover configuration and consists of a list of steps that should be taken successively in an attempt to prove termination. The possible steps include:

- `ElimDummy`: dummy elimination transformation [7].
- `ReduceRHS`: reduction of right hand sides [18].
- `PolyInt`: polynomial interpretations [14] (see Section 4.3)
- `SemLabBool`: semantic labelling with boolean values [16] (see Section 4.4).
- `SemLabNat`: semantic labelling with natural numbers [13] (see Section 4.4).
- `Rpo`: recursive path order [6] (see Section 4.5).
- `DP`: a simple variant of dependency pairs method [3] (see Section 4.6).
- `Parallel`: this meta-tactic allows to try few different approaches in parallel using threads. As soon as one of them succeeds the other are abandoned and the termination procedure continues with the initial configuration.

Every transformational technique (like semantic labelling or dependency pair method) is parameterized by the configuration to be used after the transformation. Hence such technique is considered to be successful if the TRS can be simplified by applying transformation and all those techniques (for semantic labelling resulting system is subject to un-labelling at the end of this procedure).

Note that the proof search procedure in TPA is fully deterministic. The only non-deterministic behavior can occur due to the use of threads for parallel computations.

The present configuration of TPA is as follows:

- simple transformations: dummy elimination and reduction of right sides are tried,
- then a simple argument of counting different symbols in left and right hand sides of the rules is tried (which corresponds to the use of polynomial interpretations with identity and successor as the only interpretation functions).
- then an attempt is made at direct termination proof with RPO.

– If all of the above fail or cannot be applied anymore then the following tactics are all executed in parallel:
  - semantic labelling with natural numbers,
  - semantic labelling with booleans with two different sets of default interpretations and also with recursive labelling (see Section 4.4),
  - polynomial interpretations method with an extended set of interpretations and
  - the dependency pairs approach.

Now we will describe the main techniques used in TPA in some more detail.

4.3 Polynomial Interpretations

The idea of proving termination with polynomial interpretations goes back to [14], [4]. We briefly present the theory for this approach.

Let Σ be a signature and let A be a non-empty set. Now for every function symbol $f \in \Sigma$ of arity n fix an *interpretation function* $f_A : A^n \to A$. Let $\alpha : \mathcal{V} \to A$ be an assignment to variables. We define the term evaluation $[\cdot]^\alpha : \mathcal{T}(\Sigma, \mathcal{V}) \to A$:

$$[x]^\alpha = \alpha(x) \qquad [f(t_1, \ldots, t_n)]^\alpha = f_A([t_1]^\alpha, \ldots, [t_n]^\alpha)$$

Theorem 2. *Let A be a non-empty set, $>$ be a well-founded order on A. Let $f_A : A^n \to A$ be strictly monotone for every $f : \Sigma$. Let R, S be disjoint TRSs over Σ. If $\forall \alpha\ [\ell]^\alpha > [r]^\alpha$ for every $\ell \to r \in R$ and $\forall \alpha\ [\ell]^\alpha \geq [r]^\alpha$ for every $\ell \to r \in S$ then $\mathsf{SN}(R/S)$.*

This approach of monotone algebras goes back to Lankford. If for the set A we take $\mathbb{N} \setminus \{0, 1\}$, for $>$ usual comparison on $\mathbb{N}$ and all f_A are polynomials then this approach is called *polynomial interpretations*.

TPA uses a fixed (per arity) set of polynomial interpretations. To further limit the search space for every arity there is a default interpretation and in the search procedure only limited number (being a parameter of the search procedure) of deviations from those standard interpretations is allowed. The standard interpretations are 2 for constants, $\lambda x.x$ for unary symbols and $\lambda xy.x + y - 2$ for binary symbols (corresponding to zero, identity and summation in $\mathbb{N}$).

4.4 Semantic Labelling

The technique of semantic labelling goes back to Zantema [16]. We present its variant that is used in TPA.

Let A be a non-empty set. For every $f \in \Sigma$ fix an interpretation function $f_A : A^n \to A$, where n is the arity of f. We define the term evaluation function $[\cdot]$ as in Section 4.3. We define the extended signature $\Sigma_L = \{f_{l_1,\ldots,l_n} \mid f \in \Sigma;\ l_1, \ldots, l_n \in A\}$ and the labelling function on terms $lab : \mathcal{T}(\Sigma, \mathcal{V}) \times A^{\mathcal{V}} \to \mathcal{T}(\Sigma, \mathcal{V})$ as:

$$\mathsf{lab}(x, \alpha) = x$$
$$\mathsf{lab}(f(t_1, \ldots, t_n), \alpha) = f_{[t_1]^\alpha,\ldots,[t_n]^\alpha}(\mathsf{lab}(t_1, \alpha), \ldots, \mathsf{lab}(t_n, \alpha))$$

For TRS R we define its labelled version over Σ_L as $\mathsf{lab}(R) = \{\mathsf{lab}(\ell) \to \mathsf{lab}(r) \mid \ell \to r \in R, \text{for all } \alpha : \mathcal{V} \to A\}$. We define Decr to be the TRS consisting of the rules: $f_{s_1,\ldots,s_i,\ldots,s_n}(x_1,\ldots,x_n) \to f_{s_1,\ldots,s'_i,\ldots,s_n}(x_1,\ldots,x_n)$ for all $1 \leq i \leq n; s_1,\ldots,s_n \in A; s'_i \in A$ and $s_i > s'_i$.

Now the main theorem for semantic labelling reads:

Theorem 3. *Let R, S be two disjoint TRSs over Σ. Let $>$ be a well-founded order on a non-empty set A. Let $f_A : A^n \to A$ for every $f \in \Sigma$ be weakly monotonic in all arguments and such that:*

$$[\ell]^\alpha \geq [r]^\alpha \text{ for all } \ell \to r \in R \cup S \text{ and all } \alpha : \mathcal{V} \to A$$

Then $\mathsf{SN}(R/S)$ *iff* $\mathsf{SN}(\mathsf{lab}(R)/(\mathsf{lab}(S) \cup \mathsf{Decr}))$

In a sense semantic labelling is a central technique for TPA. Its two variants are used: one rather standard where A is the set $\{0,1\}$ and another, presently implemented in no other tool except for TPA, where $A = \mathbb{N}$. Note that it is possible to label already labelled system and TPA makes use of this fact doing recursive labelling with boolean models.

The difficulty posed by using semantic labelling with $\mathbb{N}$ is that the labelled system has infinite signature and infinitely many rules. Hence all the standard techniques may need to be somehow adopted in order to be applied to such systems. In the present version of TPA two techniques are used in combination with semantic labelling with natural numbers: RPO and polynomial interpretations. For details about using RPO on systems with infinite signature we refer the reader to [13].

Polynomial interpretations are incorporated into this setting in a straightforward way. Denote by Φ_0, Φ_1 and Φ_2 sets of polynomial interpretations for constants, unary and binary symbols respectively. Constants get no label but for unary and binary symbols after labelling the following interpretations are used:

$$[f_i(x)] = \phi(x) \text{ or } \phi(x) + i \text{ with } \phi \in \Phi_1$$
$$[f_{i,j}(x,y)] = \phi(x,y) \text{ or } \phi(x,y) + i \text{ or } \phi(x,y) + j \text{ or } \phi(x,y) + i + j \text{ with } \phi \in \Phi_2$$

We will illustrate this approach on a simple relative termination problem.

Example 1. $R = \{T(I(x),y) \to T(x,y)\}$, $S = \{T(x,y) \to T(x,I(y))\}$.

There are two rules in this system: the rule from R allows to remove one I symbol from the left argument of T and the rule from S allows introducing this symbol in the right argument of T but it can be applied only finitely many times and thus $\mathsf{SN}(\to_R \,/\, \to_S)$. Figure 1 shows the proof produced by TPA for this system. TPA prints R-rules as `l -> r` and S-rules as `l ->= r`.

In Figure 2 we present two TRSs that can be solved by TPA with the use of semantic labelling with natural numbers and presently no other tool can deal with them. The first one is the SUBST TRS encoding the process of substitution in combinatory categorical logic and the second one is the GCD TRS encoding the computation of the greatest common divisor. For termination proofs of those systems generated by TPA we refer the reader to its homepage.

```
  TPA v.1.0
Result: TRS is terminating
[1] TRS loaded from input file:
     (1) T(I(x),y) -> T(x,y)
     (2) T(x,y) ->= T(x,I(y))
[2] Label this TRS using following interpretation over N\{0,1}:
     [T(x,y)] = 2
     [I(x)] = x + 1
    This interpretation is a model and yields the following TRS:
     (1) T{i + 1,j}(I{i}(x),y) -> T{i,j}(x,y)
     (2) T{i,j}(x,y) ->= T{i,j + 1}(x,I{j}(y))
[3] Use the following polynomial interpretation:
     [T_{i,j}(x,y)] = x + y - 2 + i
     [I_{i}(x)] = x
    Remove rules with left hand side strictly bigger than right hand
    side: (1)
[4] Since there are no remaining strict rules, relative termination is
    proven.
```

Fig. 1. Proof generated by TPA for the system from Example 1. For the purpose of the presentation the proof has been made slightly less verbose.

4.5 RPO

Recursive path order (RPO) is an ordering introduced by Dershowitz [6].

Definition 2 (RPO). *Given an order $\triangleright$ on function symbols called* precedence *and a* status *function τ associating every function symbol with either lexicographic or multiset status, we define the* RPO ordering $\succ_{rpo}$ *as follows:*

$s = f(s_1, \ldots, s_n) \succ_{rpo} g(t_1, \ldots, t_m) = t$ *iff one of the following holds:*

- $s_i \succeq_{rpo} t$ *for some* $1 \leq i \leq n$.
- $f \triangleright g$ *and* $s \succ_{rpo} t_i$ *for all* $1 \leq i \leq m$
- $f = g$ *and* $(s_1, \ldots, s_n) \succ_{rpo}^{\tau(f)} (t_1, \ldots, t_m)$

Theorem 4. *If $\triangleright$ is well-founded and $\ell \succ_{rpo} r$ for all $\ell \to r \in R$ then* $\mathsf{SN}(R)$.

In TPA RPO is implemented in this form with both lexicographic (left-to-right and right-to-left) and multiset statuses. However due to the transformation from Definition 1 the power of multiset status is limited and in the most recent configuration it is not even used. To avoid extensive branching in TPA it is also possible to use a slightly weaker variant where the first clause of the RPO definition is used only if t can be embedded in s.

RPO in its standard variant is not very suitable for proving relative termination since the equality part of the preorder it generates is rather small. Some experiments have been made to make it more suitable for proving relative termination but they were not very successful.

Note that due to the use of semantic labelling with natural numbers also a variant of RPO for infinite signatures is used in TPA. See [13] for more details.

$$
\begin{aligned}
\mathsf{min}(x,0) &\to 0\\
\mathsf{min}(0,y) &\to 0\\
\mathsf{min}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{s}(\mathsf{min}(x,y))\\
\mathsf{max}(x,0) &\to x\\
\mathsf{max}(0,y) &\to y\\
\mathsf{max}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{s}(\mathsf{max}(x,y))\\
-(x,0) &\to x\\
-(\mathsf{s}(x),\mathsf{s}(y)) &\to -(x,y)\\
\mathsf{gcd}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{gcd}(-(\mathsf{s}(\mathsf{max}(x,y)),\\
&\qquad \mathsf{s}(\mathsf{min}(x,y))),\mathsf{s}(\mathsf{min}(x,y)))
\end{aligned}
$$

$$
\begin{aligned}
\lambda(x)\circ y &\to \lambda(x\circ(1\cdot(y\circ\uparrow)))\\
(x\cdot y)\circ z &\to (x\circ z)\cdot(y\circ z)\\
(x\circ y)\circ z &\to x\circ(y\circ z)\\
\mathsf{id}\circ x &\to x\\
1\circ\mathsf{id} &\to 1\\
\uparrow\circ\mathsf{id} &\to 1\\
1\circ(x\cdot y) &\to x\\
\uparrow\circ(x\cdot y) &\to y
\end{aligned}
$$

Fig. 2. TRSs: GCD(left) and SUBST (right)

4.6 Dependency Pairs

The technique of dependency pairs was introduced in [3] and since then is a central technique for many termination provers. Due to that fact we decided to concentrate on other techniques and in TPA dependency pairs play only a minor role.

Definition 3 (Dependency pairs). *Let R be TRS over Σ. We split Σ into* defined symbols $\Sigma_D = \{root(\ell) \mid \ell \to r \in R\}$ *and* constructor symbols $\Sigma_C = \Sigma \setminus \Sigma_C$. *We extend signature Σ with fresh symbols for every defined symbol: $\Sigma' = \Sigma \cup \{\overline{f} \mid f \in \Sigma_D\}$.*

We define a TRS $\mathsf{DP}(R)$ *over Σ' to represent dependency pairs of R:*

$$\mathsf{DP}(R) = \{\overline{f}(s_1,\ldots,s_n) \to \overline{g}(t_1,\ldots,t_n) \mid g \in \Sigma_D \wedge f(s_1,\ldots,s_n) \to C[g(t_1,\ldots,t_n)] \in R\}$$

The main theorem, as it is used in TPA, relates termination of R with relative termination of $\mathsf{DP}(R)/R$.

Theorem 5. *Let R be TRS over Σ. Then* $\mathsf{SN}(R)$ *iff* $\mathsf{SN}(\mathsf{DP}(R)/R)$.

So using the dependency pair method TPA transforms a termination problem to a, hopefully easier, relative termination problem. Moreover it computes a dependency graph approximation and treats its strongly connected components separately but it does not use more involved refinements of dependency pairs method such as argument filtering or narrowing.

5 Conclusions and Further Research

TPA is a new termination tool that got the 3rd place in the international termination competition in 2005. It also proved that the approach of semantic labelling with natural numbers can be successfully used for proving termination

automatically; using this technique TPA could prove termination of systems for which all other tools failed.

Clearly there is plenty of room for improvement and further research. For instance the idea of semantic labelling with natural numbers can be investigated further and techniques other than only RPO and polynomial interpretations can be adopted and applied on labelled systems. We also want to improve the proving capabilities of TPA in the subject of relative termination, either by developing new techniques devoted particularly to relative termination or by refining existing termination techniques. Finally we want to propagate the idea of certified termination by continuing developments in the CoLoR project.

References

1. The termination competition. `http://www.lri.fr/~marche/termination-competition`.
2. T. Aoto and T.Yamada. Termination of Simply Typed Term Rewriting by Translation and Labelling. In Proceedings of *RTA*, LNCS 2706:380–394, 2003.
3. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theor. Comput. Sci.*, 236(1-2):133–178, 2000.
4. A. B. Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Sci. Comput. Program.*, 9(2):137–159, 1987.
5. E. Contejean, C. Marché, B. Monate, and X. Urbain. Proving termination of rewriting with CiME. 71–73, 2003. `http://cime.lri.fr`.
6. N. Dershowitz. Orderings for term-rewriting systems. *Theor. Comput. Sci.*, 17:279–301, 1982.
7. M. C. F. Ferreira and H. Zantema. Dummy elimination: Making termination easier. In Proceedings of *FCT*, LNCS 965:243–252, 1995.
8. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In Proceedings of *RTA*, LNCS 3091:210–220, 2004.
9. J. Giesl and H. Zantema. Liveness in rewriting. In Proceedings of *RTA*, LNCS 2706:321–336, 2003.
10. T. Hardin and A. Laville. Proof of termination of the rewriting system SUBST on CCL. *Theor. Comput. Sci.*, 46(2-3):305–312, 1986.
11. N. Hirokawa and A. Middeldorp. Tyrolean termination tool. In Proceedings of *RTA*, LNCS 3467:175–184, 2005.
12. A. Koprowski and H. Zantema. Proving liveness with fairness using rewriting. In Proceedings of *FroCoS*, LNCS 3717:232–247, 2005.
13. A. Koprowski and H. Zantema. Automation of recursive path ordering for infinite labelled rewrite systems. In J.Harrison, editor, *IJCAR '06*, LNCS, 2006.
14. D. S. Lankford. On proving term rewriting systems are noetherian. Tech. Rep. MTP-3, Louisiana Tech. Univ., Ruston, 1979.
15. S. Lucas and J. Meseguer. Termination of fair computations in term rewriting. In Proceedings of *LPAR*, LNCS 3835:184–198, 2005.
16. H. Zantema. Termination of term rewriting by semantic labelling. *Fundam. Inform.*, 24(1/2):89–105, 1995.
17. H. Zantema. Torpa: Termination of rewriting proved automatically. In Proceedings of *RTA*, LNCS 3091:95–104, 2004.
18. H. Zantema. Reducing right-hand sides for termination. In *Processes, Terms and Cycles*, LNCS 3838:173–197, 2005.