Implementing RPO and POLO using SAT*

Carsten Fuhs[1], Peter Schneider-Kamp[1], Rene Thiemann[1], Jürgen Giesl[1], Elena Annov[2], Michael Codish[2], Aart Middeldorp[3], and Harald Zankl[3]

[1] LuFG Informatik 2, RWTH Aachen, Germany,
{fuhs,psk,thiemann,giesl}@informatik.rwth-aachen.de
[2] Department of Computer Science, Ben-Gurion University, Israel,
{annov,mcodish}@cs.bgu.ac.il
[3] Institute of Computer Science, University of Innsbruck, Austria,
{aart.middeldorp,harald.zankl}@uibk.ac.at

Abstract. Well-founded orders are the most basic, but also most important ingredient to virtually all termination analyses. Numerous fully automated search algorithms for these classes have therefore been devised and implemented in termination tools. Unfortunately, for termination problems occurring in practice, the performance of existing algorithms is often insufficient.
Performance can be improved significantly by reducing these search problems to decision problems for which more efficient algorithms already exist. Here, we introduce an encoding of RPO and POLO to the satisfiability of propositional logic (SAT). We implemented these encodings in our termination tool AProVE. Extensive experiments have shown that one can obtain speedups in orders of magnitude by this encoding and the application of modern SAT solvers.

Keywords. termination, term rewriting, SAT solving, recursive path order, polynomial interpretation, dependency pairs

1 Introduction

Well-founded orders are the most basic, but also the most important ingredient to virtually all termination analyses. The recursive path order with status (RPO) and polynomial interpretations (POLO) are two classes that are among the most popular ones in the termination analysis of term rewrite systems (TRSs). Numerous fully automated search algorithms for these classes have therefore been devised and implemented in termination tools.

Unfortunately, the performance of these algorithms on all but the smallest termination problems has been lacking. E.g., recently developed transformations from programming languages like Haskell [2] or Prolog [14] allow to apply termination tools that were developed for term rewrite systems to real programming languages. The results of the transformations are often of non-trivial size, though, and cannot be handled efficiently by the existing algorithms.

* Supported by the DFG grant GI 274/5-1 and the FWF project P18763.

Dagstuhl Seminar Proceedings 07401
Deduction and Decision Procedures
http://drops.dagstuhl.de/opus/volltexte/2007/1249

The need for more efficient search algorithms has triggered research in reducing these search problems into decision problems for which more efficient algorithms already exist. Here, we introduce an encoding of RPO and POLO to the satisfiability of propositional logic (SAT).

Since last year, several papers have illustrated the potential in applying SAT solvers for termination analysis of TRSs. The key idea is classic: the termination problem for a TRS is encoded to a propositional formula that is satisfiable if the TRS has the desired termination property (for RPO we even have "iff"). Satisfiability is decided using state of the art SAT solvers.

For the lexicographic path order (LPO) and the Knuth-Bendix order (KBO), such encodings are described in [2,3,12] and [16] respectively. This draft extends those works by introducing an encoding for RPO and for POLO. The main new and interesting components[4] are the encodings for the lexicographic comparison w.r.t. permutation and for the multiset extension of the base order for RPO in Section 3 as well as the encoding of non-linear inequalities over integers for POLO in Section 5.

All of the described encodings have been implemented in the termination analysis tool AProVE. Extensive experiments indicate speedups in orders of magnitude by these encodings and by the application of modern SAT solvers (cf. Section 6).

2 Recursive Path Order (RPO)

Most termination methods for TRSs transform the termination problem into a set of inequalities between terms. For example, the classical approach is to generate inequalities $\ell \succ r$ for all rules $\ell \to r$. One way to instantiate the order $\succ$ is to use a recursive path order with status.

Let $\geq_{\mathcal{F}}$ denote a quasi-order (a so-called *precedence*) on the set of function symbols $\mathcal{F}$ and let $>_{\mathcal{F}} = (\geq_{\mathcal{F}} \setminus \leq_{\mathcal{F}})$ and $\approx_{\mathcal{F}} = (\geq_{\mathcal{F}} \cap \leq_{\mathcal{F}})$. Each precedence $\geq_{\mathcal{F}}$ and status function σ induce a recursive path order $\succ_{rpo}$ on terms. If $\ell \succ_{rpo} r$ holds for each rule $\ell \to r$ in a TRS, then the TRS is RPO-terminating.

Definition 1 (status and multiset cover). *A status function σ maps each $f \in \mathcal{F}$ of arity n to* `mul` *or to a permutation on $\{1,\ldots,n\}$ and each pair of tuples of terms $\bar{s} = \langle s_1,\ldots,s_n\rangle$ and $\bar{t} = \langle t_1,\ldots,t_m\rangle$ to a multiset cover (γ,ε) s.t. $\gamma : \{1,\ldots,m\} \to \{1,\ldots,n\}$, $\varepsilon : \{1,\ldots,n\} \to \{false, true\}$ and for each $1 \leq i \leq n$, if $\varepsilon(i)$ (indicating equality) then $\{j \mid \gamma(j) = i\}$ is a singleton set.*

The status of symbol f indicates if the arguments of a term rooted by f are compared lexicographically w.r.t. a permutation or as multisets. That a status also maps pairs $\bar{s}$ and $\bar{t}$ to a multiset cover is non-standard, but facilitates the encoding to SAT. Such a multiset cover for $\bar{s}$ and $\bar{t}$ is a mapping between indices which specifies which element $i = \gamma(j)$ of $\bar{s}$ covers element j of $\bar{t}$ and in terms of $\varepsilon(i)$ if that covering is by equality.

[4] All results described in this draft have been published in two full papers [6,15].

Definition 2 (recursive path order with status). *For a precedence $\geq_{\mathcal{F}}$ and status function σ we define the relations $\succ_{rpo}$ and $\sim_{rpo}$ on terms. We use the notation $\bar{s} = \langle s_1, \ldots, s_n \rangle$ and $\bar{t} = \langle t_1, \ldots, t_m \rangle$.*

- *$s \succ_{rpo} t$ iff $s = f(\bar{s})$ and one of the following holds:*
  (1) $s_i \succ_{rpo} t$ or $s_i \sim_{rpo} t$ for some $1 \leq i \leq n$; or
  (2) $t = g(\bar{t})$ and $s \succ_{rpo} t_j$ for all $1 \leq j \leq m$ and either:
  (i) $f >_{\mathcal{F}} g$ or (ii) $f \approx_{\mathcal{F}} g$ and $\bar{s} \succ_{rpo}^{f,g} \bar{t}$;
- *$s \sim_{rpo} t$ iff (a) $s = t$; or (b) $s = f(\bar{s})$, $t = g(\bar{t})$, $f \approx_{\mathcal{F}} g$, and $\bar{s} \sim_{rpo}^{f,g} \bar{t}$;*

where $\succ_{rpo}^{f,g}$ and $\sim_{rpo}^{f,g}$ are the tuple extensions of $\succ_{rpo}$ and $\sim_{rpo}$ defined by:

- *$\langle s_1, \ldots, s_n \rangle \succ_{rpo}^{f,g} \langle t_1, \ldots, t_m \rangle$ iff one of the following holds:*
  (1) σ maps f and g to permutations μ_f and μ_g; and
  $\mu_f\langle s_1, \ldots, s_n \rangle \succ_{rpo}^{lex} \mu_g\langle t_1, \ldots, t_n \rangle$ where $\langle u_1, \ldots, u_n \rangle \succ_{rpo}^{lex} \langle v_1, \ldots, v_m \rangle$ iff (a) $m = 0$ and $n > 0$; or (b) $u_1 \succ_{rpo} v_1$; or
  (c) $u_1 \sim_{rpo} v_1$ and $\langle u_2, \ldots, u_n \rangle \succ_{rpo}^{lex} \langle v_2, \ldots, v_m \rangle$;
  (2) σ maps f and g to `mul`; and $\langle \bar{s}, \bar{t} \rangle$ to a multiset cover (γ, ε) such that for all i, j, if $\gamma(j) = i$ then $\varepsilon(i) \rightarrow s_i \sim_{rpo} t_j$ and $\neg\varepsilon(i) \rightarrow s_i \succ_{rpo} t_j$; and for some i, $\neg\varepsilon(i)$, i.e., some s_i is not used for equality.
- *$\langle s_1, \ldots, s_n \rangle \sim_{rpo}^{f,g} \langle t_1, \ldots, t_m \rangle$ iff $n = m$ and one of the following holds:*
  (1) σ maps f and g to μ_f and μ_g; and for all i, $s_{\mu_f(i)} \sim_{rpo} t_{\mu_g(i)}$.
  (2) σ maps f and g to `mul`; and $\langle \bar{s}, \bar{t} \rangle$ to a multiset cover (γ, ε) such that for all i, $\varepsilon(i)$ and for some $1 \leq j \leq m$ we have $\gamma(j) = i$ and $s_i \sim_{rpo} t_j$.

Definition 2 can be specialized to other standard path orders by taking specific forms of status functions: lexicographic path order (LPO) when σ maps all symbols to the identity permutation; lexicographic path order w.r.t. permutation (LPOS) when σ maps all symbols to some permutation; multiset path order (MPO) when σ maps all symbols to `mul`.

The RPO termination problem is to determine for a given TRS if there exists a precedence and a status function such that the system is RPO terminating. There are two variants of the problem: "strict-" and "quasi-RPO termination" depending on if the precedence, $\geq_{\mathcal{F}}$, is required to be strict or not. The corresponding decision problems, strict- and quasi-RPO termination, are decidable and NP complete [4]. In this draft we address the implementation of decision procedures for RPO termination problems by encoding them into corresponding SAT problems.

3 Encoding RPO problems

We introduce an encoding τ which maps constraints of the form $s \succ_{rpo} t$ to propositional statements about the status and the precedence of the symbols in the terms s and t. A satisfying assignment for the encoding of such a constraint indicates a precedence and a status function such that the constraint holds.

The first part of the encoding is straightforward and similar to [2,3]. All "missing" cases (e.g., $\tau(x \succ_{rpo} t)$ for variables x) are defined to be *false*.

$$\tau(f(\bar{s}) \succ_{rpo} t) = \bigvee_{i=1}^{n} (\tau(s_i \succ_{rpo} t) \vee \tau(s_i \sim_{rpo} t)) \quad \vee \quad \tau_2(f(\bar{s}) \succ_{rpo} t)$$

$$\tau_2(f(\bar{s}) \succ_{rpo} g(\bar{t})) = \bigwedge_{j=1}^{m} \tau(f(\bar{s}) \succ_{rpo} t_j) \wedge \Big((f \succ_{\mathcal{F}} g) \vee ((f \approx_{\mathcal{F}} g) \wedge \tau(\bar{s} \succ_{rpo}^{f,g} \bar{t}))\Big)$$

$$\tau(s \sim_{rpo} s) = \mathit{true} \qquad \tau(f(\bar{s}) \sim_{rpo} g(\bar{t})) = (f \approx_{\mathcal{F}} g) \wedge \tau(\bar{s} \sim_{rpo}^{f,g} \bar{t})$$

We proceed to show how to encode lexicographic comparisons w.r.t. permutation and multiset comparisons by $\succ_{lex}^{f,g}$ and $\succ_{mul}$. Then, we combine the two into $\succ_{rpo}^{f,g}$.

With each symbol $f \in \mathcal{F}$ (of arity n) we associate n^2 propositional variables $f_{i,k}$ with $i, k \in \{1, \ldots, n\}$. Here, $f_{i,k}$ is *true* iff $\mu_f(i) = k$ (i.e., the i-th argument of $f(s_1, \ldots, s_n)$ is considered at k-th position when comparing lexicographically). For the encoding to be correct, we introduce constraints on the variables $f_{i,k}$ ensuring that they indeed correspond to a permutation on $\{1, \ldots, n\}$. To encode $\succ_{lex}^{f,g}$, we define auxiliary relations $\succ_{lex}^{f,g,k}$, where $k \in \mathbb{N}$ denotes that the k-th component of $\bar{s}$ and $\bar{t}$ is being compared. Thus, $\succ_{lex}^{f,g} = \succ_{lex}^{f,g,1}$ and we obtain:

$$\tau(\bar{s} \succ_{lex}^{f,g,k} \bar{t}) = \begin{cases} \mathit{false} & \text{if } k > n \\ \mathit{true} & \text{if } m < k \leq n \\ \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} (f_{i,k} \wedge g_{j,k} \rightarrow & \text{otherwise} \\ \quad (\tau(s_i \succ_{rpo} t_j) \vee (\tau(s_i \sim_{rpo} t_j) \wedge \tau(\bar{s} \succ_{lex}^{f,g,k+1} \bar{t})))) \end{cases}$$

$$\tau(\bar{s} \sim_{lex}^{f,g} \bar{t}) = (n = m) \wedge \left(\bigwedge_{k=1}^{n} \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} f_{i,k} \wedge g_{j,k} \rightarrow \tau(s_i \sim_{rpo} t_j) \right)$$

With each pair $\bar{s}$ and $\bar{t}$ of term tuples, we associate $n * m$ propositional variables $\gamma_{i,j}$, where $\gamma_{i,j}$ is *true* iff s_i covers t_j, and n variables ε_i, where ε_i is *true* iff s_i is used to cover a t_j by equality. For the below encoding to be correct, we introduce constraints on these variables to ensure that the implied mappings are indeed a multiset cover. Then we obtain:

$$\tau(\bar{s} \succsim_{mul} \bar{t}) = \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} (\gamma_{i,j} \rightarrow ((\varepsilon_i \rightarrow \tau(s_i \sim_{rpo} t_j)) \wedge (\neg\varepsilon_i \rightarrow \tau(s_i \succ_{rpo} t_j))))$$

$$\tau(\bar{s} \succ_{mul} \bar{t}) = \tau(\bar{s} \succsim_{mul} \bar{t}) \wedge \neg \bigwedge_{i=1}^{n} \varepsilon_i \qquad \tau(\bar{s} \sim_{mul} \bar{t}) = \tau(\bar{s} \succsim_{mul} \bar{t}) \wedge \bigwedge_{i=1}^{n} \varepsilon_i$$

Finally, to combine $\succ_{lex}^{f,g}$ and $\succ_{mul}$ into $\succ_{rpo}^{f,g}$, we introduce for each symbol $f \in \mathcal{F}$ an additional propositional variable m_f, which is *true* iff the arguments of f are to be compared as multisets (i.e., the status function maps f to `mul`). Then we encode:

$$\tau(\bar{s} \circ_{rpo}^{f,g} \bar{t}) = \Big(m_f \wedge m_g \wedge \tau(\bar{s} \circ_{mul} \bar{t})\Big) \vee \Big(\neg m_f \wedge \neg m_g \wedge \tau(\bar{s} \circ_{lex}^{f,g} \bar{t})\Big) \quad \text{for } \circ \in \{\succ, \sim\}$$

One can show that the overall size of $\tau(s \succ_{rpo} t)$ is in $\mathcal{O}(k^3)$ where k is the combined size of s and t.

Similar to Definition 2, the above encoding function τ can be specialized to other standard path orders: lexicographic path order w.r.t. permutation when m_f is set to *false* for all $f \in \mathcal{F}$; lexicographic path order when additionally $f_{i,k}$ is set to *true* iff $i = k$; multiset path order when m_f is set to *true* for all $f \in \mathcal{F}$.

Instead of the classical approach used so far (where one generates inequalities $\ell \succ r$ for all rules $\ell \to r$), an alternative is to use the dependency pair (DP) framework [1,8,9]. Here, one generates strict inequalities for the DPs and non-strict ones for the usable rules. One of the main differences is that montonicity of the strict part of the order is not required anymore. As RPO is always monotonic, using it directly is not advisable in this context. To search for RPOs where $\succ$ may be non-monotonic, one must combine the search for the order with the search for an argument filtering. How to encode this combined search to SAT is described in detail in [15].

4 Polynomial Interpretations (POLO)

Another popular method to search for terminatin orders automatically are orders based on *polynomial interpretations* [13]. The basic idea is to map terms to polynomials over the naturals.

More precisely, a polynomial interpretation $\mathcal{P}ol$ maps each n-ary function symbol f to a polynomial $f_{\mathcal{P}ol}$ over n variables $x_1, \ldots, x_n$ with coefficients from $\mathbb{N} = \{0, 1, 2, \ldots\}$. It is extended to a mapping $[\cdot]_{\mathcal{P}ol}$ on terms where $[x]_{\mathcal{P}ol} = x$ for variables x and $[f(t_1, \ldots, t_n)]_{\mathcal{P}ol} = f_{\mathcal{P}ol}\{x_1/[t_1]_{\mathcal{P}ol}, \ldots, x_n/[t_n]_{\mathcal{P}ol}\}$. We often write $[\cdot]$ if $\mathcal{P}ol$ is clear from the context. Now a term u is greater (resp. greater-equal) than v iff $[u] \geq [v] + 1$ (resp. $[u] \geq [v]$) holds for all instantiations of the variables with natural numbers.

In contrast to RPO, orders based on polynomial interpretatios can trivially be non-monotonic by having some of the coefficients be 0. Thus, in the following we directly use the more powerful dependency pair (DP) framework to generate inequalities.

Consider the following example for subtraction.

$$\begin{array}{ll} \mathsf{p}(\mathsf{0}) \to \mathsf{0} & \mathsf{minus}(x, \mathsf{0}) \to x \\ \mathsf{p}(\mathsf{s}(x)) \to x & \mathsf{minus}(x, \mathsf{s}(y)) \to \mathsf{minus}(\mathsf{p}(x), y) \end{array}$$

For the DP of the recursive minus-call we get the following constraints. Here, M is the tuple-symbol of minus.

$$\mathsf{p}(\mathsf{0}) \succsim \mathsf{0} \quad (1)\ \mathsf{p}(\mathsf{s}(x)) \succsim x \quad (2)\ \mathsf{M}(x, \mathsf{s}(y)) \succ \mathsf{M}(\mathsf{p}(x), y) \quad (3)$$

The constraints of the above example can indeed be satisfied using the polynomial interpretation $\mathcal{P}ol_1$ with $\mathsf{M}_{\mathcal{P}ol_1} = x_2$, $\mathsf{p}_{\mathcal{P}ol_1} = x_1$, $\mathsf{s}_{\mathcal{P}ol_1} = x_1 + 1$, and $\mathsf{0}_{\mathcal{P}ol_1} = 0$. Thus, termination of the example is proved.

To find such interpretations automatically, one starts with an *abstract* polynomial interpretation. In the linear case we obtain

$$f_{\mathcal{P}ol} = f_0 + f_1 x_1 + \cdots + f_n x_n \quad \text{for each } f \text{ with arity } n \tag{4}$$

where the coefficients f_i are left open. Then one translates the term constraints into polynomial constraints. In the example we obtain

$$\mathsf{p}_0 + \mathsf{p}_1\mathsf{0}_0 \geq \mathsf{0}_0 \quad (5) \qquad (\mathsf{p}_0 + \mathsf{p}_1\mathsf{s}_0) + (\mathsf{p}_1\mathsf{s}_1) * x \geq x \quad (6)$$
$$(\mathsf{M}_0 + \mathsf{M}_2\mathsf{s}_0) + \mathsf{M}_1 * x + \mathsf{M}_2\mathsf{s}_1 * y \geq (\mathsf{M}_0 + \mathsf{M}_1\mathsf{p}_0 + 1) + \mathsf{M}_1\mathsf{p}_1 * x + \mathsf{M}_2 * y \quad (7)$$

Next one simplifies these constraints by deleting the variables $x, y, \ldots$ that are (implicitly) universally quantified. To this end, instead of an inequality between polynomials we only compare the respective coefficients ("absolute positiveness" [11]). In the example, the resulting constraints are (5), (8) and (9) (using $x = 0 + 1 * x$), and (10) – (12).

$$\mathsf{p}_0 + \mathsf{p}_1\mathsf{s}_0 \geq 0 \quad (8) \qquad \mathsf{p}_1\mathsf{s}_1 \geq 1 \quad (9)$$
$$\mathsf{M}_2\mathsf{s}_0 \geq \mathsf{M}_1\mathsf{p}_0 + 1 \quad (10) \qquad \mathsf{M}_1 \geq \mathsf{M}_1\mathsf{p}_1 \quad (11) \qquad \mathsf{M}_2\mathsf{s}_1 \geq \mathsf{M}_2 \quad (12)$$

Now to prove termination one has to show the *satisfiability* of such *Diophantine constraints* over the naturals. In the example, a solution of the constraints is $\mathsf{0}_0 = \mathsf{p}_0 = \mathsf{M}_0 = \mathsf{M}_1 = 0$ and $\mathsf{p}_1 = \mathsf{s}_0 = \mathsf{s}_1 = \mathsf{M}_2 = 1$. In this way, the abstract interpretation is turned into the polynomial interpretation $\mathcal{P}ol_1$.

In the next section, we show how to check satisfiability of Diophantine constraints by encoding to satisfiability of propositional logic.

5 Encoding Diophantine Constraints to SAT

To encode Diophantine constraints into SAT we first present a mapping $||\cdot||$ from polynomials to tuples of propositional formulas which are interpreted as *binary representations* of the polynomials. We restrict the search to coefficients in the range $\{0, \ldots, 2^k - 1\}$ for a fixed k. Then each coefficient f is encoded into $||f|| = \langle f^{k-1}, \ldots, f^0 \rangle$ where $f^0, \ldots, f^{k-1}$ are propositional variables. Similarly, a natural number $n = b_\ell * 2^\ell + \ldots b_1 * 2^1 + b_0$ is encoded into $||n|| = \langle b_\ell, \ldots, b_1, b_0 \rangle$ where 0 and 1 are identified with false and true. So if $k = 2$ then $||\mathsf{s}_0|| = \langle \mathsf{s}_0^1, \mathsf{s}_0^0 \rangle$ and $||6|| = \langle 1, 1, 0 \rangle$. For addition and multiplication, we introduce operations B^+ and B^* on tuples of propositional formulas and define

$$||p + q|| = B^+(||p||, ||q||) \quad \text{and} \quad ||p * q|| = B^*(||p||, ||q||)$$

for all polynomials p and q. For B^+ we essentially use the idea of a ripple-carry-adder. The details are presented in [6]. For example $||\mathsf{s}_0 + 6|| = \langle \mathsf{s}_0^1, \neg\mathsf{s}_0^1, \neg\mathsf{s}_0^1, \mathsf{s}_0^0 \rangle$.

We encode multiplication by summing up partial products as follows:

- $B^*(\langle\varphi_1,\ldots,\varphi_n\rangle,\langle\psi\rangle) = \langle\varphi_1\wedge\psi,\ldots,\varphi_n\wedge\psi\rangle$
- $B^*(\langle\varphi_1,\ldots,\varphi_n\rangle,\langle\psi_1,\ldots,\psi_m\rangle) = B^+(\langle\varphi_1\wedge\psi_1,\ldots,\varphi_n\wedge\psi_1,\overbrace{0,\ldots,0}^{m-1\text{ times}}\rangle,$ $B^*(\langle\varphi_1,\ldots,\varphi_n\rangle,\langle\psi_2,\ldots,\psi_m\rangle)\,)$ if $m\geq 2$.

Now we extend $||\cdot||$ to map each Diophantine constraint to a formula (not to a tuple). To this end, we define the operation $B^{\geq}$ which encodes comparisons:

$$||p\geq q|| = B^{\geq}(||p||,||q||)$$

For $B^{\geq}$ we apply zero-padding and compare tuples lexicographically:

- $B^{\geq}(\langle\varphi\rangle,\langle\psi\rangle) = \psi\rightarrow\varphi$
- $B^{\geq}(\langle\varphi_1,\ldots,\varphi_n\rangle,\langle\psi_1,\ldots,\psi_n\rangle) = (\varphi_1\wedge\neg\psi_1)\vee$ $((\varphi_1\leftrightarrow\psi_1)\wedge B^{\geq}(\langle\varphi_2,\ldots,\varphi_n\rangle,\langle\psi_2,\ldots,\psi_n\rangle))$ if $n\geq 2$.

So to determine the satisfiability of a set of Diophantine constraints $p_i\geq q_i$ with coefficients from $\{0,\ldots,2^k-1\}$, we encode it as a conjunction $\bigwedge_i||p_i\geq q_i||$ of propositional formulas.

Then we use a SAT solver to find an assignment for the coefficients. Note that the space complexity of our encoding is polynomial. More precisely, whenever all numbers in "$p\geq q$" are smaller than 2^k-1, then the size of $||p\geq q||$ is in $\mathcal{O}(|p\geq q|^2*k^2)$.

6 Implementation and Experiments

We have implemented the encodings of RPO and POLO in the termination analyzer AProVE [7]. The implementation of RPO is modularized in such a way that all path orders using lexicographic and/or multiset comparisons can be encoded. The implementation supports also RPO with argument filters (cf. [15]) and POLO with negative constants (cf. [6]).

The tables below summarize the results of the experiments running on the set of 865 TRSs from the TPDB 2006 [17]. All experiments were run on a 2.2 GHz AMD Athlon 64 with a time limit of 60 seconds (comparable to the setting of the annual International Termination Competition 2006). For each encoding we provide the number of TRSs which can be proved terminating (with the number of time-outs in brackets) and the total analysis times (in seconds) for the full collection.

In the first table, the first two rows compare the performance of our new SAT-based approach to the dedicated solvers for path orders in AProVE 1.2 which do not use SAT solving. The third and the fourth row apply path orders (combined with argument filters) within the dependency pair framework.

The columns show data for LPO with strict and quasi-precedence (denoted *lpo*/*qlpo*), for LPO with status (*lpos*/*qlpos*), for MPO (*mpo*/*qmpo*), and, finally, for RPO (*rpo*/*qrpo*).

	Solver	*lpo*	*qlpo*	*lpos*	*qlpos*	*mpo*	*qmpo*	*rpo*	*qrpo*
1	SAT-based	**123** (0)	**127** (0)	**141** (0)	**155** (0)	**92** (0)	**98** (0)	**146** (0)	**162** (0)
	(direct)	**31.0**	**44.7**	**26.1**	**40.6**	**49.4**	**74.2**	**50.0**	**85.3**
2	dedicated	**123** (5)	**127**(16)	**141** (6)	**154**(45)	**92** (7)	**98**(31)	**145**(10)	**158** (65)
	(direct)	**334.4**	**1426.3**	**460.4**	**3291.7**	**653.2**	**2669.1**	**908.6**	**4708.2**
3	SAT-based	**357** (0)	**389** (0)	**362** (0)	**395** (2)	**369** (0)	**408** (1)	**375** (0)	**416** (2)
	(arg. filt.)	**79.3**	**199.6**	**69.0**	**261.1**	**110.9**	**267.8**	**108.8**	**331.4**
4	dedicated	**350**(55)	**374**(79)	**355**(57)	**380**(92)	**359**(69)	**391**(82)	**364**(74)	**394**(102)
	(arg. filt.)	**4039.6**	**5469.4**	**4522.8**	**6476.5**	**5169.7**	**5839.5**	**5536.6**	**7186.1**

Table 1. SAT-based vs. dedicated solvers for (subclasses of) RPO

The above table shows an orders of magnitude improvement over existing dedicated solvers both for direct analysis with recursive path orders and for the combination of recursive path orders and argument filters in the dependency pair framework. Note that without a time limit, this effect would only be aggravated.

A similar situation can be seen in the following table for POLO. Here, we evaluate our new SAT-based implementation (AProVE-SAT) against the non-SAT-based implementations in the termination tools AProVE 1.2 and TTT [10]. The implementation in AProVE 1.2 solves Diophantine constraints by a specialized finite domain constraint satisfaction procedure [5], while TTT uses a "generate-and-test" approach instead.[5]

The columns show data for POLO with different finite domains and different degrees of the polynomial which are specified by a pair (n, d). Here, if the the first component is n, then we only searched for coefficients from $\{0, ..., n\}$. If the second component is "*lin*", then we used linear polynomials, and if it is "*sm*", we used simple-mixed[6] polynomials (which are not available in TTT).

	Solver	$(1, lin)$	$(2, lin)$	$(3, lin)$	$(3, sm)$
1	AProVE-SAT	**421** (0)	**431** (0)	**434** (0)	**440** (51)
		45.5	**91.8**	**118.6**	**5585.9**
2	AProVE 1.2	**421** (1)	**414** (48)	**408** (81)	**404** (171)
		151.8	**3633.2**	**5793.2**	**11608.1**
3	TTT	**326** (32)	**335** (83)	**338** (110)	n/a
		2568.5	**5677.6**	**7426.9**	n/a

Table 2. SAT-based vs. dedicated solvers for POLO

The comparison of the SAT-based configuration AProVE-SAT with the non-SAT-based configurations shows that the provers based on SAT solving with our proposed encoding are faster by orders of magnitude.

[5] As AProVE and TTT use slightly different techniques for estimating dependency graphs and usable rules, their performance is not directly comparable. The experiments are meant to show that there is a big difference between the SAT-based implementation on the one side and all other implementations on the other side.

[6] A non-unary polynomial (with $n > 1$ in (4)) is *simple-mixed* if we have $e_{i,j} \leq 1$ for all its exponents. A unary polynomial is simple-mixed if it has the form $a + b\, x_1 + c\, x_1^2$.

This holds in particular if one considers a higher time limit or polynomials with higher coefficients or degrees (which are needed to increase the number of examples that can be proved, i.e., the power of automated termination proving). Note that for linear polynomials, there are no time-outs in the configuration AProVE-SAT, whereas the non-SAT-based configurations have many time-outs. Due to the increased efficiency, the number of examples where termination can be proved within the time limit is considerably higher in the SAT-based configuration.

7 Conclusion

The SAT-based implementations of RPO and POLO were used by AProVE in the *International Competition of Termination Tools 2006*. Here, AProVE was configured to use several other termination techniques in addition to RPO and POLO. Due to the speed of our new SAT-based approach, AProVE could try polynomial interpretations (also with higher ranges) as one of the first termination techniques. In case of failure, there was still enough time to try other termination techniques afterwards. With a time limit of 60 s for each example, AProVE could prove termination of 633 TRSs and thereby it was the winner of the competition for termination of TRSs. Similarly, AProVE also won the corresponding competition in 2007.

To summarize, automated termination analysis is a field where SAT solving has turned out to be extremely useful. At the same time, this field also poses new challenges for SAT solving, since for higher ranges and higher degrees of the polynomials, one sometimes obtains SAT problems which are hard for current SAT solvers.[7]

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. M. Codish, V. Lagoon, and P. J. Stuckey. Solving partial order constraints for LPO termination. In *Proc. RTA '06*, LNCS 4098, pages 4–18, 2006.
3. M. Codish, P. Schneider-Kamp, V. Lagoon, R. Thiemann, and J. Giesl. SAT solving for argument filterings. In *Proc. LPAR '06*, LNAI 4246, pages 30–44, 2006.
4. H. Comon and R. Treinen. Ordering constraints on trees. In *Proc. CAAP '94*, LNCS 787, pp. 1–14, 1994.
5. E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325–363, 2005.
6. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT Solving for Termination Analysis with Polynomial Interpretations. In *Proc. SAT '07*, LNCS 4501, pages 340–354, 2007.

[7] We have therefore submitted some of these problems to the SAT competition 2007.

7. J. Giesl, P. Schneider-Kamp, and R. Thiemann AProVE 1.2: Automatic Termination Proofs in the Dependency Pair Framework. In *Proc. IJCAR '06*, LNAI 4130, pages 281–286, 2006.
8. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs. In *Proc. LPAR'04*, LNAI 3452, pages 301–331, 2005.
9. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005.
10. N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.
11. H. Hong and D. Jakuš. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21(1):23–38, 1998.
12. M. Kurihara and H. Kondo. Efficient BDD encodings for partial order constraints with application to expert systems in software verification. In *Proc. IEA/AIE '04*, LNCS 3029, pages 827–837, 2004.
13. D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.
14. P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated Termination Analysis for Logic Programs by Term Rewriting, In *Proc. LOPSTR '06*, LNCS 4407, pages 177–193, 2007.
15. P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving Termination using Recursive Path Orders and SAT solving. In *Proc. FroCoS '07*, LNAI 4720, pages 267–282, 2007.
16. H. Zankl and A. Middeldorp. Satisfying KBO Constraints. In *Proc. RTA '07*, LNCS 4533, pages 389–403, 2007.
17. The termination problem data base. http://www.lri.fr/~marche/tpdb/.