SAT Solving for Termination Analysis with Polynomial Interpretations*

Carsten Fuhs[1], Jürgen Giesl[1], Aart Middeldorp[2], Peter Schneider-Kamp[1], René Thiemann[1], and Harald Zankl[2]

[1] LuFG Informatik 2, RWTH Aachen, Germany,
{fuhs,giesl,psk,thiemann}@informatik.rwth-aachen.de
[2] Institute of Computer Science, University of Innsbruck, Austria,
{aart.middeldorp,harald.zankl}@uibk.ac.at

Abstract. Polynomial interpretations are one of the most popular techniques for automated termination analysis and the search for such interpretations is a main bottleneck in most termination provers. We show that one can obtain speedups in orders of magnitude by encoding this task as a SAT problem and by applying modern SAT solvers.

1 Introduction

Termination is one of the most important properties of programs and therefore, there is a need for techniques and tools that analyze the termination behavior of programs automatically. In particular, there has been intensive research on methods for termination analysis of *term rewrite systems* (TRSs) [4]. Instead of developing several separate termination techniques for different programming languages, a promising approach is to transform programs from different languages into TRSs instead. Then termination tools for TRSs can be used for termination analysis of many different programming languages, cf. e.g. [13, 22].

The increasing interest in termination analysis for TRSs is also shown by the annual *International Competition of Termination Tools*.[3] In 2006, for the first time some tools used SAT solvers to automate certain termination techniques, cf. [1, 5, 6, 11, 18, 25, 26]. But although *polynomial interpretations* [20] are one of the most popular techniques in these tools, up to now there has not been any paper on using SAT solvers for finding polynomial interpretations automatically.

In this paper, we show that SAT solving is extremely useful for this task. We recapitulate TRSs in Sect. 2. Sect. 3 shows how to encode the search for polynomial interpretations as a SAT problem. Sect. 4 extends our approach to *negative* polynomial interpretations [17]. Sect. 5 presents our implementation in the tool AProVE [14], which was the most powerful termination prover for TRSs in all the competitions 2004 - 2006. Our experiments show that our approach improves dramatically over previous methods for generating polynomial interpretations.

* Supported by the DFG (Deutsche Forschungsgemeinschaft) grant GI 274/5-1 and the FWF (Austrian Science Fund) project P18763.

[3] See http://www.lri.fr/~marche/termination-competition/

2 Termination of TRSs and Polynomial Interpretations

A TRS $\mathcal{R}$ is a set of rules $\ell \to r$ where ℓ and r are terms. A rule $\ell \to r$ applies to a term t if ℓ matches a subterm u of t with some substitution σ (namely, $u = \sigma(\ell)$). The rule is applied by replacing the subterm u by $\sigma(r)$, resulting in a new term v (a so-called *rewrite step*, denoted "$t \to_{\mathcal{R}} v$"). A *reduction* is a sequence of rewrite steps. A TRS is *terminating* if all its reductions are finite. For example, consider the following TRS where s represents the successor function, $\mathsf{half}(x)$ computes $\lfloor \frac{x}{2} \rfloor$, and $\mathsf{bits}(x)$ is the number of bits needed to represent all numbers up to x.

$$\begin{array}{rlrl} \mathsf{half}(\mathsf{0}) \to \mathsf{0} & \text{(i)} & \mathsf{bits}(\mathsf{0}) \to \mathsf{0} & \text{(iv)} \\ \mathsf{half}(\mathsf{s}(\mathsf{0})) \to \mathsf{0} & \text{(ii)} & \mathsf{bits}(\mathsf{s}(\mathsf{0})) \to \mathsf{s}(\mathsf{0}) & \text{(v)} \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{half}(x)) & \text{(iii)} & \mathsf{bits}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{bits}(\mathsf{s}(\mathsf{half}(x)))) & \text{(vi)} \end{array}$$

So we have $\mathsf{half}(\mathsf{s}(\mathsf{s}(\mathsf{0}))) \to_{\mathcal{R}} \mathsf{s}(\mathsf{half}(\mathsf{0})) \to_{\mathcal{R}} \mathsf{s}(\mathsf{0})$, i.e., $\mathsf{half}(\mathsf{s}(\mathsf{s}(\mathsf{0}))) \to_{\mathcal{R}}^* \mathsf{s}(\mathsf{0})$.

One of the most powerful termination methods is the *dependency pair* (DP) technique [2], implemented in virtually all current termination tools for TRSs.

Definition 1 (Dependency Pairs [2]). *For a TRS $\mathcal{R}$, the* defined *symbols are the root symbols of the left-hand sides of rules. For every defined symbol f, we extend the signature by a fresh* tuple symbol *$f^\sharp$ with the same arity as f. If $t = f(t_1, \ldots, t_n)$ and f is a defined symbol, we write $t^\sharp$ for $f^\sharp(t_1, \ldots, t_n)$. If $\ell \to r \in \mathcal{R}$ and t is a subterm of r with defined root symbol, then the rule $\ell^\sharp \to t^\sharp$ is a* dependency pair *of $\mathcal{R}$. The set of all dependency pairs of $\mathcal{R}$ is denoted $DP(\mathcal{R})$.*

In our example, half and bits are defined symbols and $DP(\mathcal{R}) = \{\text{(vii)}, \text{(viii)}, \text{(ix)}\}$:

$$\begin{array}{rlrl} \mathsf{half}^\sharp(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{half}^\sharp(x) & \text{(vii)} & & \\ \mathsf{bits}^\sharp(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{half}^\sharp(x) & \text{(viii)} & \mathsf{bits}^\sharp(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{bits}^\sharp(\mathsf{s}(\mathsf{half}(x))) & \text{(ix)} \end{array}$$

Intuitively, a DP corresponds to a (possibly recursive) function call. To prove termination, we have to show that there cannot be infinitely many function calls in any reduction. More precisely, one has to prove that there is no infinite chain

$$\sigma_1(u_1) \to_{DP(\mathcal{R})} \sigma_1(v_1) \ \to_{\mathcal{R}}^* \ \sigma_2(u_2) \to_{DP(\mathcal{R})} \sigma_2(v_2) \ \to_{\mathcal{R}}^* \ \sigma_3(u_3) \to_{DP(\mathcal{R})} \sigma_3(v_3) \ \ldots$$

where $u_i \to v_i \in DP(\mathcal{R})$ and σ_i are substitutions. To this end, the DP method[4] requires $u \succ v$ for all $u \to v \in DP(\mathcal{R})$ and $\ell \succsim r$ for all rules $\ell \to r \in \mathcal{R}$:

$$\bigwedge\nolimits_{u \to v \,\in\, DP(\mathcal{R})} \quad u \succ v \qquad \wedge \qquad \bigwedge\nolimits_{\ell \to r \,\in\, \mathcal{R}} \quad \ell \succsim r \tag{1}$$

A popular method to search for relations $\succ$ and $\succsim$ automatically are *polynomial interpretations* [20]. A polynomial interpretation $\mathcal{P}ol$ maps each n-ary function symbol f to a polynomial $f_{\mathcal{P}ol}$ over n variables $x_1, ..., x_n$ with coefficients from $\mathbb{N} = \{0, 1, 2, ...\}$. This mapping is extended to terms by defining $[x]_{\mathcal{P}ol} = x$ for all variables x and $[f(t_1, ..., t_n)]_{\mathcal{P}ol} = f_{\mathcal{P}ol}([t_1]_{\mathcal{P}ol}, ..., [t_n]_{\mathcal{P}ol})$. If the interpretation $\mathcal{P}ol$ is clear from the context, we also write $[t]$ instead of $[t]_{\mathcal{P}ol}$.

For example, consider $\mathcal{P}ol_1$ with $\mathsf{half}_{\mathcal{P}ol_1} = \mathsf{half}^\sharp_{\mathcal{P}ol_1} = x_1$, $\mathsf{bits}_{\mathcal{P}ol_1} = \mathsf{bits}^\sharp_{\mathcal{P}ol_1} =$

[4] For further refinements of the DP method we refer to [2, 12, 15–17], for example.

$\mathsf{s}_{\mathcal{P}ol_1} = x_1 + 1$, $\mathsf{0}_{\mathcal{P}ol_1} = 0$. Then $[\mathsf{half}(\mathsf{s}(\mathsf{s}(x)))] = x + 2$ and $[\mathsf{s}(\mathsf{half}(x))] = x + 1$. Now a term u is considered to be greater (resp. greater-equal) than v iff $[u] > [v]$ (resp. $[u] \geq [v]$) holds for all instantiations of the variables with natural numbers. So with $\mathcal{P}ol_1$ we obtain $\mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{s}(\mathsf{half}(x))$. In fact, all DPs (vii) - (ix) are strictly decreasing and the rules (i) - (vi) are at least weakly decreasing, i.e., the requirement (1) holds. Thus, termination of the TRS (i) - (vi) is proved.

To find such interpretations automatically, one starts with an *abstract* polynomial interpretation. It maps each n-ary symbol f to a polynomial of the form

$$a_0 + a_1\, x_1^{e_{11}} \dots x_n^{e_{n1}} \; + \; \dots \; + \; a_m\, x_1^{e_{1m}} \dots x_n^{e_{nm}} \tag{2}$$

Here, the e_{ij} are actual numbers (i.e., one has to determine the degree and the shape of the polynomials), but the coefficients a_i are left open (i.e., they are *variable* or *abstract* coefficients). For example, we could use the abstract polynomial interpretation $\mathcal{P}ol_2$ with $\mathsf{half}_{\mathcal{P}ol_2} = a\, x_1 + b$, $\mathsf{s}_{\mathcal{P}ol_2} = c\, x_1 + d$, etc.

Every inequality $u \succ v$ (resp. $u \succsim v$) can be transformed into the constraint $[u] - [v] > 0$ (resp. $[u] - [v] \geq 0$). Here, $[u] - [v]$ is a polynomial of the form

$$p_0 + p_1\, x_1^{e_{11}} \dots x_n^{e_{n1}} + \dots + p_k\, x_1^{e_{1k}} \dots x_n^{e_{nk}} \tag{3}$$

where p_i are polynomials over abstract coefficients. So with $\mathcal{P}ol_2$, $\mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{s}(\mathsf{half}(x))$ is transformed to $a\,c^2\,x + a\,c\,d + a\,d + b - c\,a\,x - c\,b - d \;>\; 0$, i.e. to

$$p_0 + p_1\, x \;>\; 0 \quad \text{where} \quad p_0 = a\,c\,d + a\,d + b - c\,b - d \quad \text{and} \quad p_1 = a\,c^2 - c\,a \tag{x}$$

If p is a polynomial like (3), then instead of inequalities or equalities of the form $p > 0$, $p \geq 0$, $p = 0$, it suffices[5] to require the following constraints [19]:

$$\alpha_{p>0} \;=\; (p_0 > 0 \wedge p_1 \geq 0 \wedge \ldots \wedge p_k \geq 0) \tag{4}$$
$$\alpha_{p\geq 0} \;=\; (p_0 \geq 0 \wedge p_1 \geq 0 \wedge \ldots \wedge p_k \geq 0) \tag{5}$$
$$\alpha_{p=0} \;=\; (p_0 = 0 \wedge p_1 = 0 \wedge \ldots \wedge p_k = 0) \tag{6}$$

So instead of (x), it is sufficient to demand $p_0 > 0$ and $p_1 \geq 0$:

$$a\,c\,d + a\,d + b - c\,b - d > 0 \qquad \wedge \qquad a\,c^2 - c\,a \;\geq\; 0 \tag{xi}$$

Such constraints can be transformed further such that they do not contain subtractions and "$\geq$" anymore. For example, (xi) can be transformed into

$$a\,c\,d + a\,d + b > c\,b + d \;\;\wedge\;\; (a\,c^2 > c\,a \;\;\vee\;\; a\,c^2 = c\,a) \tag{xii}$$

Now to prove termination one has to show the *satisfiability* of such *Diophantine constraints* over the naturals. Def. 2 introduces their syntax and semantics.

Definition 2 (Diophantine Constraints). *Let $\mathcal{A}$ be a set of Diophantine variables. The set of* polynomials $\mathcal{P}$ *is the smallest set with*

- $\mathcal{A} \subseteq \mathcal{P}$ *and* $\mathbb{N} \subseteq \mathcal{P}$
- *If* $\{p,\ q\} \subseteq \mathcal{P}$ *then* $\{p + q,\ p * q\} \subseteq \mathcal{P}$

The set of Diophantine constraints $\mathcal{C}$ *is the smallest set with*

- $\{\mathit{true}, \mathit{false}\} \subseteq \mathcal{C}$
- *If* $\{p,\ q\} \subseteq \mathcal{P}$ *then* $\{p > q,\ p = q\} \subseteq \mathcal{C}$

[5] Of course, $\alpha_{p>0}$ and $\alpha_{p\geq 0}$ are sufficient, but not necessary for $p > 0$ and $p \geq 0$.

- *If* $\{\alpha,\ \beta\} \subseteq \mathcal{C}$ *then* $\{\neg\alpha,\ \alpha \wedge \beta,\ \alpha \vee \beta,\ \alpha \to \beta,\ \alpha \leftrightarrow \beta,\ \alpha \oplus \beta\} \subseteq \mathcal{C}$

A Diophantine interpretation $\mathcal{D}$ *is a mapping* $\mathcal{D} : \mathcal{A} \to \mathbb{N}$*. It can be extended to polynomials by defining* $\mathcal{D}(n) = n$ *for all* $n \in \mathbb{N}$*,* $\mathcal{D}(p + q) = \mathcal{D}(p) + \mathcal{D}(q)$*, and* $\mathcal{D}(p * q) = \mathcal{D}(p) * \mathcal{D}(q)$*. It can also be extended to Diophantine constraints as follows (i.e., we then have* $\mathcal{D} : \mathcal{C} \to \{0, 1\}$*, where* 0 *stands for "false" and* 1 *stands for "true"). As usual,* $\mathcal{D}$ *is called a* model *of a constraint* α *iff* $\mathcal{D}(\alpha) = 1$*.*

- $\mathcal{D}(\mathit{true}) = 1$, $\mathcal{D}(\mathit{false}) = 0$
- $\mathcal{D}(p > q) = 1$ *if* $\mathcal{D}(p) > \mathcal{D}(q)$ *and* $\mathcal{D}(p > q) = 0$*, otherwise*
- $\mathcal{D}(p = q) = 1$ *if* $\mathcal{D}(p) = \mathcal{D}(q)$ *and* $\mathcal{D}(p = q) = 0$*, otherwise*
- $\mathcal{D}(\neg\alpha) = 1$ *if* $\mathcal{D}(\alpha) = 0$ *and* $\mathcal{D}(\neg\alpha) = 0$*, otherwise,*
  and similarly for the other Boolean connectives, where $\oplus$ *is exclusive-or*

For example, let $a \in \mathcal{A}$ and let $\mathcal{D}$ with $\mathcal{D}(a) = 2$. Then $\mathcal{D}(2 * a) = \mathcal{D}(2) * \mathcal{D}(a) = 2 * 2 = 4$ and $\mathcal{D}(1 + a) = 3$. Thus, $\mathcal{D}(2 * a > 1 + a) = 1$, since $4 > 3$.

Similarly, the constraint (xii) is satisfied by the interpretation $\mathcal{D}(a) = 1$, $\mathcal{D}(b) = 0$, $\mathcal{D}(c) = 1$, and $\mathcal{D}(d) = 1$. This Diophantine interpretation instantiates the abstract polynomial interpretation $\mathcal{P}ol_2$ with $\mathsf{half}_{\mathcal{P}ol_2} = a\,x_1 + b$ and $\mathsf{s}_{\mathcal{P}ol_2} = c\,x_1 + d$ to the concrete polynomial interpretation $\mathcal{P}ol_1$ with $\mathsf{half}_{\mathcal{P}ol_1} = x_1$ and $\mathsf{s}_{\mathcal{P}ol_1} = x_1 + 1$ (i.e., we also write[6] $\mathcal{D}(\mathcal{P}ol_2) = \mathcal{P}ol_1$).

To summarize, to prove termination we proceed as follows:

1. Transform the termination problem into inequalities $u \succ v$ or $u \succsim v$ between terms. If one uses the DP method, then one obtains a requirement like (1).
2. Fix an abstract polynomial interpretation and transform the inequalities into $[u] - [v] > 0$ or $[u] - [v] \geq 0$, respectively.
3. Replace $[u] - [v] > 0$ and $[u] - [v] \geq 0$ by $\alpha_{[u]-[v]>0}$ and $\alpha_{[u]-[v]\geq 0}$, cf. (4), (5).
4. Transform the obtained constraint into a Diophantine constraint containing only $>$ and $=$ and no subtractions.
5. Check the satisfiability of the resulting Diophantine constraint. In the next section, we will show how to perform this check using SAT solvers.

3 Encoding Diophantine Constraints to SAT

We have shown that to prove termination, it suffices to prove the satisfiability of a Diophantine constraint. Now we reduce this problem to a SAT problem. We first give the syntax and semantics of propositional logic. Here, we also regard *tuples* of formulas which are interpreted as binary representations of numbers.

Definition 3 (Propositional Logic). *Let* $\mathcal{V}$ *be a set of propositional variables. Then the set of propositional* formulas $\mathcal{F}$ *is the smallest set with*

- $\mathcal{V} \subseteq \mathcal{F}$ *and* $\{0,\ 1\} \subseteq \mathcal{F}$
- *If* $\{\varphi,\ \psi\} \subseteq \mathcal{F}$ *then* $\{\neg\varphi,\ \varphi \wedge \psi,\ \varphi \vee \psi,\ \varphi \to \psi,\ \varphi \leftrightarrow \psi,\ \varphi \oplus \psi\} \subseteq \mathcal{F}$

A propositional interpretation $\mathfrak{I} : \mathcal{V} \to \{0, 1\}$ *can be extended to formulas as*

[6] $\mathcal{D}$ only instantiates abstract coefficients like a, b, c, d. For variables x_i we define $\mathcal{D}(x_i) = x_i$. Thus $\mathcal{D}(a\,x_1 + b) = 1 * x_1 + 0 = x_1$.

follows (i.e., we then have $\Im : \mathcal{F} \to \{0,1\}$*).* $\Im$ *is called a* model *of* φ *iff* $\Im(\varphi) = 1$.

- $\Im(0) = 0$, $\Im(1) = 1$
- $\Im(\neg\varphi) = 1$ *if* $\Im(\varphi) = 0$ *and* $\Im(\neg\varphi) = 0$, *otherwise (similarly for* $\wedge, \vee, \to, \leftrightarrow, \oplus$*)*

Finally, a propositional interpretation can also be extended to tuples *of* n *propositional formulas (with* $n \geq 1$*) by defining* $\Im : \mathcal{F}^n \to \mathbb{N}$ *where*

$$\Im(\langle \varphi_1, \ldots, \varphi_n \rangle) = 2^{n-1} * \Im(\varphi_1) + 2^{n-2} * \Im(\varphi_2) + \ldots + 2 * \Im(\varphi_{n-1}) + \Im(\varphi_n)$$

As an example, let $a_1, a_2 \in \mathcal{V}$ with $\Im(a_1) = 1$ and $\Im(a_2) = 0$. Then we have $\Im(\langle a_1, \neg a_2 \wedge 1, a_2 \rangle) = 4 * \Im(a_1) + 2 * \Im(\neg a_2 \wedge 1) + \Im(a_2) = 4 * 1 + 2 * 1 + 0 = 6$.

Note that one can always delete zeros at the beginning of a tuple since $\Im(\langle 0, \ldots, 0, \varphi_1, \ldots, \varphi_n \rangle) = \Im(\langle \varphi_1, \ldots, \varphi_n \rangle)$ for any interpretation $\Im$. Moreover, we identify one-element-tuples with the element itself since $\Im(\langle \varphi \rangle) = \Im(\varphi)$.

Satisfiability of Diophantine constraints is undecidable (it corresponds to Hilbert's 10th problem). Therefore, we restrict the search to Diophantine interpretations of the form $\mathcal{D} : \mathcal{A} \to \{0, \ldots, 2^k - 1\}$ for a fixed $k \geq 1$. Then variables are only instantiated by numbers that can be represented by k bits. Satisfiability of Diophantine constraints by such restricted interpretations is NP-complete.

We now introduce a mapping $||.|| : \mathcal{C} \to \mathcal{F}$ from Diophantine constraints to propositional formulas such that a constraint α is satisfiable by an interpretation $\mathcal{D} : \mathcal{A} \to \{0, \ldots, 2^k - 1\}$ iff the propositional formula $||\alpha||$ is satisfiable.

We first define $||.||$ on Diophantine variables. Every Diophantine variable is mapped to a tuple of k propositional variables, i.e., we have $||.|| : \mathcal{A} \to \mathcal{V}^k$:

$$||a|| = \langle a_1, \ldots, a_k \rangle \text{ for every Diophantine variable } a \in \mathcal{A} \tag{7}$$

The idea is that $\langle a_1, \ldots, a_k \rangle$ should be the binary representation of a. For any propositional interpretation $\Im$ we define the *corresponding* interpretation $\mathcal{D}_\Im$.

Definition 4 (Corresponding Interpretations). *Let* $\mathcal{V}$ *contain* $a_1, \ldots, a_k$ *for any Diophantine variable* $a \in \mathcal{A}$*. For any propositional interpretation* $\Im$*, we define the* corresponding Diophantine interpretation *as* $\mathcal{D}_\Im(a) = \Im(\langle a_1, \ldots, a_k \rangle)$*.*

So if $k = 2$, then $||a|| = \langle a_1, a_2 \rangle$. The propositional interpretation $\Im(a_1) = 1$ and $\Im(a_2) = 0$ corresponds to the interpretation with $\mathcal{D}_\Im(a) = \Im(\langle a_1, a_2 \rangle) = 2$.

Now we define $||.||$ for natural numbers. Again, $||.||$ maps numbers to their binary representation, i.e., we have $||.|| : \mathbb{N} \to \{0,1\}^+$:

$$||n|| = \langle b^1, \ldots, b^\ell \rangle \text{ for every } n \in \mathbb{N} \tag{8}$$

where all $b^i \in \{0,1\}$ and $n = 2^{\ell-1} * b^1 + 2^{\ell-2} * b^2 + \ldots + 2 * b^{\ell-1} + b^\ell$. To avoid unnecessary long encodings with zeros at the beginning, we require $b^1 = 1$ for all $n > 0$ (i.e., we require that as few bits as possible are used for representing $n > 0$). So for example, we have $||2|| = \langle 1, 0 \rangle$. For the representation of the number 0 we define $||0|| = \langle 0 \rangle$. Note that $\mathcal{D}_\Im(n) = n = \Im(||n||)$ for all $n \in \mathbb{N}$.

Next we define $||.||$ for polynomials. As before, every polynomial is mapped to a tuple of propositional formulas, i.e., $||.|| : \mathcal{P} \to \mathcal{F}^+$. The goal is to obtain the following correspondence for all polynomials p and all interpretations $\Im$:

$$\mathcal{D}_\Im(p) = \Im(||p||) \tag{9}$$

To handle addition and multiplication, we introduce operations $B^+ : \mathcal{F}^+ \times \mathcal{F}^+ \to \mathcal{F}^+$ and $B^* : \mathcal{F}^+ \times \mathcal{F}^+ \to \mathcal{F}^+$ on tuples of propositional formulas. We then define

$$||p+q|| = B^+(||p||, ||q||) \quad \text{and} \quad ||p*q|| = B^*(||p||, ||q||) \tag{10}$$

for all polynomials p and q. We first give the definition of B^+.

- $B^+(\langle \varphi_1, ..., \varphi_n \rangle, \langle \psi_1, ..., \psi_m \rangle) = B^+(\langle \varphi_1, ..., \varphi_n \rangle, \langle \underbrace{0, ..., 0}_{n-m \text{ times}}, \psi_1, ..., \psi_m \rangle)$ if $n > m$
- $B^+(\langle \varphi_1, ..., \varphi_n \rangle, \langle \psi_1, ..., \psi_m \rangle) = B^+(\langle \underbrace{0, ..., 0}_{m-n \text{ times}}, \varphi_1, ..., \varphi_n \rangle, \langle \psi_1, ..., \psi_m \rangle)$ if $n < m$
- $B^+(\langle \varphi \rangle, \langle \psi \rangle) = \langle \varphi \wedge \psi, \varphi \oplus \psi \rangle$
- $B^+(\langle \varphi_1, ..., \varphi_n \rangle, \langle \psi_1, ..., \psi_n \rangle) = \langle B^{2or3}(\varphi_1, \psi_1, \xi_1), B^{1or3}(\varphi_1, \psi_1, \xi_1), \xi_2, ..., \xi_n \rangle$
  if $B^+(\langle \varphi_2, ..., \varphi_n \rangle, \langle \psi_2, ..., \psi_n \rangle) = \langle \xi_1, ..., \xi_n \rangle$

Thus, ξ_1 is the carry resulting from adding $\langle \varphi_2, ..., \varphi_n \rangle$ and $\langle \psi_2, ..., \psi_n \rangle$. Here "$B^{1or3}(\varphi_1, \psi_1, \xi_1)$" abbreviates $\varphi_1 \oplus \psi_1 \oplus \xi_1$ (i.e., either one or all three of the formulas φ_1, ψ_1, and ξ_1 must be true). Similarly, "$B^{2or3}(\varphi_1, \psi_1, \xi_1)$" abbreviates $(\varphi_1 \wedge \psi_1) \vee (\varphi_1 \wedge \xi_1) \vee (\psi_1 \wedge \xi_1)$. For example, we have[7]

$$B^+(\langle 1 \rangle, \langle a_2 \rangle) = \langle 1 \wedge a_2, 1 \oplus a_2 \rangle = \langle a_2, \neg a_2 \rangle$$
$$B^+(\langle 0, 1 \rangle, \langle a_1, a_2 \rangle) = \langle B^{2or3}(0, a_1, a_2), B^{1or3}(0, a_1, a_2), \neg a_2 \rangle = \langle a_1 \wedge a_2,\, a_1 \oplus a_2,\, \neg a_2 \rangle$$

Therefore, we obtain $||1+a|| = B^+(||1||, ||a||) = B^+(\langle 1 \rangle, \langle a_1, a_2 \rangle) = \langle a_1 \wedge a_2,\, a_1 \oplus a_2,\, \neg a_2 \rangle$. Indeed, if $\mathfrak{I}(a_1) = 1$ and $\mathfrak{I}(a_2) = 0$ (i.e., $\mathcal{D}_\mathfrak{I}(a) = 2$), then $\mathcal{D}_\mathfrak{I}(1+a) = 3$ and $\mathfrak{I}(||1+a||) = \mathfrak{I}(\langle a_1 \wedge a_2, a_1 \oplus a_2, \neg a_2 \rangle) = 3$. Hence, $\mathcal{D}_\mathfrak{I}(1+a) = \mathfrak{I}(||1+a||)$, as desired in (9). Next we give the definition of $B^* : \mathcal{F}^+ \times \mathcal{F}^+ \to \mathcal{F}^+$.

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

$$\begin{aligned} \text{E.g., } ||2*a|| &= B^*(||2||, ||a||) = B^*(\langle 1, 0 \rangle, \langle a_1, a_2 \rangle) \\ &= B^+(\langle 1 \wedge a_1, 0 \wedge a_1, 0 \rangle, B^*(\langle 1, 0 \rangle, \langle a_2 \rangle)) = B^+(\langle a_1, 0, 0 \rangle, \langle a_2, 0 \rangle) \\ &= B^+(\langle a_1, 0, 0 \rangle, \langle 0, a_2, 0 \rangle) = \langle 0, a_1, a_2, 0 \rangle = \langle a_1, a_2, 0 \rangle. \end{aligned}$$

Indeed, if $\mathfrak{I}(a_1) = 1$ and $\mathfrak{I}(a_2) = 0$ (i.e., $\mathcal{D}_\mathfrak{I}(a) = 2$), then $\mathcal{D}_\mathfrak{I}(2*a) = 4 = \mathfrak{I}(\langle a_1, a_2, 0 \rangle) = \mathfrak{I}(||2*a||)$, as desired in (9). We state (9) as a general lemma.

Lemma 5 (Correctness of Encoding Polynomials). *For every polynomial $p \in \mathcal{P}$ and every propositional interpretation $\mathfrak{I}$, we have $\mathcal{D}_\mathfrak{I}(p) = \mathfrak{I}(||p||)$.*[8]

Now we extend the mapping $||.||$ to $||.|| : \mathcal{C} \to \mathcal{F}$. Thus, every Diophantine constraint is mapped to a formula (not to a tuple). Obviously, we define

$$||true|| = 1 \quad \text{and} \quad ||false|| = 0 \tag{11}$$

For Diophantine constraints that are polynomial inequalities or equalities, we introduce operations $B^> : \mathcal{F}^+ \times \mathcal{F}^+ \to \mathcal{F}$ and $B^= : \mathcal{F}^+ \times \mathcal{F}^+ \to \mathcal{F}$ and define

[7] For readability, we perform Boolean simplifications like replacing $1 \wedge a_2$ by a_2, etc.
[8] All proofs can be found in [10].

$$||p > q|| = B^{>}(||p||, ||q||) \quad \text{and} \quad ||p = q|| = B^{=}(||p||, ||q||) \tag{12}$$

for all polynomials p and q. To define $B^{>}$ and $B^{=}$, we first handle the case where the argument tuples have different lengths. For $\circ \in \{=, >\}$ we define

- $B^{\circ}(\langle \varphi_1, ..., \varphi_n \rangle, \langle \psi_1, ..., \psi_m \rangle) = B^{\circ}(\langle \varphi_1, ..., \varphi_n \rangle, \langle \underbrace{0, ..., 0}_{n-m \text{ times}}, \psi_1, ..., \psi_m \rangle)$ if $n > m$
- $B^{\circ}(\langle \varphi_1, ..., \varphi_n \rangle, \langle \psi_1, ..., \psi_m \rangle) = B^{\circ}(\langle \underbrace{0, ..., 0}_{m-n \text{ times}}, \varphi_1, ..., \varphi_n \rangle, \langle \psi_1, ..., \psi_m \rangle)$ if $n < m$

Now we define $B^{>}$ and $B^{=}$ for tuples of equal length.

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

For example,
$$\begin{aligned} ||2 * a > 1 + a|| &= B^{>}(||2 * a||, ||1 + a||) \\ &= B^{>}(\langle a_1, a_2, 0 \rangle, \langle a_1 \wedge a_2, a_1 \oplus a_2, \neg a_2 \rangle) \\ &= (a_1 \wedge \neg a_2) \vee ((a_1 \leftrightarrow a_2) \wedge ((a_2 \wedge \neg(a_1 \oplus a_2)) \vee \ldots)) \\ &= a_1 \end{aligned}$$

So $||2 * a > 1 + a||$ only holds for the propositional interpretations where $\mathfrak{I}(a_1) = 1$. Indeed, the corresponding Diophantine interpretations with $\mathcal{D}_{\mathfrak{I}}(a) = 2$ or $\mathcal{D}_{\mathfrak{I}}(a) = 3$ are the only ones satisfying the constraint $2 * a > 1 + a$ (if we are restricted to $\mathcal{D}(a) \in \{0, \ldots, 3\}$). Finally, we define $||.||$ on non-atomic constraints:

$$||\neg\alpha|| = \neg||\alpha|| \quad \text{and} \quad ||\alpha \circ \beta|| = ||\alpha|| \circ ||\beta|| \text{ for all } \circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\} \tag{13}$$

By Thm. 6, our encoding defined in (7), (8), (10), (11), (12), (13) is correct.

Theorem 6 (Correctness of Encoding Diophantine Constraints). *For every $\alpha \in \mathcal{C}$ and every propositional interpretation $\mathfrak{I}$, we have $\mathcal{D}_{\mathfrak{I}}(\alpha) = \mathfrak{I}(||\alpha||)$.*

So to determine the satisfiability of a Diophantine constraint α by a Diophantine interpretation with numbers from $\{0, \ldots, 2^k - 1\}$, we now encode α as a propositional formula $||\alpha||$ and then use a SAT solver to find a model $\mathfrak{I}$ of $||\alpha||$. Thm. 7 shows that the size of our encoding is polynomial.

Theorem 7 (Size of Encoding). *Let $\alpha \in \mathcal{C}$ such that every number in α is $\leq 2^k - 1$. Then the size of $||\alpha||$ is in $\mathcal{O}(|\alpha|^2 * k^2)$, where $|\alpha|$ is the size of α.*

4 Polynomials with Negative Constant

Now we regard polynomials $f_{\mathcal{P}ol}$ which may have a negative constant coefficient (i.e., in (2) one may have $a_0 < 0$). All other coefficients still have to be natural numbers. As demonstrated by the tools TTT [17] and AProVE [14] in the termination competitions, such polynomials (in connection with the DP method) are very helpful in practice. We show how to extend our approach in order to use SAT solvers also for such polynomial interpretations.

As in [3, Ex. 4.28], we replace the rules (v) and (vi) of our TRS by

$$\mathsf{bits}(\mathsf{s}(x)) \rightarrow \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))).$$

Instead of (viii) and (ix) we get the DPs $\mathsf{bits}^\sharp(\mathsf{s}(x)) \to \mathsf{half}^\sharp(\mathsf{s}(x))$ and $\mathsf{bits}^\sharp(\mathsf{s}(x)) \to \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x)))$. Now there is no polynomial interpretation with non-negative coefficients where the DPs are strictly and the rules are weakly decreasing.

Thus, we use a polynomial interpretation $\mathcal{P}ol_3$ with $\mathsf{half}_{\mathcal{P}ol_3} = x_1 - 1$. However, if one extends such interpretations to terms naively, then terms could be mapped to negative numbers and thus, the resulting order would not be well founded. Hence, [17] proposed the following modification in the definition of $[.]$: $[x] = x$ for all variables x and $[f(t_1, \ldots, t_n)] = \max(f_{\mathcal{P}ol}([t_1], \ldots, [t_n]), 0)$. So if $\mathsf{s}_{\mathcal{P}ol_3} = x_1 + 1$, then $[\mathsf{s}(\mathsf{half}(x))]_{\mathcal{P}ol_3} = \max(\max(x - 1, 0) + 1, 0)$. Now one can again replace inequalities $u \succ v$ (resp. $u \succsim v$) by $[u] > [v]$ (resp. $[u] \geq [v]$).

We are interested in *abstract* polynomial interpretations with variable coefficients. To find suitable values for the coefficients, up to now inequalities like $[u] > [v]$ were transformed into Diophantine constraints by building $\alpha_{[u]-[v]>0}$ etc., cf. (4) and (5). Here, we simply required all coefficients of the polynomial $[u] - [v]$ to be non-negative resp. positive. However, now $[u] - [v]$ contains "max" (i.e., it is no longer a polynomial). Thus, it is unclear how to transform $[u] > [v]$ into a satisfiability problem of a Diophantine constraint.

To solve this problem, let us first regard *concrete* polynomial interpretations (where the coefficients are actual numbers). Here, the occurrences of "max" in inequalities $[u] > [v]$ could be eliminated by case analyses. But to increase efficiency, [17] presented an alternative approach to transform inequalities like $[u] > [v]$ into ordinary polynomial inequalities without "max". The idea is to define an under-approximation $[.]^{left}$ and an over-approximation $[.]^{right}$ which do not contain "max" anymore. Then instead of $[u] > [v]$ one requires $[u]^{left} > [v]^{right}$.

Definition 8 ($[.]^{left}$ and $[.]^{right}$ for Concrete Interpretations [17]). *For every polynomial p we denote its constant part by $con(p)$ and the non-constant part $p - con(p)$ by $ncon(p)$. For any concrete polynomial interpretation $\mathcal{P}ol$ and any term t, we define the polynomials $[t]^{left}_{\mathcal{P}ol}$ and $[t]^{right}_{\mathcal{P}ol}$ as follows:*[9]

$$[t]^{left} = \begin{cases} t & \textit{if } t \textit{ is a variable} \\ 0 & \textit{if } t = f(t_1, \ldots, t_n),\ ncon(p_1) = 0, \textit{ and } 0 > con(p_1) \\ p_1 & \textit{if } t = f(t_1, \ldots, t_n), \textit{ otherwise} \end{cases}$$

$$[t]^{right} = \begin{cases} t & \textit{if } t \textit{ is a variable} \\ ncon(p_2) & \textit{if } t = f(t_1, \ldots, t_n) \textit{ and } 0 > con(p_2) \\ p_2 & \textit{if } t = f(t_1, \ldots, t_n), \textit{ otherwise} \end{cases}$$

where $p_1 = f_{\mathcal{P}ol}([t_1]^{left}, \ldots, [t_n]^{left})$ and $p_2 = f_{\mathcal{P}ol}([t_1]^{right}, \ldots, [t_n]^{right})$.

As shown in [17], we have $[t]^{left} \leq [t] \leq [t]^{right}$ for all terms t. Moreover, if the polynomial interpretation has no negative constants, then we have $[t]^{left} = [t] = [t]^{right}$. For the polynomial interpretation with $\mathsf{half}_{\mathcal{P}ol_3} = x_1 - 1$, we obtain

$$[\mathsf{half}(x)]^{left}_{\mathcal{P}ol_3} = x - 1 \qquad [\mathsf{half}(x)]_{\mathcal{P}ol_3} = \max(x - 1, 0) \qquad [\mathsf{half}(x)]^{right}_{\mathcal{P}ol_3} = x \quad \text{(xiii)}$$

The reason is that for both $i \in \{1, 2\}$, we have $p_i = \mathsf{half}_{\mathcal{P}ol_3}(x) = x - 1$ and thus

[9] If $\mathcal{P}ol$ is clear from the context we again omit the subscript "$\mathcal{P}ol$".

$ncon(p_i) = x$ and $con(p_i) = -1$. If $\mathcal{P}ol_3$ is defined like our previous interpretation $\mathcal{P}ol_1$ on all remaining function symbols except half, then we obtain $[u]^{\mathit{left}} > [v]^{\mathit{right}}$ for all DPs $u \to v$ and $[\ell]^{\mathit{left}} \geq [r]^{\mathit{right}}$ for all rules $\ell \to r$. Thus, the termination of our modified example can now easily be shown.

The disadvantage of Def. 8 is that one can only compute $[t]^{\mathit{left}}$ and $[t]^{\mathit{right}}$ for concrete polynomial interpretations.[10] However, if one wants to find the coefficients of the polynomial interpretations automatically, then it would be better to start with *abstract* polynomial interpretations again where the coefficients a_i in (2) are left open (i.e., they are *variable coefficients*).

For example, we would use an abstract interpretation $\mathcal{P}ol_2$ with $\mathsf{half}_{\mathcal{P}ol_2} = a\,x_1 + \boldsymbol{b}$. Here, a may only be instantiated by natural numbers, whereas we denote Diophantine variables like $\boldsymbol{b}$ that may be instantiated by integers in **bold** face. However, to compute $[\mathsf{half}(x)]^{\mathit{left}}_{\mathcal{P}ol_2}$ and $[\mathsf{half}(x)]^{\mathit{right}}_{\mathcal{P}ol_2}$ we would have to decide whether $ncon(p_i) = a\,x$ and $con(p_i) = \boldsymbol{b}$ are equal to resp. less than 0. This of course depends on the instantiation of the variable coefficients a and $\boldsymbol{b}$.

Therefore, we now modify Def. 8 to make it suitable for abstract polynomial interpretations. The idea is to introduce new variables $\boldsymbol{b_t^{left}}$ and b_t^{right} for any term t and to create Diophantine constraints α_t^{left} and $\alpha_t^{\mathit{right}}$ which guarantee that $\boldsymbol{b_t^{left}}$ and b_t^{right} are instantiated correctly. To this end, we express the conditions $ncon(p_1) = 0$ and $0 > con(p_i)$ from Def. 8 as Diophantine constraints.

Definition 9 ($[.]^{\mathit{left}}$ and $[.]^{\mathit{right}}$ for Abstract Interpretations). *For any abstract polynomial interpretation $\mathcal{P}ol$ and any term t, we define:*

- *If t is a variable, then $[t]^{\mathit{left}} = t$, $[t]^{\mathit{right}} = t$, $\alpha_t^{\mathit{left}} = \mathit{true}$, and $\alpha_t^{\mathit{right}} = \mathit{true}$.*
- *If $t = f(t_1, \dots, t_n)$, then*[11] $[t]^{\mathit{left}} = ncon(p_1) + \boldsymbol{b_t^{left}}$, $[t]^{\mathit{right}} = ncon(p_2) + b_t^{\mathit{right}}$,

$$\begin{aligned}
\alpha_t^{\mathit{left}} \; = \; & \alpha_{t_1}^{\mathit{left}} \wedge \ldots \wedge \alpha_{t_n}^{\mathit{left}} \wedge (\quad \alpha_{ncon(p_1)=0} \wedge 0 > con(p_1) \;\; \to \boldsymbol{b_t^{left}} = 0) \\
& \qquad\qquad\qquad\quad \wedge (\neg(\alpha_{ncon(p_1)=0} \wedge 0 > con(p_1)) \to \boldsymbol{b_t^{left}} = con(p_1)) \\
\alpha_t^{\mathit{right}} = \; & \alpha_{t_1}^{\mathit{right}} \wedge \ldots \wedge \alpha_{t_n}^{\mathit{right}} \wedge (\quad 0 > con(p_2) \;\; \to b_t^{\mathit{right}} = 0) \\
& \qquad\qquad\qquad\quad \wedge (\neg(0 > con(p_2)) \to b_t^{\mathit{right}} = con(p_2))
\end{aligned}$$

Here, p_1 and p_2 are defined as in Def. 8 and $\alpha_{ncon(p_i)=0}$ is defined as in (6).

For $\mathsf{half}_{\mathcal{P}ol_2} = a\,x_1 + \boldsymbol{b}$ and $t = \mathsf{half}(x)$, we have $ncon(p_i) = a\,x$, $con(p_i) = \boldsymbol{b}$,

$$[\mathsf{half}(x)]^{\mathit{left}}_{\mathcal{P}ol_2} = a\,x + \boldsymbol{b_t^{left}} \quad \text{and} \quad [\mathsf{half}(x)]^{\mathit{right}}_{\mathcal{P}ol_2} = a\,x + b_t^{\mathit{right}} \tag{xiv}$$

$$\alpha_t^{\mathit{left}} = ((a = 0 \wedge 0 > \boldsymbol{b}) \to \boldsymbol{b_t^{left}} = 0) \;\wedge\; (\neg(a = 0 \wedge 0 > \boldsymbol{b}) \to \boldsymbol{b_t^{left}} = \boldsymbol{b}) \tag{xv}$$

$$\alpha_t^{\mathit{right}} = ((0 > \boldsymbol{b}) \to b_t^{\mathit{right}} = 0) \;\wedge\; (\neg(0 > \boldsymbol{b}) \to b_t^{\mathit{right}} = \boldsymbol{b}) \tag{xvi}$$

Thm. 10 shows that Def. 9 extends Def. 8 to abstract interpretations correctly.

Theorem 10 (Correspondence of Def. 8 and 9). *Let $\mathcal{D}$ be a Diophantine interpretation (which may also map* **bold** *variables to integers). Let $\mathcal{P}ol$ be an abstract polynomial interpretation, and let t be a term. Then $\mathcal{D}(\alpha_t^{\mathit{left}}) = 1$ implies $\mathcal{D}([t]^{\mathit{left}}_{\mathcal{P}ol}) = [t]^{\mathit{left}}_{\mathcal{D}(\mathcal{P}ol)}$ and $\mathcal{D}(\alpha_t^{\mathit{right}}) = 1$ implies $\mathcal{D}([t]^{\mathit{right}}_{\mathcal{P}ol}) = [t]^{\mathit{right}}_{\mathcal{D}(\mathcal{P}ol)}$.*

[10] Thus, current implementations for negative polynomials like TTT and AProVE simply *test* several choices for the coefficients. More sophisticated algorithms for *systematically finding* coefficients like [8] only work for non-negative coefficients.

[11] Note that according to Def. 8, $[t]^{\mathit{left}} = ncon(p_1)$ if $ncon(p_1) = 0$ and $0 > con(p_1)$.

For example, let $\mathcal{D}$ be an interpretation which turns the abstract polynomial interpretation $\mathcal{P}ol_2$ into the concrete interpretation $\mathcal{P}ol_3$. Thus, we have $\mathcal{D}(a) = 1$ and $\mathcal{D}(\boldsymbol{b}) = -1$ and indeed, $\mathcal{D}(\mathsf{half}_{\mathcal{P}ol_2}) = \mathcal{D}(a\,x_1 + \boldsymbol{b}) = x_1 - 1 = \mathsf{half}_{\mathcal{P}ol_3}$. To satisfy the Diophantine constraints α_t^{left} and α_t^{right} in (xv) and (xvi), we must have $\mathcal{D}(\boldsymbol{b_t^{left}}) = -1$ and $\mathcal{D}(b_t^{right}) = 0$. Then by (xiii) and (xiv) we indeed obtain

$$\begin{aligned}
\mathcal{D}([\mathsf{half}(x)]^{left}_{\mathcal{P}ol_2}) &= \mathcal{D}(ax + \boldsymbol{b_t^{left}}) &&= x - 1 &&= [\mathsf{half}(x)]^{left}_{\mathcal{P}ol_3}\\
\mathcal{D}([\mathsf{half}(x)]^{right}_{\mathcal{P}ol_2}) &= \mathcal{D}(ax + b_t^{right}) &&= x &&= [\mathsf{half}(x)]^{right}_{\mathcal{P}ol_3}
\end{aligned}$$

So we generate Diophantine constraints containing **bold** variables like $\boldsymbol{b}$ and $\boldsymbol{b_t^{left}}$ which may be instantiated by *integers*. However, our encoding to propositional formulas in Sect. 3 only handles instantiations with *natural* numbers. Therefore, we now show how to remove **bold** variables from constraints α.

In the encoding $||\alpha||$, we restricted ourselves to interpretations $\mathcal{D}$ where for all (non-**bold**) variables a we have $\mathcal{D}(a) \in \{0, \ldots, 2^k - 1\}$ for some fixed $k \geq 1$. Now one has to fix an additional number $n \geq 0$ and for all **bold** variables $\boldsymbol{a}$, we restrict ourselves to $\mathcal{D}(\boldsymbol{a}) \in \{-n, \ldots, 2^k - 1 - n\}$. Hence, to encode a Diophantine constraint α with **bold** variables, we first replace every **bold** variable $\boldsymbol{a}$ in α by "$a - n$" for a fresh (non-**bold**) variable a. Then (after removing subtractions), one can again use our encoding $||.||$ from Sect. 3.

To summarize, the procedure from the end of Sect. 2 to transform a termination problem into a satisfiability problem is now modified as follows:

1. Transform the termination problem to inequalities $u \succ v$ or $u \succsim v$, cf. (1).
2. Fix an abstract polynomial interpretation and transform the inequalities into $[u]^{left} - [v]^{right} > 0$ or $[u]^{left} - [v]^{right} \geq 0$, respectively. Add the conjunction of all corresponding constraints α_u^{left} and α_v^{right}.
3. Replace $[u]^{left} - [v]^{right} \mathrel{_{(}\geq_{)}} 0$ by $\alpha_{[u]^{left} - [v]^{right} \mathrel{_{(}\geq_{)}} 0}$.
4. Fix a number $n \geq 0$ and replace all Diophantine variables $\boldsymbol{a}$ that may be instantiated by integers by "$a - n$" for a fresh variable a.
5. Remove "$\geq$" and subtractions from the obtained constraint and check its satisfiability using SAT solving as in Sect. 3.

5 Implementation, Experiments, and Conclusion

We implemented our new SAT-based approach for polynomial interpretations in the termination prover AProVE [14]. We used the MiniSAT solver [9] and to convert formulas to CNF, we applied SAT4J's [21] implementation of Tseitin's algorithm [24]. For efficiency, our implementation uses several optimizations:

(a) Simplification: In addition to standard simplifications for Diophantine constraints and for propositional formulas, we developed a new graph-based approach to detect possible simplifications of Diophantine constraints quickly. We build a graph whose nodes consist of all occurring Diophantine variables and of all possible values they can take (e.g., $\{0, ..., 2^k - 1\}$). An edge from a node n_1 to n_2 denotes that $\mathcal{D}(n_1) \geq \mathcal{D}(n_2)$ for any Diophantine model $\mathcal{D}$ of the given Diophantine constraint. This graph is constructed and maintained while performing the other simplifications. Whenever there is a non-trivial strongly connected component (SCC) in the graph, we can deduce that all its nodes must take the same value under any Diophantine model. If there is more than one number in the SCC, then the Diophantine constraint is not satisfiable. If there is one number in the SCC, we instantiate all Diophantine variables in the SCC by that number. If the SCC only consists of Diophantine variables, we choose an arbitrary one and replace all other variables in the SCC by the chosen one.

(b) Sharing: We use sharing for common subexpressions, both on the level of Diophantine constraints and on the level of propositional formulas.

(c) Tracking maximum values: By taking into account that Diophantine variables are only instantiated by values from a certain set (e.g., $\{0, ..., 2^k - 1\}$), one can keep track of the maximum possible values for all polynomials occurring in the Diophantine constraint. This can help to improve the conversion from Diophantine constraints to tuples of propositional formulas. The reason is that we can detect cases where the most significant bits are equivalent to 0.

As an example, suppose that all Diophantine variables can take values from $\{0, ..., 3\}$ and that consequently, the conversion $||.||$ transforms Diophantine variables into tuples of two propositional variables (i.e., $k = 2$). Note that by definition, $B^*(\langle\varphi_1, ..., \varphi_n\rangle, \langle\psi_1, ..., \psi_m\rangle)$ is always a tuple of length $n+m$, if $m \geq 2$. So if $a, b, c \in \mathcal{A}$, then $||a||$ and $||b||$ have length 2, $||a*b||$ has length 4, and $||a*b*c||$ has length 6. However, if one takes the ranges of the coefficients into account, then one can determine that $a*b*c$ has at most the value $3*3*3 = 27$. Thus, only 5 bits are needed for $||a*b*c||$, i.e., the most significant bit of $||a*b*c||$ is always equivalent to 0. Therefore, it can be omitted (i.e., one should delete the leftmost formula in the 6-tuple $||a*b*c||$, resulting in a 5-tuple).

This optimization is particularly helpful when using other ranges than $\{0, ..., 2^k - 1\}$ (e.g., when using $\{0, 1, 2\}$ instead of $\{0, 1, 2, 3\}$). Then we have to introduce subformulas that prohibit certain values for the Diophantine variables, but this usually pays off due to the reduced search space.

To evaluate our new SAT-based implementation of polynomial interpretations (AProVE-SAT), we compared it with the non-SAT-based implementations in the termination tools AProVE 1.2 and TTT [17]. In addition, we experimented with a version of AProVE which uses the Diophantine solver of the CiME-tool [7] (AProVE-CiME). The implementations in AProVE 1.2 and AProVE-CiME solve Diophantine constraints by a specialized finite domain constraint satisfaction procedure [8], while TTT uses a "generate-and-test" approach instead. Moreover, we considered a variant AProVE-CLP which applies the *constraint logic programming* engine of SICStus Prolog to find polynomial interpretations.

Finally, we also implemented a variant AProVE-PB which uses the *pseudo-boolean solver* Pueblo [23]. Here, instead of encoding Diophantine constraints to propositional formulas, we adapted the encoding $||.||$ from Sect. 3 in order to yield *pseudo-boolean constraints*: For Diophantine variables a over $\{0, \ldots, 2^k - 1\}$ we now define $||a|| = 2^{k-1}\, a_1 + \ldots + 2\, a_{k-1} + a_k$, and we define $||n|| = n$

for $n \in \mathbb{N}$ and $||p \circ q|| = ||p|| \circ ||q||$ for polynomials $p, q \in \mathcal{P}$ and $\circ \in \{+, *\}$. Afterwards, the resulting constraints are linearized.

We tested the six tools on all 865 TRSs from the *Termination Problem Data Base 3.2*.[12] This is the collection of examples used in the *International Competition of Termination Tools 2006*. For our experiments, the tools were run on an AMD Athlon 64 at 2.2 GHz. To measure the effect of the different implementations for polynomial interpretations, we configured all tools to use only a basic version of the DP method and no other termination technique.[13]

For each example, we imposed a time limit of 60 seconds (corresponding to the way tools are evaluated in the annual competition) or of 10 minutes, indicated by "*Limit*" in the following table. The columns "*Yes*" and "*TO*" show the number of TRSs for which proving termination with the given configuration succeeds or times out. Finally, "*Time*" gives the total time in seconds needed for analyzing all 865 examples. The column "*Range*" specifies the range of the coefficients of polynomials (i.e., if the "*Range*" is n, then we only searched for coefficients from $\{0, ..., n\}$). The column "*Degree*" gives the degree of the polynomials. If the "*Degree*" is 1, then we used linear polynomials and "sm" means that we used simple-mixed[14] polynomials (these are not available in TTT).

			AProVE-SAT			AProVE-PB			AProVE 1.2		
Limit	Range	Degree	Yes	TO	Time	Yes	TO	Time	Yes	TO	Time
60s	1	1	421	0	45.5	421	0	61.6	421	1	151.8
60s	2	1	431	0	91.8	431	0	158.5	414	48	3633.2
60s	3	1	434	0	118.6	434	1	222.1	408	81	5793.2
60s	3	sm	440	51	5585.9	427	82	7280.3	404	171	11608.1
10m	1	1	421	0	45.5	421	0	61.6	421	1	691.8
10m	2	1	431	0	91.8	431	0	158.5	418	41	27888.4
10m	3	1	434	0	118.6	434	0	689.6	415	53	38286.4

			AProVE-CLP			AProVE-CiME			TTT		
Limit	Range	Degree	Yes	TO	Time	Yes	TO	Time	Yes	TO	Time
60s	1	1	420	16	1357.8	408	1	168.3	326	32	2568.5
60s	2	1	420	37	3558.3	408	43	3201.0	335	83	5677.6
60s	3	1	407	91	6459.5	402	67	5324.1	338	110	7426.9
60s	3	sm	367	145	10357.4	361	147	10107.7			
10m	1	1	421	11	7852.2	408	0	332.7	328	16	14007.8
10m	2	1	423	25	18795.6	412	33	22190.4	337	68	45046.6
10m	3	1	420	51	41493.8	407	46	33873.6	340	91	61209.2

The comparison of the SAT-based configurations AProVE-SAT and AProVE-PB with the non-SAT-based configurations shows that the provers based on SAT solving with our proposed encoding are faster by orders of magnitude. 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 "*Yes*"-results, i.e., to increase the power of automated termination proving). Note that for *Degree* = 1, there are no timeouts in the configuration AProVE-SAT, whereas the non-SAT-based configurations have many timeouts. 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 configurations. To indicate the size of the

[12] The data base is available from `http://www.lri.fr/~marche/tpdb/`.

[13] Such a configuration was not possible for other tools beside AProVE, TTT, and CiME.

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

SAT problems obtained, the largest resulting propositional formula contained almost 3.5 million variables and more than 12 million clauses. Comparing the SAT-based configurations AProVE-SAT and AProVE-PB shows that the approach of converting termination problems to propositional formulas is currently preferable to the related approach of converting them to pseudo-boolean constraints.

We also ran experiments with higher ranges but it turned out that they are rarely needed. For *Degree* = 1 and *Limit* = 10 minutes, a range of 6 would increase the number of "*Yes*"-results from 434 to 436 while the runtime increases from 118.6 to 748.1 seconds. Even if one uses a range of 63, the number of "*Yes*"-results does not increase further, but the runtime goes up to 56235.5 seconds.

The next table shows the effect of our optimizations (with linear polynomials and a 60 seconds time limit). While AProVE-SAT uses all optimizations (a) - (c), we also give the results obtained if one omits any one of these optimizations. The table demonstrates that each optimization has a considerable positive effect, especially if one uses higher ranges for the coefficients.

	AProVE-SAT			no optimization (a)			no optimization (b)			no optimization (c)		
Range	Yes	TO	Time	Yes	TO	Time	Yes	TO	Time	Yes	TO	Time
1	421	0	45.5	421	0	56.6	421	0	49.7	421	0	50.1
2	431	0	91.8	431	0	107.5	431	0	93.9	431	0	114.7
3	434	0	118.6	434	1	159.4	434	0	202.8	434	0	138.7

The last table demonstrates the use of SAT solving for negative linear polynomials with a time limit of 60 seconds. If the "*Range*" is n, then now the constant coefficient may take values from $\{-n, ..., n\}$. Again, the SAT-based configuration is much faster and substantially more powerful than the non-SAT-based ones. Compared to the results for non-negative polynomials, a few timeouts occur for larger ranges, but negative polynomials increase the power significantly whereas the runtimes only increase moderately. In future work, we will extend our SAT encoding in order to deal also with polynomials where other (non-constant) coefficients can be negative [17].

	AProVE-SAT			AProVE 1.2			TTT		
Range	Yes	TO	Time	Yes	TO	Time	Yes	TO	Time
1	440	0	98.0	441	22	1863.7	341	106	7307.3
2	479	1	305.4	460	126	8918.3	360	181	12337.3
3	483	4	1092.4	434	221	15570.9	361	247	16927.7

As mentioned in Sect. 1, the SAT-based implementation of polynomial interpretations was used by AProVE in the *International Competition of Termination Tools 2006*. Here, AProVE was configured to use several other termination techniques in addition to polynomial interpretations. 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 seconds for each example, AProVE could prove termination of 633 TRSs and thereby it was the winner of the competition.

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.[15] To experiment with our implementation, for further details on our experiments (also with other SAT solvers), and for all proofs please see [10].

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

Acknowledgments. We thank Daniel Le Berre for helpful comments.

References

1. E. Annov, M. Codish, J. Giesl, P. Schneider-Kamp, and R. Thiemann. A SAT-based implementation for RPO termination. In *Short Papers of LPAR '06*, 2006.
2. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133-178, 2000.
3. T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-2001-09, RWTH Aachen, 2001.
4. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge, 1998.
5. M. Codish, V. Lagoon, and P. Stuckey. Solving partial order constraints for LPO termination. In *Proc. RTA '06*, LNCS 4098, p. 4-18, 2006.
6. M. Codish, P. Schneider-Kamp, V. Lagoon, R. Thiemann, and J. Giesl. SAT solving for argument filterings. In *Proc. LPAR '06*, LNAI 4246, p. 30-44, 2006.
7. E. Contejean, C. Marché, B. Monate, and X. Urbain. CiME. `http://cime.lri.fr`.
8. E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *J. Aut. Reason.*, 34(4):325-363, 2005.
9. N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. SAT '03*, LNCS 2919, p. 502-518, 2004.
10. Empirical evaluation of "SAT solving for termination analysis with polynomial interpretations". `http://aprove.informatik.rwth-aachen.de/eval/SATPOLO`.
11. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. In *Proc. IJCAR '06*, LNAI 4130, p. 574-588, 2006.
12. J. Giesl, R. Thiemann, P. Schneider-Kamp. The DP framework: Combining Techniques for Automated Termination Proofs. *LPAR'04*, LNAI 3452, p.301-331, 2005.
13. J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated termination analysis for Haskell: From term rewriting to programming languages. In *Proc. RTA '06*, LNCS 4098, p. 297-312, 2006.
14. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the DP framework. *Proc. IJCAR '06*, LNAI 4130, p. 281-286, 2006.
15. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3): 155-203, 2006.
16. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172-199, 2005.
17. N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474-511, 2007.
18. D. Hofbauer and J. Waldmann. Termination of string rewriting with matrix interpretations. In *Proc. RTA '06*, LNCS 4098, p. 328-342, 2006.
19. H. Hong and D. Jakuš. Testing positiveness of polynomials. *JAR*, 21(1):23-38, 1998.
20. D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.
21. D. Le Berre et al. SAT4J satisfiability library for Java. `http://www.sat4j.org`.
22. P. Schneider-Kamp, J. Giesl, A. Serebrenik, R. Thiemann. Automated termination analysis for logic programs by term rewriting. In *Proc. LOPSTR '06*, LNCS, 2007.
23. H. M. Sheini and K. A. Sakallah. Pueblo: A hybrid pseudo-boolean SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:61-96, 2006.
24. G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, p. 115-125, 1968.
25. H. Zankl, N. Hirokawa, and A. Middeldorp. Constraints for argument filterings. In *Proc. SOFSEM '07*, LNCS 4362, p. 579-590, 2007.
26. H. Zankl and A. Middeldorp. KBO as a satisfaction problem. *Proc. WST'06*, 2006.