Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 206 (2008) 75–90
www.elsevier.com/locate/entcs

Comparing CSP and SAT Solvers for Polynomial Constraints in Termination Provers

Salvador Lucas and Rafael Navarro-Marset [1,2]

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Valencia, Spain

Abstract

Proofs of termination in term rewriting involve solving constraints between terms coming from (parts of) the rules of the term rewriting system. A common way to deal with such constraints in termination tools is treating them as *polynomial constraints*. Several recent works develop connections between these problems and more standard constraint solving problems for which well-known and efficient techniques apply. In particular, SAT techniques are receiving increasing attention in the field. The main idea is encoding polynomial constraints as propositional constraints which can (hopefully) be efficiently managed by using state-of-the-art SAT solvers. We have recently developed an algorithm for solving constraints in finite (small) domains of coefficients which are appropriate for termination tools. This algorithm benefits from the use of a specialized representation of the elements in the domain and the corresponding polynomials which permits using efficient arithmetics and constraint propagation techniques. In this paper we discuss these approaches, compare them from an experimental point of view, and point to possible improvements.

Keywords: Polynomial interpretations, term rewriting, program analysis, termination.

1 Introduction

Proofs of termination in term rewriting involve solving constraints between terms s and t coming from (parts of) the rules of the Term Rewriting System (TRS [18,20]). For instance, in proofs of termination using the dependency pairs approach [2], given a rewrite rule $l \to r$ of a TRS $\mathcal{R}$, we get dependency pairs $l^\sharp \to s^\sharp$ for all subterms s of r which are rooted by a defined symbol [3]; the notation $t^\sharp$ for a given term t means that the root symbol f of t is *marked* thus becoming $f^\sharp$ (often just capitalized: F).

[1] This work has been partially supported by the EU (FEDER) and the Spanish MEC, under grants TIN 2004-7943-C04-02 and HA 2006-0007, and the Generalitat Valenciana under grant GV06/285. Rafael Navarro-Marset was partially supported by the Spanish MEC under FPU grant AP2006-026.

[2] Email: {slucas,rnavarro}@dsic.upv.es

[3] A symbol f is said to be *defined* in a TRS $\mathcal{R}$ if $\mathcal{R}$ contains a rule $f(l_1, \ldots, l_k) \to r$.

doi:10.1016/j.entcs.2008.03.076

Example 1.1 Consider the following TRS $\mathcal{R}$ from [2, Example 1]:

```
[1] minus(x,0) -> x
[2] minus(s(x),s(y)) -> minus(x,y)
[3] quot(0,s(y)) -> 0
[4] quot(s(x),s(y)) -> s(quot(minus(x,y),s(y)))
```

This TRS contains three dependency pairs:

```
[5] MINUS(s(x),s(y)) -> MINUS(x,y)
[6] QUOT(s(x),s(y)) -> QUOT(minus(x,y),s(y))
[7] QUOT(s(x),s(y)) -> MINUS(x,y)
```

The dependency pairs conform a new TRS $\mathsf{DP}(\mathcal{R})$ which (together with $\mathcal{R}$) determines the so-called *dependency chains*. The absence of infinite dependency chains characterize termination of $\mathcal{R}$. The dependency pairs can be presented as a *dependency graph* (DG), where the absence of infinite chains can be analyzed by considering the *cycles* in the graph.

Example 1.2 Consider the TRS $\mathcal{R}$ in Example 1.1. There are two cycles in the dependency graph: $\{5\}$ and $\{6\}$.

Basically, given a *cycle* in the dependency graph, we require $l \succeq r$ for all rules in the TRS $\mathcal{R}$, $u \succeq v$ for all dependency pairs in $\mathfrak{C}$ and $u > v$ for *at least one* dependency pair $u \to v \in \mathfrak{C}$. Here, $\succeq$ is a quasi-ordering on terms and $>$ is a *well-founded* ordering.

Example 1.3 Consider the TRS $\mathcal{R}$ in Example 1.1 and the cycle $\mathfrak{C} = \{6\}$ (see Example 1.2): `QUOT(s(x),s(y)) -> QUOT(minus(x,y),s(y))`. In order to prove termination of $\mathcal{R}$ we have to find a reduction pair $(\succeq, >)$ which satisfies the following constraints:

$$\begin{aligned}
\texttt{minus(x,0)} &\succeq \texttt{x}\\
\texttt{minus(s(x),s(y))} &\succeq \texttt{minus(x,y)}\\
\texttt{quot(0,s(y))} &\succeq \texttt{0}\\
\texttt{quot(s(x),s(y))} &\succeq \texttt{s(quot(minus(x,y),s(y)))}\\
\texttt{QUOT(s(x),s(y))} &> \texttt{QUOT(minus(x,y),s(y))}
\end{aligned}$$

Many termination tools (AProVE [7], C*i*ME 2.02 [3], MU-TERM [1,14], TTT [11],...) use polynomials as a principal ingredient to achieve termination proofs. In this setting, each k-ary symbol $f \in \mathcal{F}$ is given a *parametric* polynomial $[f]$ like, e.g., $a_k x_k + \cdots + a_1 x_1 + a_0$.

Example 1.4 Consider the constraints in Example 1.3. The following (parametric)

polynomials are given to the symbols:

$$\begin{aligned}
[\mathtt{0}] &= a_0 \\
[\mathtt{s}](X) &= s_1X + s_0 \\
[\mathtt{minus}](X,Y) &= m_1X + m_2Y + m_0 \\
[\mathtt{quot}](X,Y) &= q_1X + q_2Y + q_0 \\
[\mathtt{QUOT}](X,Y) &= q'_1X + q'_2Y + q'_0
\end{aligned}$$

Constraints $s \succeq t$ and $s > t$ are treated as *polynomial constraints* $P_{s,t} \geq 0$ and $P_{s,t} > 0$, respectively, where $P_{s,t} = [s] - [t]$ is the polynomial obtained from terms s and t by interpreting them as polynomials $[s]$ and $[t]$, see [4,15] for further details.

Example 1.5 The first constraint in Example 1.3 is translated into the polynomial constraint:

$$(m_1 - 1)x + m_2a_0 + m_0 \geq 0$$

Variables in terms s and t (e.g., $\mathtt{x}$ in the first constraint in Example 1.3) become universally quantified numeric variables in polynomial constraints $P_{s,t} \geq 0$ (e.g., x in Example 1.5). In contrast, the parametric coefficients become *existentially quantified variables* (e.g., a_0, m_0, m_1, and m_2 in Example 1.5). The use of non-negative numbers as interpretation domains and well-known positiveness criteria like Hong and Jakuš' [12] allows us to center the attention on *solving existential constraints* where all variables correspond to parametric coefficients.

Example 1.6 According to [4,15] and also [12], we have to solve the following (conjuntion of) polynomial constraints:

$$\begin{aligned}
&(1)\ a_0m_2 + m_0 \geq 0 \\
&(2)\ m_1 - 1 \geq 0 \\
&(3)\ m_1s_0 + m_2s_0 \geq 0 \\
&(4)\ m_1s_1 - m_1 \geq 0 \\
&(5)\ m_2s_1 - m_2 \geq 0 \\
&(6)\ a_0q_1 + q_2s_0 + q_0 - a_0 \geq 0 \\
&(7)\ q_2s_1 \geq 0
\end{aligned}$$

$$(8)\quad q_1 s_0 + q_2 s_0 + q_0 - m_0 q_1 s_1 - q_2 s_0 s_1 - q_0 s_1 - s_0 \geq 0$$

$$(9)\quad q_1 s_1 - m_1 q_1 s_1 \geq 0$$

$$(10)\quad q_2 s_1 - m_2 q_1 s_1 - q_2 s_1 s_1 \geq 0$$

$$(11)\quad q'_1 s_0 - m_0 q'_1 > 0$$

$$(12)\quad q'_1 s_1 - m_1 q'_1 \geq 0$$

$$(13)\quad -m_2 q'_1 \geq 0$$

Note that the *variables* which have to be *solved* here are the *coefficients* of the parametric polynomials in Example 1.4. The previous set of constraints is sound regarding DG-based termination proofs (i.e., its satisfaction implies that the cycle is harmless) provided that *all such variables/parametric coefficients take non-negative values*, see [4,15] for a justification of this claim.

Now, the termination problem is just a standard *constraint solving* problem which can be treated by using standard algorithms and techniques.

1.1 Solving polynomial constraints in termination provers

Constraints like those showed in Example 1.6 are expected to be solved on a suitable domain of *coefficients* because the intended meaning of the targetted variables is to serve as particular coefficients of parametric interpretations like in Example 1.4. In principle, such coefficients could be taken from any subset *real algebraic numbers* [4] [16]. Rational, integer, and natural numbers are well-known examples of real algebraic numbers. In practice, all termination tools restrict (as default or unique option) the usable coefficients to small domains like $\{0,1\}$, $\{0,1,2\}$, or $\{0,\frac{1}{2},1,2\}$. In [16] we have proposed an efficient algorithm for solving polynomial constraints over small domains of powers of 2. Note that the aforementioned small (and widely used) domains of coefficients are covered by the algorithm. The algorithm efficiently handles polynomial expressions involving numbers having quite different arithmetical treatment: 0, 1, 2, $\frac{1}{2}$, $\sqrt{2}$, $\frac{1}{\sqrt{2}}$, etc.

A recent paper by Fuhs et al. proposes the use of SAT techniques for solving polynomial constraints in termination provers [5]. Fuhs et al.'s (extensive) benchmarks show that, indeed, using $D = \{0,1\}$ as the domain for coefficients in polynomial interpretations is already a very powerful option in comparison to bigger domains. The information in Table 1 has been taken from [5]. It corresponds to the benchmarks performed with the new version of AProVE which implements a SAT-based solver for polynomial constraints (AProVE-SAT). The termination problems come from the 2006 Termination Problem Data Base (TPDB, version 3.2) [5]. 865 examples were considered. Three different ranges for coefficients were considered,

[4] A real number $x \in \mathbb{R}$ is said to be *algebraic* if it satisfies an equation $x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0$, of finite degree n where $a_i \in \mathbb{Q}$ for $0 \leq i \leq n-1$.

[5] see http://www.lri.fr/~marche/tpdb/

Coeff. Range:	1	2	3
# Success	421	431	434
% Success	49,1	49,8	50,2
Time	45.5 s.	91.8 s.	118.6 s.

Table 1
AProVE-SAT benchmarks

corresponding to $N_1 = \{0, 1\}$, $N_2 = \{0, 1, 2\}$, and $N_3 = \{0, 1, 2, 3\}$.

Table 1 shows that AProVE-SAT increments the ratio of solved examples in $0,7\%$ when using coefficients from N_2 instead of N_1, but the time for achieving the proofs is *duplicated!*

Thus, specifically considering the (small) domain N_1 to obtain an efficient solver on this particular domain still makes sense. We have addressed this task in different ways which we discuss in the remainder of the paper. Our research is motivated by the following questions:

(i) Should termination tools use existing translations of polynomial constraints into propositional formulas and then a state-of-the-art SAT solver? Which one?

(ii) Is it better to treat them as true polynomial constraints using CSP-like techniques?

(iii) How to select the appropriate technique or tool? How (where) to put them into the sequence of techniques of an 'expert'?

2 Solving polynomial constraints over small domains

We have developed an algorithm to solve existential polynomial constraints over *finite* subsets of appropriate real numbers [16]. The algorithm takes benefit from a suitable choice of domains for the coefficients and an appropriate representation of the polynomial constraints which permit both fast arithmetic and the use a number of techniques for safely avoiding a complete exploration of the search space.

2.1 Finite domains of powers of 2

The algorithm works for subsets D of rational numbers (actually powers of 2): $D \subseteq D_m = \{0\} \cup \{\pm(2^i) \mid i \in \mathbb{Z}, 0 \leq |i| \leq m\}$ for $m \in \mathbb{N}$ or square roots of powers of 2: $D \subseteq \overline{D}_m = \{0\} \cup \{\pm(2^{\frac{i}{2}}) \mid i \in \mathbb{Z}, 0 \leq |i| \leq 2m\}$. We write D_m^+ if we restrict the attention to non-negative numbers. In particular, $D_1^+ = \{0, 2^{-1}, 2^0, 2^1\} = \{0, \frac{1}{2}, 1, 2\}$ includes the non-negative coefficients used (by default) in all currently available termination tools (nowadays, MU-TERM is the only tool which supports the use of rational coefficients like $\frac{1}{2} = 2^{-1}$).

Restricting the attention to such kind of domains allows us reducing the costs of polynomial *arithmetics*, see [16] for details.

2.2 Polynomial constraints

We deal with constraints for polynomials $P \in \mathbb{Z}[X_1, \ldots, X_n]$, where $X_1, \ldots, X_n$ are variables ranging on D. We actually deal with P under the form $P = (V, M, N, K)$ for sets of variables V and polynomials $M, N \in \mathbb{N}[X_1, \ldots, X_n]$ and $K \in \mathbb{Z}$, i.e., we represent P by considering the variables V appearing in P, the positive monomials $\mathsf{m} \in M$ on one side, the negative monomials $\mathsf{n} \in N$, and the constant coefficient K (which can be positive, negative or null): $P = M - N + K$. Let Pol be the set of polynomials as above.

Constraints are sets $C \subseteq Pol \times \{weak, strict\}$ of pairs $(P, cond)$ where $cond$ indicates how the (basic) constraint $c = (P, cond)$ compares P to 0: in a *weak* ($P \geq 0$) or *strict* ($P > 0$) way.

Example 2.1 The constraint (1) in Example 1.6 is represented as follows:

$$((\{a_0, m_2, m_0\}, \{a_0 m_2, m_0\}, \varnothing, 0), weak)$$

Let $Var(P) \subseteq \{X_1, \ldots, X_n\}$ be the set of variables occurring in P and $Var(C) = \bigcup_{(P,cond) \in C} Var(P)$ be the set of variables in C. A solution σ of C is a mapping $\sigma : Var(C) \to D$ such that $(\sigma(P), cond)$ holds for all $(P, cond) \in C$, i.e., $P(\sigma(X_1), \ldots, \sigma(X_m)) \geq 0$ (resp. $P(\sigma(X_1), \ldots, \sigma(X_m)) > 0$) if $cond = weak$ (resp. $cond = strict$) and $Var(P) = \{X_1, \ldots, X_m\}$.

2.3 Constraint propagation

The constraint solving algorithm makes extensive use of *partial evaluation* of polynomials P w.r.t. one of its variables for doing *constraint propagation*. For instance, given $P(X_1, X_2, \ldots, X_n)$ and $d \in D$, we could need to obtain $P_{1,d}(X_2, \ldots, X_n) = P(d, X_2, \ldots, X_n)$. This involves the partial evaluation of *each* monomial $\mathsf{m} = cX_1^{\alpha_1} \cdots X_n^{\alpha_n}$ in P and the reconfiguration of the obtained polynomial as a tuple (V, M, N, K).

An important aspect of the algorithm is performing frequent *partial checkings* of the constraints in order to cut the search space. This means that we are often able to conclude the truth or falsity of a basic constraint $c = (P, cond)$ *with variables* without instantiating any variable in P. A (three valued) predicate *checkCS* performs this task. *checkCS*(c) returns either *true* if c is definitely true, or *false* if c is definitely false, or *??* otherwise. According to the representation $P = (V, M, N, K)$, and since we use domains D of *non-negative* numbers, we have the following cases (here expressed in logical form for saving space):

(i) $M \equiv 0 \wedge K < 0 \Rightarrow P \not\geq 0 \wedge P \not> 0$.

(ii) $N \equiv 0 \wedge K > 0 \Rightarrow P \geq 0 \wedge P > 0$.

(iii) $N \equiv 0 \wedge K = 0 \Rightarrow P \geq 0$.

(iv) $M \equiv 0 \wedge K = 0 \Rightarrow P \not> 0$.

here $M \equiv 0$ means that M is identically null.

Example 2.2 By using rule iii above we can remove (or definitely replace by True) constraints (1), (3), and (7) in Example 1.6.

Let β be the maximum element of D. Then, for all $x_1, \ldots, x_n \in D$,

- $K - N(\beta, \ldots, \beta) \leq P(x_1, \ldots, x_n)$, and
- $P(x_1, \ldots, x_n) \leq M(\beta, \ldots, \beta) + K$

This leads to the following:

(i) $M(\beta, \ldots, \beta) + K < 0 \Rightarrow P \not\geq 0 \wedge P \not> 0$.

(ii) $K - N(\beta, \ldots, \beta) > 0 \Rightarrow P \geq 0 \wedge P > 0$.

(iii) $K - N(\beta, \ldots, \beta) \geq 0 \Rightarrow P \geq 0$.

(iv) $M(\beta, \ldots, \beta) + K = 0 \Rightarrow P \not> 0$.

In particular, if $\beta = 1 = 2^0$, then we can easily compute $M(\beta, \ldots, \beta) + K$ and $K - N(\beta, \ldots, \beta) + K$ by just adding the *coefficients* of the corresponding monomials, and then adding (or substracting from) the constant K.

2.4 The algorithm

We describe our algorithm by means of two mutually recursive functions *solveCS* and *solveCSvar*. The initial call is $solveCS(D, [], [], C)$.

(i) $solveCS(D, V, pSol, C)$ performs an initial checking of all basic constraints $(P, cond)$ in the constraint C by using *checkCS*. If all constraints are true, then a singleton containing a pair $(V, pSol)$ consisting of the list of previously visited variables V and the list $pSol$ of partial solutions for these variables is returned. A partial solution is just a list $d_1, \ldots, d_k$ of values which correspond to the current list of visited variables $x_1, \ldots, x_k$, i.e., $x_i \mapsto d_i$ will be a binding of the final solution of the constraint. When the final solution is returned, variables x which were not instantiated receive a binding $x \mapsto d$ for an arbitrary $d \in D$ (typically $x \mapsto 0$).

(ii) $solveCSvar(D, V, pSol, C)$ tries values $d \in D$ on a variable x_i occurring in a constraint $c = (P, cond)$ in C. The instantiation of x_i with a value d yields a new constraint $c_{i,d} = (P_{i,d}, cond)$ consisting of the partial evaluation $P_{i,d}$ of P with d on the variable x_i and the same condition *cond*. The constraint $c_{i,d}$ is checked by using *checkCS* and if the inconsistency of $c_{i,d}$ is shown, then d is discarded as a possible value for solving c on x_i. Otherwise, the variable x_i is recorded as 'visited' and the value d which permits to make progress is registered in the list of tuples which are partial solutions. Also, each constraint in $C - \{c\}$ is partially evaluated w.r.t. x_i and d as above and a new problem $C_{i,d}$ is raised. If $c_{i,d}$ is found true, then the constraint solving process continues with $C_{i,d}$. If nothing can be said about $c_{i,d}$, then the constraint solving process continues with $\{c_{i,d}\} \cup C_{i,d}$.

The complete description of the two functions is in Figure 1.

$\mathbf{solveCSvar}(D, V, pSol, \{c\} \cup C)$

$\quad \mathbf{if}\ CD_{Pol} = CD_{fail}\ \mathbf{then}\ \varnothing$

$\quad \mathbf{else}\ \bigcup_{(c,d)\in CD_{unknown}} \mathbf{solveCS}(D, x_i : V, d : pSol, \{c\} \cup C(d))$
$\qquad \cup \bigcup_{(c,d)\in CD_{true}} \mathbf{solveCS}(D, x_i : V, d : pSol, C(d))$

$\mathbf{where}$

$(P, cond) = c$

$(X \cup \{x_i\}, \_, \_, \_) = P$

$CD_{Pol} = \{((pEval(P, i, d), cond), d) \mid d \in D\}$

$CD_{fail} = \{(c, \_) \in CD_{Pol} \mid checkCS(c) = fail\}$

$CD_{true} = \{(c, \_) \in CD_{Pol} \mid checkCS(c) = true\}$

$CD_{unknown} = \{(c, d) \in CD_{Pol} \mid checkCS(c) = ??\}$

$C(d) = \{pEvalCS(c, x_i, d) \mid c \in C\}$

$\mathbf{solveCS}(D, V, pSol, C)$

$\quad \mathbf{if}\ C_{fail} \neq \varnothing\ \mathbf{then}\ \varnothing$

$\quad \mathbf{else\ if}\ C_{noTrue} = \varnothing\ \mathbf{then}\ \{(V, pSol)\}$

$\quad\quad \mathbf{else\ solveCSvar}(D, V, pSol, C')$

$\mathbf{where}$

$C_{noTrue} = \{c \in C \mid checkCS(c) \neq true\}$

$C_{fail} = \{c \in C_{noTrue} \mid checkCS(c) = fail\}$

$C' = \{c \in C_{noTrue} \mid checkCS(c) = ??\}$

Fig. 1. Constraint solving algorithm

3 Solving constraints over N_1

In this section we investigate how to improve the previous algorithm to obtain better performance when a domain $N_1 = \{0, 1\}$ considered.

$$
\begin{aligned}
\tau(K \geq 0) &= True, \quad \text{if } K \geq 0 \\
\tau(K \geq 0) &= False, \quad \text{if } K < 0 \\
\tau(K > 0) &= True, \quad \text{if } K > 0 \\
\tau(K > 0) &= False, \quad \text{if } K \leq 0 \\
\tau(cX_1 \ldots X_n + Q \geq 0) &= \Big(\Big(\bigvee_{1 \leq i \leq n} \neg X_i\Big) \wedge \tau(\mathsf{rmM}_{X_1,\ldots,X_n}(Q) \geq 0)\Big) \vee \\
&\quad \Big(\Big(\bigwedge_{1 \leq i \leq n} X_i\Big) \wedge \tau(\mathsf{rmV}_{X_1,\ldots,X_n}(Q) + c \geq 0)\Big) \\
\tau(cX_1 \ldots X_n + Q > 0) &= \Big(\Big(\bigvee_{1 \leq i \leq n} \neg X_i\Big) \wedge \tau(\mathsf{rmM}_{X_1,\ldots,X_n}(Q) > 0)\Big) \vee \\
&\quad \Big(\Big(\bigwedge_{1 \leq i \leq n} X_i\Big) \wedge \tau(\mathsf{rmV}_{X_1,\ldots,X_n}(Q) + c > 0)\Big) \\
\tau(C \wedge C') &= \tau(C) \wedge \tau(C')
\end{aligned}
$$

Fig. 2. SAT encoding of polynomial constraints over N_1

3.1 Simplifying the polynomial representation

Since variables in the considered polynomials range on N_1 and for all $x \in N_1$ and all $n > 0$ we have $x^n = x$, when considering the representation of a polynomial P, we can *replace* monomials $\mathsf{m} = cX_1^{\alpha_1} \cdots X_n^{\alpha_n}$ in P by $\mathsf{m}' = cX_1^{\beta_1} \cdots X_n^{\beta_n}$ where $\beta_i = 1$ if $\alpha_i \neq 0$ and $\beta_i = 0$ if $\alpha_i = 0$. Then, we add all coefficients of monomials of the same degree $\beta_1, \ldots, \beta_n$ to obtain a single one and proceed like that to obtain a simpler representation P' of P.

Example 3.1 The polynomial constraint (10) in Example 1.6 would be transformed into

$$(10') \quad -m_2 q_1 s_1 \geq 0$$

3.2 SAT-solving for constraints over N_1

When considering polynomial constraints over N_1, the arithmetics on N_1 become very close to boolean operations when 0 is interpreted as $False$ and 1 as $True$, respectively. In particular, the product of values in N_1 correspond to conjunction. Following this intuition, we have developed a simple encoding of polynomial constraints as propositional formulas.

The translation function τ is given in Figure 2, where Q is a polynomial, c and K are numeric constants (with $c \neq 0$), $X_1, \ldots, X_n$ are variables (ranging on N_1), $\mathsf{rmM}_{X_1,\ldots,X_n}(P)$ removes all monomials in P which include *all* variables $X_1, \ldots, X_n$, and $\mathsf{rmV}_{X_1,\ldots,X_n}(P)$ removes from P all occurrences of variables in $X_1, \ldots, X_n$. According to the discussion in Section 3.1, we also assume that we only have to deal with polynomials consisting of monomials like $cX_1 \cdots X_n$ (i.e., *without* any power greater than 1).

Example 3.2 Consider the constraint (9) in Example 1.6. It is translated into a

propositional formula as follows:

$$\begin{aligned}&\tau(q_1 s_1 - m_1 q_1 s_1 \geq 0)\\ &= ((\neg q_1 \vee \neg s_1) \wedge \tau(0 \geq 0)) \vee ((q_1 \wedge s_1) \wedge \tau(-m_1 + 1 \geq 0))\\ &= ((\neg q_1 \vee \neg s_1) \wedge True) \vee ((q_1 \wedge s_1) \wedge \tau(-m_1 + 1 \geq 0))\end{aligned}$$

Since we have:

$$\begin{aligned}\tau(-m_1 + 1 \geq 0) &= (\neg m_1 \wedge \tau(1 \geq 0)) \vee (m_1 \wedge \tau(0 \geq 0))\\ &= (\neg m_1 \wedge True) \vee (m_1 \wedge True)\\ &\Leftrightarrow \neg m_1 \vee m_1\\ &\Leftrightarrow True\end{aligned}$$

we conclude:

$$\begin{aligned}&\tau(q_1 s_1 - m_1 q_1 s_1 \geq 0)\\ &= ((\neg q_1 \vee \neg s_1) \wedge True) \vee ((q_1 \wedge s_1) \wedge ((\neg m_1 \wedge True) \vee (m_1 \wedge True)))\\ &\Leftrightarrow (\neg q_1 \vee \neg s_1) \vee (q_1 \wedge s_1)\\ &\Leftrightarrow (\neg q_1 \vee \neg s_1) \vee \neg (\neg q_1 \vee \neg s_1)\\ &\Leftrightarrow True\end{aligned}$$

In order to obtain a propositional formula in CNF format, we call an external module implementing the algorithm in [19].

3.3 Benchmarks for N_1

We have compared the behavior of MU-TERM when different polynomial constraint solving engines are used to prove termination of programs and the domain of coefficients is N_1:

(i) MU-TERM-SD uses the constraint solving algorithm in Section 2 together with the improvements described in Section 3.1.

(ii) MU-TERM-SAT uses the translation of polynomial constraints into propositional formulas described in Section 3.2 and then uses MiniSat [6] to obtain a solution.

(iii) MU-TERM-ApSAT uses an external module implementing the SAT-based constraint solving algorithm described in [5] and implemented as part of AProVE, and which also uses MiniSat for solving the generated propositional constraints.

(iv) MU-TERM-CiME uses C*i*ME as an external module implementing the constraint solving algorithm described in [4].

[6] http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/MiniSat.html

	SD	SAT	ApSAT	CiME
# YES	305	306	306	305
# ??	596	610	599	612
# TOs	51	36	47	35
YES Av.T.	0.50	0.66	2.52	0.36
?? Av.T.	1,25	1,83	5,41	1,64

Table 2
Different solvers for N_1

Tool:	MU-TERM-SD			MU-TERM-CiME			MU-TERM-SAT			MU-TERM-ApSAT		
TO	YES	??	TO	YES	??	TO	YES	??	TO	YES	??	TO
1 s.	289	511	152	289	521	142	291	523	138	210	143	599
10 s.	301	578	73	302	585	65	302	578	72	288	517	147
30 s.	303	592	57	303	602	47	303	599	50	297	574	81
60 s.	305	596	51	304	613	35	306	610	36	306	599	47

Table 3
Different time-outs for N_1

We have considered the 952 examples in the 'Standard' TRS subcategory of the 2007 Termination Competition [7] which are part of the 2007 Termination Problem Data Base (TPDB, version 4.0) [8]. The tools were executed under OS Linux Ubuntu 4.1.1-13ubuntu5, on a Intel Core 2 CPU at 2.13 GHz and 1 GByte of primary memory. Complete information about all benchmarks in the paper can be found here:

http://www.dsic.upv.es/~rnavarro/prole07/benchmarks

Table 2 summarizes the proofs obtained by the different versions of MU-TERM. Row '# YES' indicates the number of successful proofs; row '# ??' indicates the number of unsuccessful proofs; and row '# TOs' indicates the number of unfinished proofs interrupted by the time-out of 60 seconds. Rows 'YES Av. T.'/ '?? Av. T.' indicate the average time of successful/unsuccessful proofs (in seconds).

Remark 3.3 Note that, although MU-TERM-SAT directly implements the encoding described in Section 3.2, it still performs *two* calls to external tools (the CNF converter and MiniSat). Similarly, MU-TERM-ApSAT actually performs two external calls (one to AProVE's SAT-solving engine which then calls to MiniSat). In this sense, we believe that comparing our SAT-encoding and Fuhs et al.'s one through MU-TERM-SAT and MU-TERM-ApSAT is fair in our experimental setting.

3.3.1 Different time-outs.

Tools for proving termination do not use a *single* technique for proving termination. Termination provers rather proceed stepwise by following some particular sequence

[7] http://www.lri.fr/~marche/termination-competition/2007
[8] http://www.lri.fr/~marche/tpdb

	SD	ApSAT	CiME
# YES	299	313	287
# ??	472	583	563
# TOs	181	56	102
YES Av.T.	0.85	2.18	1.20
?? Av.T.	3.04	5.05	2.15

Table 4
Different solvers for N_2

Tool:	MU-TERM-SD			MU-TERM-CiME			MU-TERM-ApSAT		
TO	YES	??	TO	YES	??	TO	YES	??	TO
1 s.	280	356	316	263	454	235	219	168	565
10 s.	292	433	227	280	530	142	295	517	140
30 s.	295	456	201	282	551	119	308	552	92
60 s.	299	472	181	287	563	102	313	583	56

Table 5
Different time-outs for N_2

of several techniques which are given 'partial' time-outs which are a (small) fraction of the global time-out.

Remark 3.4 Nowadays, the termination expert implemented in MU-TERM performs the proofs according to a sequence of 10 different techniques among which we try different kinds of polynomial interpretations and different bounds for the coefficients. The global time-out is equitatively distributed among the different techniques. Hence, a global time-out of 60 s. amounts at each technique to have at most six seconds to obtain a proof.

Thus, we have also considered the behavior of the four solvers when different time-outs (below 60 seconds) are considered. Table 3 shows our results for N_1.

4 Solving constraints over bigger domains

In this section we report on the performance of the constraint solving methods when bigger domains are used. First, $N_2 = \{0, 1, 2\}$ is considered for solving the polynomial constraints. We have used the same collection of examples, but MU-TERM-SAT is not considered anymore for obvious reasons. Table 4 summarize our results for N_2. Let's briefly consider the performance of the constraint solving methods when $N_5 = \{0, 1, 2, 3, 4, 5\}$ is considered for solving the polynomial constraints. Since N_5 cannot be expressed as a subset of powers of 2, we cannot properly use MU-TERM-

	GSD	ApSAT	CiME
# YES	283	311	264
# ??	297	550	477
# TOs	372	91	211
YES Av.T.	1.00	2.88	1.03
?? Av.T.	4.49	7.06	2.38

Table 6
Different solvers for N_5

Tool:	MU-TERM-GSD			MU-TERM-CiME			MU-TERM-ApSAT		
TO	YES	??	TO	YES	??	TO	YES	??	TO
1 s.	269	200	483	248	376	328	200	69	683
10 s.	277	259	416	257	443	252	290	448	214
30 s.	280	281	391	259	468	225	303	524	125
60 s.	283	297	372	264	477	211	311	550	91

Table 7
Different time-outs for N_5

SD. However, it is very easy to use the algorithm in Section 2.4 together with a generalized arithmetical treatment of the numeric domains by just using the standard arithmetic operations (addition, product, power) instead of relying on binary shiftings as in [16]. This easily leads to a generalization of the original algorithm. We call MU-TERM-GSD the new version of MU-TERM which implements such a generalized version of the algorithm described in Section 2. Table 6 summarize our results for N_5.

Remark 4.1 Note that MU-TERM-(G)SD directly implements the algorithm described in Section 2 (without any external call) whereas MU-TERM-ApSAT still performs two external calls, and MU-TERM-CiME performs one external call (to C*i*ME). This has to be taken into account to provide a fair interpretation of the benchmarks.

5 Analysis of benchmarks

In order to answer the questions posed at the end of Section 1.1, we need to classify the existing choices according to their suitability. On the basis of our experience in the development of tools for proving termination, we believe that the following concrete criteria are appropriate to make this selection:

(i) Take the more successful technique, i.e., having the bigger '# YES'. This seems to require few justification.

(ii) Among equally successful techniques, take the ones which are *complete* regarding the implemented technique, i.e., '??' answers actually mean that the considered technique does *not* work on the considered problem [9].

(iii) Among complete techniques, take the ones having the bigger '# ??'. This permits switching to a different technique more often.

(iv) Among techniques having the same '# ??', take the ones having lesser average time for ?? answers. This permits a fast switching to a different technique, thus saving time from the assigned time slot.

According to this, we conclude the following.

(i) The results in Tables 2 and 3 show that our encoding of polynomial constraints over N_1 as propositional formulas and the use of state-of-the-art SAT solvers (e.g., MiniSat) seems to be the best way to deal with such kind of constraints when N_1 is the domain of coefficients.

(ii) Benchmarks in Table 2 also show that our SAT-encoding of polynomial constraints over N_1 is better (in practice) than [5] when used with N_1: although both of them succeed on the same number of examples, MU-TERM-SAT has a bigger '# ??'. Furthermore, MU-TERM-SAT is $\frac{2.52}{0.66} = 3.8$ times *faster* than MU-TERM-ApSAT in giving a *positive* answer and $\frac{5.41}{1.83} = 3.0$ times *faster* in giving a *negative* answer. According to Table 3, the differences are even more important when small time-outs are used.

(iii) Benchmarks in Tables 4 and 6 show that MU-TERM-ApSAT exhibits the best behavior over N_2 and N_5. Furthermore, since the running conditions of MU-TERM-(G)SD, MU-TERM-ApSAT, and MU-TERM-CiME are quite different (see Remark 4.1), the detailed analysis of Tables 5 and 7 shows that, indeed, the *external* use of the SAT-based constraint solving algorithm reported in [5] is *better* than the direct use of the algorithms in [4,16] for domains of natural numbers like N_2 or N_5. Actually, regarding [4] our benchmarks provide an independent confirmation of a similar claim in [5].

(iv) Finally, we note that, although $N_i \subset N_j$ whenever $i < j$, the number of successful proofs with N_j is *not* bigger than with N_i (in general). This is due to dealing with a bigger search space in the presence of time-outs. Actually, except for MU-TERM-ApSAT in the transition from N_1 to N_2, in all cases the use of the same solver leads to *loosing* successful proofs when the upper bound for coefficients increases. Furthermore, since there are examples *requiring* the use of N_j instead of N_i, the number of 'lost' proofs when moving from N_i to N_j for some $i < j$ is actually bigger than suggested by our numbers. This means that first considering the smallest domains is better than a direct attempt on a bigger but 'heavier' domain of coefficients.

[9] All tools considered here, except MU-TERM-CiME, are complete in this sense.

Summarizing, we can say that, among the considered techniques, our new SAT encoding for constrains over N_1 is the choice for constraint solving over N_1; otherwise the SAT-based solver in [5] should be used when domains of natural numbers are considered (even as an external tool). Finally, the algorithm in [16] is appropriate for constraint solving over domains of *rational* coefficients.

An automatic 'termination expert' implementing the corresponding techniques should combine them accordingly starting with N_1, then N_2, etc.

6 Conclusions

We have developed an efficient encoding of polynomial constraints over $N_1 = \{0, 1\}$ as propositional formulas which (in our benchmarks) is almost four times faster than the recent SAT-based algorithm by Fuhs et al. [5] when applied to solve polynomial constraints over N_1. We have also generalized the algorithm in [16] to deal with arbitrary non-negative numbers. We have investigated the use of different constraint-solving algorithms for the efficient generation of polynomial interpretations in termination provers. We have considered CSP-based solvers like the ones described in [4,16] and SAT-based solvers like the recent proposal in [5] and the new one introduced in this paper. We have implemented or connected (implementations of) the different algorithms as part of the tool MU-TERM.

The benchmarks for MU-TERM-SD (and even for MU-TERM-GSD) suggest that, in comparison to similar polynomial constraint solvers like the one reported in [4], it performs quite well. But the algorithm described in [16] can still benefit from some usual heuristics coming from the CSP area which have not been considered yet. Also, the SAT-encodings discussed here (both our new proposal in Section 3.2 and also [5]) do not take into account more sophisticated SAT frameworks like SMT (SAT modulo theories, see, e.g., [17]) which seem to be a natural choice for polynomial constraints. Furthermore, since the main goal of the algorithm in [16] is providing an efficient way to deal with polynomial constraints over *rational* (or even *real algebraic*) numbers, an interesting open problem is how to encode polynomial constraints over such more general domains using SAT/SMT techniques. These are interesting subjects for future work.

Acknowledgement

We thank Jürgen Giesl and Peter Schneider-Kamp for providing an executable version of the SAT-based polynomial constraint solving engine implemented in AProVE-SAT [5]. We also thank Jürgen Giesl, Carsten Fuhs, and Peter Schneider-Kamp for many valuable discussions regarding the topic of this paper.

References

[1] B. Alarcón, R. Gutiérrez, J. Iborra, and S. Lucas. Proving Termination of Context-Sensitive Rewriting with MU-TERM. *Electronic Notes in Theoretical Computer Science*, 188:105-115, 2007.

[2] T. Arts and J. Giesl. Termination of Term Rewriting Using Dependency Pairs. *Theoretical Computer Science*, 236:133-178, 2000.

[3] E. Contejean and C. Marché, B. Monate and X. Urbain. Proving termination of rewriting with C*i*ME. In A. Rubio, editor, *Proc. of 6th International Workshop on Termination, WST'03*, pages 71-73, Technical Report DSIC II/15/03, Valencia, Spain, 2003. Available at http://cime.lri.fr.

[4] E. Contejean, C. Marché, A.-P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. Journal of Automated Reasoning, 34(4):325-363, 2006.

[5] C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT Solving for Termination Analysis with Polynomial Interpretations. In J. Marques-Silva and K.A. Sakallah, editors, *Proc. of the 10th International Conference on Theory and Applications of Satisfiability Testing, SAT'07*, LNCS 4501:340-354, Springer-Verlag, Berlin, 2007.

[6] J. Giesl, T. Arts, and E. Ohlebusch Modular Termination Proofs for Rewriting Using Dependency Pairs. *Journal of Symbolic Computation* 34(1):21-58, 2002.

[7] J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic Termination Proofs in the Dependency Pair Framework. In U. Furbach and N. Shankar, editors, *Proc. of Third International Joint Conference on Automated Reasoning, IJCAR'06*, LNAI 4130:281-286, Springer-Verlag, Berlin, 2006. Available at http://aprove.informatik.rwth-aachen.de.

[8] J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated Termination Analysis for Haskell: From Term Rewriting to Programming Languages. In F. Pfenning, editor, *Proc of the 18th International Conference on Rewriting Techniques and Applications, RTA'06*, LNCS 4098:297-312, Springer Verlag, Berlin, 2006.

[9] J. Giesl, R. Thiemann, and P. Schneider-Kamp. The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs. In F. Baader and A. Voronkov, editors *Proc. of 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR'04*, LNCS 3452:301-331, Springer-Verlag, Berlin, 2004.

[10] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and Improving Dependency Pairs. Journal of Automated Reasoning, 37(3):155-203, Springer-Verlag, 2006.

[11] N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205:474-511, 2007.

[12] H. Hong and D. Jakuš. Testing Positiveness of Polynomials. *Journal of Automated Reasoning* 21:23-38, 1998.

[13] D.S. Lankford. On proving term rewriting systems are noetherian. Technical Report, Louisiana Technological University, Ruston, LA, 1979.

[14] S. Lucas. MU-TERM: A Tool for Proving Termination of Context-Sensitive Rewriting In V. van Oostrom, editor, *Proc. of 15h International Conference on Rewriting Techniques and Applications, RTA'04*, LNCS 3091:200-209, Springer-Verlag, Berlin, 2004. Available: http://zenon.dsic.upv.es/muterm.

[15] S. Lucas. Polynomials over the reals in proofs of termination: from theory to practice. *RAIRO Theoretical Informatics and Applications*, 39(3):547-586, 2005.

[16] S. Lucas. Practical use of polynomials over the reals in proofs of termination. In *Proc. of 9th International Symposium on Principles and Practice of Declarative Programming, PPDP'07*, pages 39-50, ACM Press, 2007.

[17] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Longemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937-977, 2006.

[18] E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer-Verlag, Berlin, 2002.

[19] D. Sheridan. The Optimality of a Fast CNF Conversion and its use with SAT. In *Proc. of 7th International Conference on Theory and Applications of Satisfiability Testing, SAT'04*, 2004.

[20] TeReSe, editor, *Term Rewriting Systems*, Cambridge University Press, 2003.