Satisfying KBO Constraints*

Harald Zankl and Aart Middeldorp

Institute of Computer Science
University of Innsbruck
Austria

Abstract. This paper presents two new approaches to prove termination of rewrite systems with the Knuth-Bendix order efficiently. The constraints for the weight function and for the precedence are encoded in (pseudo-)propositional logic and the resulting formula is tested for satisfiability. Any satisfying assignment represents a weight function and a precedence such that the induced Knuth-Bendix order orients the rules of the encoded rewrite system from left to right.

1 Introduction

This paper is concerned with proving termination of term rewrite systems (TRSs) with the Knuth-Bendix order (KBO), a method invented by Knuth and Bendix in [14] well before termination research in term rewriting became a very popular and competitive endeavor (as witnessed by the annual termination competition).[1] We know of only two termination tools that contain an implementation of KBO, AProVE [11] and TTT [12], but neither of these tools incorporate KBO in their fully automatic mode for the TRS category. This is perhaps due to the fact that the algorithms known for deciding KBO orientability ([5,15]) are not easy to implement efficiently, despite the fact that the problem is known to be decidable in polynomial time [15]. The aim of this paper is to make KBO a more attractive choice for termination tools by presenting two simple encodings of KBO orientability into (pseudo-)propositional logic such that checking satisfiability of the resulting formula amounts to proving KBO termination.

Kurihara and Kondo [16] were the first to encode a termination method for term rewriting into propositional logic. They showed how to encode orientability with respect to the lexicographic path order as a satisfaction problem. Codish *et al.* [3] presented a more efficient formulation for the properties of a precedence. In [4,22] encodings of argument filterings are presented which can be combined with propositional encodings of reduction pairs in order to obtain logic-based implementations of the dependency pair method. Propositional encodings of other termination methods are described in [9,10,13].

In Section 2 the necessary definitions for KBO are presented. Section 3 introduces a purely propositional encoding of KBO also describing the optimizations

* This research is supported by FWF (Austrian Science Fund) project P18763. Some of the results in this paper were first announced in [23].

[1] `www.lri.fr/~marche/termination-competition`

applied in the implementation. In Section 4 an alternative encoding is given using pseudo-boolean constraints. We compare the power and run times of our implementations with the ones of AProVE and T$_{\text{T}}$T in Section 5 and show the enormous gain in efficiency. We draw some conclusions in Section 6. One of these is that our pseudo-boolean encoding of KBO revealed a bug in MiniSat+. Section 7 summarizes the main contributions of this paper.

2 Preliminaries

We assume familiarity with the basics of term rewriting (e.g. [2]). In this preliminary section we recall the definition of KBO. A *quasi-precedence* $\succsim$ (*strict precedence* $\succ$) is a quasi-order (proper order) on a signature $\mathcal{F}$. Sometimes we find it convenient to call a quasi-precedence simply precedence. A *weight function* for a signature $\mathcal{F}$ is a pair (w, w_0) consisting of a mapping $w\colon \mathcal{F} \to \mathbb{N}$ and a constant $w_0 > 0$ such that $w(c) \geqslant w_0$ for every constant $c \in \mathcal{F}$. Let $\mathcal{F}$ be a signature and (w, w_0) a weight function for $\mathcal{F}$. The *weight* of a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ is defined as follows:

$$w(t) = \begin{cases} w_0 & \text{if } t \text{ is a variable,} \\ w(f) + \sum_{i=1}^{n} w(t_i) & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

A weight function (w, w_0) is *admissible* for a quasi-precedence $\succsim$ if $f \succsim g$ for all function symbols g whenever f is a unary function symbol with $w(f) = 0$. For a term t, $|t|$ denotes its size and $|t|_a$ for $a \in \mathcal{F} \cup \mathcal{V}$ denotes how often the symbol a occurs in t.

Definition 1 ([14,5,19]). *Let $\succsim$ be a quasi-precedence and (w, w_0) a weight function. We define the* Knuth-Bendix order $>_{\text{kbo}}$ *on terms inductively as follows: $s >_{\text{kbo}} t$ if $|s|_x \geqslant |t|_x$ for all variables $x \in \mathcal{V}$ and either*

(a) $w(s) > w(t)$, or
(b) $w(s) = w(t)$ and one of the following alternatives holds:
  (1) $t \in \mathcal{V}$, $s \in \mathcal{T}(\mathcal{F}^{(1)}, \{t\})$, and $s \neq t$, or
  (2) $s = f(s_1, \ldots, s_n)$, $t = g(t_1, \ldots, t_m)$, $f \sim g$, and there exists an $1 \leqslant i \leqslant \min\{n, m\}$ such that $s_i >_{\text{kbo}} t_i$ and $s_j = t_j$ for all $1 \leqslant j < i$, or
  (3) $s = f(s_1, \ldots, s_n)$, $t = g(t_1, \ldots, t_m)$, and $f \succ g$.

where $\mathcal{F}^{(n)}$ denotes the set of all function symbols $f \in \mathcal{F}$ of arity n. Thus in case (b)(1) the term s consists of a nonempty sequence of unary function symbols applied to the variable t.

Specializing the above definition to (the reflexive closure of) a strict precedence, one obtains the definition of KBO in [2], except that we restrict weight functions to have range $\mathbb{N}$ instead of $\mathbb{R}$. According to [15] this does not decrease the power of the order.

Lemma 2. *A TRS $\mathcal{R}$ is terminating whenever there exist a quasi-precedence $\succsim$ and a weight function (w, w_0) such that $\mathcal{R} \subseteq >_{\mathrm{kbo}}$.* □

Example 3. The TRS SK_90.2.42[2] consisting of the rules

$$\begin{aligned}
\mathsf{flatten}(\mathsf{nil}) &\to \mathsf{nil} & \mathsf{rev}(\mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{flatten}(\mathsf{unit}(x)) &\to \mathsf{flatten}(x) & \mathsf{rev}(\mathsf{unit}(x)) &\to \mathsf{unit}(x)\\
\mathsf{flatten}(x +\!\!+ y) &\to \mathsf{flatten}(x) +\!\!+ \mathsf{flatten}(y) & \mathsf{rev}(x +\!\!+ y) &\to \mathsf{rev}(y) +\!\!+ \mathsf{rev}(x)\\
\mathsf{flatten}(\mathsf{unit}(x) +\!\!+ y) &\to \mathsf{flatten}(x) +\!\!+ \mathsf{flatten}(y) & \mathsf{rev}(\mathsf{rev}(x)) &\to x\\
\mathsf{flatten}(\mathsf{flatten}(x)) &\to \mathsf{flatten}(x) & (x +\!\!+ y) +\!\!+ z &\to x +\!\!+ (y +\!\!+ z)\\
x +\!\!+ \mathsf{nil} &\to x & \mathsf{nil} +\!\!+ y &\to y
\end{aligned}$$

is KBO terminating. The weight function (w, w_0) with $w(\mathsf{flatten}) = w(\mathsf{rev}) = w(+\!\!+) = 0$ and $w(\mathsf{unit}) = w(\mathsf{nil}) = w_0 = 1$ together with the quasi-precedence $\mathsf{flatten} \sim \mathsf{rev} \succ \mathsf{unit} \succ +\!\!+ \succ \mathsf{nil}$ ensures that $l >_{\mathrm{kbo}} r$ for all rules $l \to r$. The use of a quasi-precedence is essential here; the rules $\mathsf{flatten}(x +\!\!+ y) \to \mathsf{flatten}(x) +\!\!+ \mathsf{flatten}(y)$ and $\mathsf{rev}(x +\!\!+ y) \to \mathsf{rev}(y) +\!\!+ \mathsf{rev}(x)$ demand $w(\mathsf{flatten}) = w(\mathsf{rev}) = 0$ but KBO with strict precedence does not allow different unary functions to have weight zero.

One can imagine a more general definition of KBO. For instance, in case (b)(2) we could demand that $s_j \sim_{\mathrm{kbo}} t_j$ for all $1 \leqslant j < i$ where $s \sim_{\mathrm{kbo}} t$ if and only if $s \sim t$ and $w(s) = w(t)$. Here $s \sim t$ denotes syntactic equality with respect to equivalent function symbols of the same arity. Another obvious extension would be to compare the arguments according to an arbitrary permutation or as multisets. To keep the discussion and implementation simple, we do not consider such refinements in the sequel.

3 A Pure SAT Encoding of KBO

In order to give a propositional encoding of KBO termination, we must take care of representing a precedence and a weight function. For the former we introduce two sets of new variables $X = \{X_{fg} \mid f, g \in \mathcal{F} \text{ with } f \neq g\}$ and $Y = \{Y_{fg} \mid f, g \in \mathcal{F} \text{ with } f \neq g\}$ depending on the underlying signature $\mathcal{F}$ ([16,21]). The intended semantics of these variables is that an assignment which satisfies a variable X_{fg} corresponds to a precedence with $f \succ g$ and similarly Y_{fg} suggests $f \sim g$. When dealing with strict precedences it is safe to assign all Y_{fg} variables to false. For the weight function, symbols are considered in binary representation and the operations $>$, $=$, $\geqslant$, and $+$ must be redefined accordingly. The propositional encodings of $>$ and $=$ given below are similar to the ones in [3]. To save parentheses we employ the binding hierarchy for the connectives where $+$ binds strongest, followed by the relation symbols $>$, $=$, and $\geqslant$. The logical connectives $\vee$ and $\wedge$ are next in the hierarchy and $\to$ and $\leftrightarrow$ bind weakest.

[2] Labels in sans-serif font refer to TRSs in the Termination Problems Data Base [18].

We fix the number k of bits that is available for representing natural numbers in binary. Let $a < 2^k$. We denote by $\mathbf{a} = \langle a_k, \ldots, a_1 \rangle$ the binary representation of a where a_k is the most significant bit.

Definition 4. *For natural numbers given in binary representation, the operations $>$, $=$, and $\geqslant$ are defined as follows (for all $1 \leqslant j \leqslant k$):*

$$\begin{aligned}
\mathbf{f} >_j \mathbf{g} &= \begin{cases} f_1 \wedge \neg g_1 & \text{if } j = 1 \\ (f_j \wedge \neg g_j) \vee \big((f_j \leftrightarrow g_j) \wedge \mathbf{f} >_{j-1} \mathbf{g}\big) & \text{if } j > 1 \end{cases} \\
\mathbf{f} > \mathbf{g} &= \mathbf{f} >_k \mathbf{g} \\
\mathbf{f} = \mathbf{g} &= \bigwedge_{i=1}^{k} (f_i \leftrightarrow g_i) \\
\mathbf{f} \geqslant \mathbf{g} &= \mathbf{f} > \mathbf{g} \vee \mathbf{f} = \mathbf{g}
\end{aligned}$$

Next we define a formula which is satisfiable if and only if the encoded weight function is admissible for the encoded precedence.

Definition 5. *For a weight function (w, w_0), let $\mathsf{ADM\text{-}SAT}(w, w_0)$ be the formula*

$$\mathbf{w_0} > \mathbf{0} \;\wedge \bigwedge_{c \in \mathcal{F}^{(0)}} \mathbf{c} \geqslant \mathbf{w_0} \;\wedge \bigwedge_{f \in \mathcal{F}^{(1)}} \big(\mathbf{f} = \mathbf{0} \;\rightarrow \bigwedge_{g \in \mathcal{F},\, f \neq g} (X_{fg} \vee Y_{fg})\big)$$

For addition we use pairs. The first component represents the bit representation and the second component is a propositional formula which encodes the constraints for each digit.

Definition 6. *We define $(\mathbf{f}, \varphi) + (\mathbf{g}, \psi)$ as $(\mathbf{s}, \varphi \wedge \psi \wedge \gamma \wedge \sigma)$ with*

$$\gamma \;=\; \neg c_k \wedge \neg c_0 \wedge \bigwedge_{i=1}^{k} \big(c_i \leftrightarrow ((f_i \wedge g_i) \vee (f_i \wedge c_{i-1}) \vee (g_i \wedge c_{i-1}))\big)$$

and

$$\sigma \;=\; \bigwedge_{i=1}^{k} \big(s_i \leftrightarrow (f_i \oplus g_i \oplus c_{i-1})\big)$$

where c_i $(0 \leqslant i \leqslant k)$ and s_i $(1 \leqslant i \leqslant k)$ are fresh variables that represent the carry and the sum of the addition and $\oplus$ denotes exclusive or. The condition $\neg c_k$ prevents a possible overflow.

Note that although theoretically not necessary, it is a good idea to introduce new variables for the sum. The reason is that in consecutive additions each bit f_i and g_i is duplicated (twice for the carry and once for the sum) and consequently using fresh variables for the sum prevents an exponential blowup of the resulting formula.

Definition 7. *We define* $(\mathbf{f},\varphi) > (\mathbf{g},\psi)$ *as* $\mathbf{f} > \mathbf{g} \wedge \varphi \wedge \psi$ *and* $(\mathbf{f},\varphi) = (\mathbf{g},\psi)$ *as* $\mathbf{f} = \mathbf{g} \wedge \varphi \wedge \psi$.

In the next definition we show how the weight of terms is computed propositionally.

Definition 8. *Let* t *be a term and* (w, w_0) *a weight function. The weight of a term is encoded as follows:*

$$W_t \;=\; \begin{cases} (\mathbf{w_0}, \top) & \text{if } t \in \mathcal{V}, \\ (\mathbf{f}, \top) + \sum_{i=1}^{n} W_{t_i} & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

We are now ready to define a propositional formula that reflects the definition of $>_{\mathrm{kbo}}$.

Definition 9. *Let* s *and* t *be terms. We define the formula* $\mathsf{SAT}(s >_{\mathrm{kbo}} t)$ *as follows. If* $s \in \mathcal{V}$ *or* $s = t$ *or* $|s|_x < |t|_x$ *for some* $x \in \mathcal{V}$ *then* $\mathsf{SAT}(s >_{\mathrm{kbo}} t) = \bot$. *Otherwise*

$$\mathsf{SAT}(s >_{\mathrm{kbo}} t) \;=\; W_s > W_t \vee \big(W_s = W_t \wedge \mathsf{SAT}(s >'_{\mathrm{kbo}} t)\big)$$

with

$$\mathsf{SAT}(s >'_{\mathrm{kbo}} t) = \begin{cases} \top & \text{if } t \in \mathcal{V},\ s \in \mathcal{T}(\mathcal{F}^{(1)}, \{t\}), \text{ and } s \neq t \\ \mathsf{SAT}(s_i >_{\mathrm{kbo}} t_i) & \text{if } s = f(s_1, \ldots, s_n), t = f(t_1, \ldots, t_n) \\ X_{fg} \vee \big(Y_{fg} \wedge \mathsf{SAT}(s_i >_{\mathrm{kbo}} t_i)\big) & \\ & \text{if } s = f(s_1, \ldots, s_n),\ t = g(t_1, \ldots, t_m), \text{ and } f \neq g \end{cases}$$

where in the second (third) clause i *denotes the least* $1 \leqslant j \leqslant n$ *(*$\min\{n, m\}$*) with* $s_j \neq t_j$.

3.1 Encoding the Precedence in SAT

To ensure the properties of a precedence we follow the approach of Codish *et al.* [3] who propose to interpret function symbols as natural numbers. The greater than or equal to relation then ensures that the function symbols are quasi-ordered. Let $|\mathcal{F}| = n$. We are looking for a mapping $m\colon \mathcal{F} \to \{1, \ldots, n\}$ such that for every propositional variable $X_{fg} \in X$ we have $m(f) > m(g)$ and for $Y_{fg} \in Y$ we get $m(f) = m(g)$. To uniquely encode one of the n function symbols, $l := \lceil log_2(n) \rceil$ fresh propositional variables are needed. The l-bit representation of f is $\langle f'_l, \ldots, f'_1 \rangle$ with f'_l the most significant bit.

Definition 10. *For all* $1 \leqslant j \leqslant l$

$$||X_{fg}||_j \;=\; \begin{cases} f'_1 \wedge \neg g'_1 & \text{if } j = 1 \\ (f'_j \wedge \neg g'_j) \vee \big((f'_j \leftrightarrow g'_j) \wedge ||X_{fg}||_{j-1}\big) & \text{if } j > 1 \end{cases}$$

$$||Y_{fg}||_l \;=\; \bigwedge_{j=1}^{l} (f'_j \leftrightarrow g'_j)$$

Note that the variables f'_i $(1 \leqslant i \leqslant l)$ are different from f_i $(1 \leqslant i \leqslant k)$ which are used to represent weights.

Definition 11. *Let $\mathcal{R}$ be a TRS. The formula* $\mathsf{KBO\text{-}SAT}(\mathcal{R})$ *is defined as*

$$\mathsf{ADM\text{-}SAT}(w, w_0) \;\wedge \bigwedge_{l \to r \in \mathcal{R}} \mathsf{SAT}(l >_{\mathrm{kbo}} r) \;\wedge \bigwedge_{z \in X \cup Y} (z \leftrightarrow ||z||_l)$$

Theorem 12. *A TRS $\mathcal{R}$ is terminating whenever the propositional formula* $\mathsf{KBO\text{-}SAT}(\mathcal{R})$ *is satisfiable.* □

The reverse does not hold (Example 21).

3.2 Optimizations

This section deals with logical simplifications concerning propositional formulas as well as optimizations which are specific for the generation of the constraint formula which encodes KBO termination of the given instance.

Logical Optimizations Since the constraint formula contains many occurrences of $\top$ and $\bot$ logical equivalences simplifying such formulas are employed.

SAT solvers typically expect their input in conjunctive normal form (CNF) but for the majority of the TRSs the constraint formula $\mathsf{KBO\text{-}SAT}(\mathcal{R})$ is too large for the standard translation. The problem is that the resulting CNF may be exponentially larger than the input formula because when distributing $\vee$ over $\wedge$ subformulas get duplicated. In [20] Tseitin proposed a transformation which is linear in the size of the input formula. The price for linearity is paid with introducing new variables. As a consequence, Tseitin's transformation does not produce an equivalent formula, but it does preserve and reflect satisfiability.

Optimizations Concerning the Encoding Before discussing the implemented optimizations in detail it is worth mentioning the bottleneck of the whole procedure. As addressed in the previous section, SAT solvers expect their input in CNF. It turned out that the generation of all non-atomic subformulas, which are needed for the translation, constitutes the main bottleneck. So every change in the implementation which reduces the size of the constraint formula will result in an additional speedup. All improvements discussed in the sequel could reduce the execution time at least a bit. Whenever they are essential it is explicitly stated.

Since $>_{\mathrm{kbo}}$ is a simplification order it contains the embedding relation. We make use of that fact by only computing the constraint formula $s >_{\mathrm{kbo}} t$ if the test $s \rhd_{\mathsf{emb}} t$ is false. Most of the other optimizations deal with representing or computing the weight function. When computing the constraints for the weights in a rule $l \to r$, removing function symbols and variables that occur both in l and in r is highly recommended or even necessary for an efficient implementation. The benefit can be seen in the example below. Note that propositional addition

is somehow expensive as new variables have to be added for representing the carry and the sum in addition to a formula which encodes the constraints for each digit.

Example 13. Consider the TRS consisting of the single rule $\mathsf{f}(y, \mathsf{g}(x), x) \to \mathsf{f}(y, x, \mathsf{g}(\mathsf{g}(x)))$. Without the optimization the constraints for the weights would amount to

$$\begin{gathered}(\mathbf{f}, \top) + (\mathbf{w_0}, \top) + (\mathbf{g}, \top) + (\mathbf{w_0}, \top) + (\mathbf{w_0}, \top) \\ \geqslant \\ (\mathbf{f}, \top) + (\mathbf{w_0}, \top) + (\mathbf{w_0}, \top) + (\mathbf{g}, \top) + (\mathbf{g}, \top) + (\mathbf{w_0}, \top)\end{gathered}$$

whereas employing the optimization produces the more or less trivial constraint $(\mathbf{0}, \top) \geqslant (\mathbf{g}, \top)$.

By using a cache for propositional addition we can test if we already computed the sum of the weights of two function symbols f and g. That reduces the number of newly introduced variables and sometimes we can omit the constraint formula for addition. This is clarified in the following example.

Example 14. Consider the TRS consisting of the rules $\mathsf{f}(\mathsf{a}) \to \mathsf{b}$ and $\mathsf{f}(\mathsf{a}) \to \mathsf{c}$. The constraints for the first rule amount to the following formula where $\underline{\mathbf{fa}}$ corresponds to the new variables which are required for the sum when adding f and a and the propositional formula φ represents the constraints which are put on each digit of $\underline{\mathbf{fa}}$:

$$\begin{aligned}\mathsf{SAT}(\mathsf{f}(\mathsf{a}) >_{\mathrm{kbo}} \mathsf{b}) &= W_{\mathsf{f(a)}} > W_{\mathsf{b}} \vee \big(W_{\mathsf{f(a)}} = W_{\mathsf{b}} \wedge X_{\mathsf{fb}}\big) \\ &= (\mathbf{f}, \top) + (\mathbf{a}, \top) > (\mathbf{b}, \top) \vee \big((\mathbf{f}, \top) + (\mathbf{a}, \top) = (\mathbf{b}, \top) \wedge X_{\mathsf{fb}}\big) \\ &= (\underline{\mathbf{fa}}, \varphi) > (\mathbf{b}, \top) \vee \big((\underline{\mathbf{fa}}, \varphi) = (\mathbf{b}, \top) \wedge X_{\mathsf{fb}}\big) \\ &= (\underline{\mathbf{fa}} > \mathbf{b} \wedge \varphi) \vee (\underline{\mathbf{fa}} = \mathbf{b} \wedge \varphi \wedge X_{\mathsf{fb}})\end{aligned}$$

We get a similar formula for the second rule and the conjunction of both amounts to

$$\big((\underline{\mathbf{fa}} > \mathbf{b} \wedge \varphi) \vee (\underline{\mathbf{fa}} = \mathbf{b} \wedge \varphi \wedge X_{\mathsf{fb}})\big) \wedge \big((\underline{\mathbf{fa}} > \mathbf{c} \wedge \varphi) \vee (\underline{\mathbf{fa}} = \mathbf{c} \wedge \varphi \wedge X_{\mathsf{fc}})\big)$$

Using commutativity and distributivity we could obtain the equivalent formula

$$\big(\underline{\mathbf{fa}} > \mathbf{b} \vee (\underline{\mathbf{fa}} = \mathbf{b} \wedge X_{\mathsf{fb}})\big) \wedge \big(\underline{\mathbf{fa}} > \mathbf{c} \vee (\underline{\mathbf{fa}} = \mathbf{c} \wedge X_{\mathsf{fc}})\big) \wedge \varphi$$

which gives rise to fewer subformulas. Note that this simplification can easily be implemented using the information of the cache for addition.

4 A Pseudo-Boolean Encoding of KBO

A *pseudo-boolean constraint* (PBC) is of the form

$$\big(\sum_{i=1}^{n} a_i * x_i\big) \circ m$$

where $a_1, \ldots, a_n, m$ are fixed integers, $x_1, \ldots, x_n$ boolean variables that range over $\{0, 1\}$, and $\circ \in \{\geqslant, =, \leqslant\}$. We separate PBCs that are written on a single line by semicolons. A sequence of PBCs is satisfiable if there exists an assignment which satisfies every PBC in the sequence. Since 2005 pseudo-boolean evaluation [17] is a track of the international SAT competition.[3] In the sequel we show how to encode KBO using PBCs.

Definition 15. *For a weight function (w, w_0) let $\mathsf{ADM\text{-}PBC}(w, w_0)$ be the collection of PBCs*

- $\overline{w}_0 \geqslant 1$
- $\overline{w}(c) - \overline{w}_0 \geqslant 0$ *for all* $c \in \mathcal{F}^{(0)}$
- $(n-1) * \overline{w}(f) + \sum_{f \neq g} (X_{fg} + Y_{fg}) \geqslant (n-1)$ *for all* $f \in \mathcal{F}^{(1)}$

where $n = |\mathcal{F}|$, $\overline{w}(f) = 2^{k-1} * f_k + \cdots + 2^0 * f_1$ denotes the weight of f in $\mathbb{N}$ using k bits, and $\overline{w}_0$ denotes the value of $\mathbf{w_0}$.

In the definition above the first two PBCs express that w_0 is strictly larger than zero and that every unary function symbol has weight at least w_0. Whenever the considered function symbol f has weight larger than zero the third constraint is trivially satisfied. In the case that the unary function symbol f has weight zero the constraints on the precedence add up to $n-1$ if and only if f is a maximal element. Note that X_{fg} and Y_{fg} are mutual exclusive (which is ensured when encoding the constraints on a quasi-precedence, cf. Definition 18).

For the encoding of $s >_{\text{kbo}} t$ and $s >'_{\text{kbo}} t$ auxiliary propositional variables $KBO_{s,t}$ and $KBO'_{s,t}$ are introduced. The intended meaning is that if $s >_{\text{kbo}} t$ ($s >'_{\text{kbo}} t$) then $KBO_{s,t}$ ($KBO'_{s,t}$) evaluates to true under a satisfying assignment. The general idea of the encoding is very similar to the pure SAT case. As we do not know anything about weights and the precedence at the time of encoding we have to consider the cases $w(s) > w(t)$ and $w(s) = w(t)$ at the same time. That is why $KBO'_{s,t}$ and the recursive call to $\mathsf{PBC}(s >'_{\text{kbo}} t)$ must be considered in any case.

The weight $w(t)$ of a term t is defined similarly as in Section 2 with the only difference that the weight $w(f)$ of the function symbol $f \in \mathcal{F}$ is represented in k bits as described in Definition 15.

Definition 16. *Let s and t be terms. The encoding of $\mathsf{PBC}(s >_{\text{kbo}} t)$ amounts to $KBO_{s,t} = 0$ if $s \in \mathcal{V}$ or $s = t$ or $|s|_x < |t|_x$ for some $x \in \mathcal{V}$. In all other cases $\mathsf{PBC}(s >_{\text{kbo}} t)$ is*

$$-(m+1) * KBO_{s,t} + w(s) - w(t) + KBO'_{s,t} \geqslant -m;\ \mathsf{PBC}(s >'_{\text{kbo}} t)$$

where $m = 2^k * |t|$. Here $\mathsf{PBC}(s >'_{\text{kbo}} t)$ is the empty constraint when $t \in \mathcal{V}$, $s \in \mathcal{T}(\mathcal{F}^{(1)}, \{t\})$, and $s \neq t$. In the remaining case $s = f(s_1, \ldots, s_n)$, $t =$

[3] `http://sat07.ecs.soton.ac.uk`

$g(t_1, \ldots, t_m)$, *and* $\mathsf{PBC}(s >'_{\mathrm{kbo}} t)$ *is the combination of* $\mathsf{PBC}(s_i >_{\mathrm{kbo}} t_i)$ *and*

$$\begin{cases} -KBO'_{s,t} + KBO_{s_i,t_i} \geqslant 0 & \text{if } f = g \\ -2 * KBO'_{s,t} + 2 * X_{fg} + Y_{fg} + KBO_{s_i,t_i} \geqslant 0 & \text{if } f \neq g \end{cases}$$

where i *denotes the least* $1 \leqslant j \leqslant \min\{n, m\}$ *with* $s_i \neq t_i$.

Since the encoding of $\mathsf{PBC}(s >_{\mathrm{kbo}} t)$ is explained in the example below here we just explain the intended semantics of $\mathsf{PBC}(s >'_{\mathrm{kbo}} t)$. In the first case where t is a variable there are no constraints on the weights and the precedence which means that the empty constraint is returned. In the case where s and t have identical root symbols it is demanded that whenever $KBO'_{s,t}$ holds then also KBO_{s_i,t_i} must be satisfied before going into the recursion. In the last case s and t have different root symbols and the PBC expresses that whenever $KBO'_{s,t}$ is satisfied then either $f > g$ or both $f \sim g$ and KBO_{s_i,t_i} must hold.

To get familiar with the encoding and to see why the definitions are a bit tricky consider the example below. For reasons of readability symbols occurring both in s and in t are removed immediately. This entails that the multiplication factor m should be lowered to

$$m = \sum_{a \in \mathcal{F} \cup \mathcal{V}} max\{0, 2^k * (|t|_a - |s|_a)\},$$

which again is a lower bound of the left-hand side of the constraint if $KBO_{s,t}$ is false.

Example 17. Consider the TRS consisting of the rule

$$s = \mathsf{f}(\mathsf{g}(x), \mathsf{g}(\mathsf{g}(x))) \to \mathsf{f}(\mathsf{g}(\mathsf{g}(x)), x) = t$$

The PB encoding $\mathsf{PBC}(s >_{\mathrm{kbo}} t)$ then looks as follows:

$$-KBO_{s,t} + w(g) + KBO'_{s,t} \geqslant 0 \tag{1}$$
$$-KBO'_{s,t} + KBO_{\mathsf{g}(x),\mathsf{g}(\mathsf{g}(x))} \geqslant 0 \tag{2}$$
$$-(2^k + 1) * KBO_{\mathsf{g}(x),\mathsf{g}(\mathsf{g}(x))} - w(g) + KBO'_{\mathsf{g}(x),\mathsf{g}(\mathsf{g}(x))} \geqslant -2^k \tag{3}$$
$$KBO'_{\mathsf{g}(x),\mathsf{g}(\mathsf{g}(x))} + KBO_{x,\mathsf{g}(x)} \geqslant 0 \tag{4}$$
$$KBO_{x,\mathsf{g}(x)} = 0 \tag{5}$$

Constraint (1) states that if $s >_{\mathrm{kbo}} t$ then either $w(g) > 0$ or $s >'_{\mathrm{kbo}} t$. Clearly the attentive reader would assign $w(g) = 1$ and termination of the TRS is shown. The encoding however is not so smart and performs the full recursive translation to PB. In (3) it is not possible to satisfy $s_1 = \mathsf{g}(x) >_{\mathrm{kbo}} \mathsf{g}(\mathsf{g}(x)) = t_1$ since the former is embedded in the latter. Nevertheless the constraint (3) must remain satisfiable because the TRS is KBO terminating. The trick is to introduce a hidden case distinction. The multiplication factor in front of the KBO_{s_1,t_1} variable does that job. Whenever $s_1 >_{\mathrm{kbo}} t_1$ is needed then KBO_{s_1,t_1} must

evaluate to true. Then implicitly the constraint demands that $w(s_1) > w(t_1)$ or $w(s_1) = w(t_1)$ and $s_1 >'_{\text{kbo}} t_1$ which reflects the definition of KBO. If $s_1 >_{\text{kbo}} t_1$ need not be satisfied (e.g., because already $s >_{\text{kbo}} t$ in (1)) then the constraint holds in any case since the left hand side in (3) never becomes smaller than -2^k.

4.1 Encoding the Precedence in PBCs

To encode a precedence in PB we again interpret function symbols in $\mathbb{N}$. For this approach an additional set of propositional variables $Z = \{Z_{fg} \mid f, g \in \mathcal{F} \text{ with } f \neq g\}$ is used. The intended semantics is that Z_{fg} evaluates to true whenever $g \succ f$ or f and g are incomparable. Just note that the Z_{fg} variables are not necessary as far as termination proving power is considered but they are essential to encode partial precedences which are sometimes handy (as explained in Section 6).

Definition 18. *For a signature $\mathcal{F}$ we define* PREC-PBC($\mathcal{F}$) *using the PBCs below. Let $l = \lceil log_2(|\mathcal{F}|)\rceil$. For all $f, g \in \mathcal{F}$ with $f \neq g$*

$$\begin{aligned} 2 * X_{fg} + Y_{fg} + Y_{gf} + 2 * Z_{fg} &= 2 \\ -X_{fg} + 2^l * Y_{fg} + 2^l * Z_{fg} + i(f) - i(g) &\geqslant 0 \\ 2^l * X_{fg} + Y_{fg} + 2^l * Z_{fg} + i(f) - i(g) &\geqslant 1 \end{aligned}$$

where $i(f) = 2^{l-1} * f'_l + \cdots + 2^0 * f'_1$ denotes the interpretation of f in $\mathbb{N}$ using l bits.

The above definition expresses all requirements of a quasi-precedence. The symmetry of $\sim$ and the mutual exclusion of the X, Y, and Z variables is mimicked by the first constraint. The second constraint encodes the conditions that are put on the X variables. Whenever a system needs $f > g$ in the precedence to be terminating then X_{fg} must evaluate to true and (because they are mutually exclusive) Y_{fg} and Z_{fg} to false. Hence in order to remain satisfiable $i(f) > i(g)$ must hold. In a case where $f > g$ is not needed (but the TRS is KBO terminating) the constraint must remain satisfiable. Thus Y_{fg} or Z_{fg} evaluate to one and because $i(g)$ is bound by $2^l - 1$ the constraint does no harm. Summing up, the second constraint encodes a proper order on the symbols in $\mathcal{F}$. The third constraint forms an equivalence relation on $\mathcal{F}$ using the Y_{fg} variables. Whenever $f \sim g$ is demanded somehow in the encoding, then X_{fg} and Z_{fg} evaluate to false by the first constraint. Satisfiability of the third constraint implies $i(f) \geqslant i(g)$ but at the same time symmetry demands that Y_{gf} also evaluates to true which leads to $i(g) \geqslant i(f)$ and thus to $i(f) = i(g)$.

Definition 19. *Let $\mathcal{R}$ be a TRS. The pseudo-boolean encoding* KBO-PBC($\mathcal{R}$) *is defined as the combination of* ADM-PBC(w, w_0), PREC-PBC($\mathcal{F}$), *and*

$$\text{PBC}(l >_{\text{kbo}} r);\ KBO_{l,r} = 1$$

for all $l \to r \in \mathcal{R}$.

Theorem 20. *A TRS $\mathcal{R}$ is terminating whenever the PBCs* KBO-PBC($\mathcal{R}$) *are satisfiable.* □

Again the reverse does not hold (Example 21).

5 Experimental Results

We implemented our encodings on top of T$_\top$T [12]. MiniSat and MiniSat+ [7,8] were used to check satisfiability of the SAT and PBC based encodings. Below we compare our implementations of KBO, sat and pbc, with the ones of T$_\top$T and AProVE [11]. T$_\top$T admits only strict precedences, AProVE also quasi-precedences. Both implement the polynomial time algorithm of Korovin and Voronkov [15] together with techniques of Dick *et al.* [5].

We used the 865 TRSs which do not specify any strategy or theory and the 322 string rewrite systems (SRSs) in version 3.2 of the Termination Problem Data Base [18]. All tests were performed on a server equipped with an Intel® Xeon™ processor running at a CPU rate of 2.40 GHz and 512 MB of system memory with a timeout of 60 seconds.

5.1 Results for TRSs

As addressed in Section 3 one has to fix the number k of bits which is used to represent natural numbers in binary representation. The actual choice is specified as argument to sat (pbc). Note that a rather small k is sufficient to handle all systems from [18] which makes Theorems 12 and 20 powerful in practice. The example below gives evidence that there does not exist a general upper bound on k.

Example 21. Consider the parametrized TRS consisting of the three rules

$$\mathsf{f}(\mathsf{g}(x,y)) \to \mathsf{g}(\mathsf{f}(x),\mathsf{f}(y)) \qquad \mathsf{h}(x) \to \mathsf{f}(\mathsf{f}(x)) \qquad \mathsf{i}(x) \to \mathsf{h}^n(x)$$

with $n = 2^k$. Since the first rule duplicates the function symbol f we must assign weight zero to it. The admissibility condition for the weight function demands that f is a maximal element in the precedence. The second rule excludes the case $\mathsf{h} \sim \mathsf{f}$ and demands that the weight of h is strictly larger than zero. It follows that the minimum weight of $\mathsf{h}^n(x)$ is $n + 1 = 2^k + 1$, which at the same time is the minimum weight of $\mathsf{i}(x)$. Thus $w(\mathsf{i})$ is at least 2^k which requires $k + 1$ bits.

The left part of Table 1 summarizes[4] the results for strict precedences. Since AProVE produced seriously slower results than T$_\top$T in the TRS category, it is not considered in Table 1. Interestingly, with $k = 4$ equally many TRSs can be proved terminating as with $k = 10$. The TRS higher-order_AProVE_HO_ReverseLastInit

[4] The experiments are described in more detail at `http://cl-informatik.uibk.ac.at/~hzankl/kbo`.

	strict precedence			quasi-precedence		
method(#bits)	total time	#successes	#timeouts	total time	#successes	#timeouts
sat/pbc(2)	19.2/16.4	72/76	0/0	20.9/16.8	73/77	0/0
sat/pbc(3)	20.2/16.3	77/77	0/0	21.9/16.9	78/78	0/0
sat/pbc(4)	21.9/16.1	78/78	0/0	22.8/17.0	79/79	0/0
sat/pbc(10)	86.1/16.7	78/78	1/0	90.2/17.2	79/79	1/0
TTT	169.5	77	1			

Table 1. KBO for 865 TRSs.

needs weight eight for the constant init and therefore can only be proved KBO terminating with $k \geqslant 4$.

Concerning the optimizations in Section 3.2, if we use the standard (exponential) transformation to CNF, the total time required increases to 2681.06 seconds, the number of successful termination proofs decreases to 69, and 45 timeouts occur (for $k = 4$). Furthermore, if we don't use a cache for adding weights and equal symbols are not removed when the weights of left and right-hand sides of rules are compared, the number of successful termination proofs remains the same but the total time increases to 92.30 seconds and one timeout occurs.

TTT without timeout requires 4747.65 seconds and can prove KBO termination of 78 TRSs. The lion's share is taken up by various_21 with 4016.23 seconds for a positive result. sat(4) needs only 0.10 seconds for this TRS and pbc(4) even only 0.03 seconds. Since TTT employs the slightly stronger KBO definition of [15] it can prove one TRS (various_27) terminating which cannot be handled by sat and pbc. On the other hand TTT gives up on HM_t000 which specifies addition for natural numbers in decimal notation (using 104 rewrite rules). The problem is not the timeout but at some point the algorithm detects that it will require too many resources. To prevent a likely stack overflow from occurring, the computation is terminated and a "don't know" result is reported. (AProVE behaves in a similar fashion on this TRS.) Also for our approaches this system is the most challenging one with 0.54 (sat(4)) and 0.11 (pbc(4)) seconds.

As can be seen from the right part of Table 1, by admitting quasi-precedences one additional TRS (SK_90.2.42, Example 3) can be proved KBO terminating. Surprisingly, AProVE 1.2 cannot prove (quasi) KBO termination of this system, for unknown reasons.

5.2 Results for SRSs

For SRSs we have similar results, as can be inferred from Table 2. The main difference is the larger number of bits needed for the propositional addition of the weights. The maximum number of SRSs is proved KBO terminating with $k \geqslant 7$ in case of sat and $k \geqslant 6$ for pbc. The reason is that in the first implementation the number of bits does not increase for intermediate sums when adding the weights. Generally speaking TTT performs better on SRSs than on TRSs concerning KBO because it can handle all systems within 546.43 seconds. The instance which

	strict precedence			quasi-precedence		
method(#bits)	total time	#successes	#timeouts	total time	#successes	#timeouts
sat/pbc(2)	9.1/5.9	8/19	0/0	13.9/6.2	8/19	0/0
sat/pbc(3)	12.1/5.9	17/24	0/0	16.9/6.4	17/24	0/0
sat/pbc(4)	15.1/6.0	24/30	0/0	20.0/6.5	24/30	0/0
sat/pbc(6)	15.8/6.1	31/33	0/0	27.4/6.7	31/33	0/0
sat/pbc(7)	17.0/6.1	33/33	0/0	31.2/6.7	33/33	0/0
sat/pbc(10)	21.6/6.3	33/33	0/0	98.8/6.9	32/33	1/0
T$_\mathsf{T}$T	72.4	29	1			

Table 2. KBO for 322 SRSs.

consumes the most time is Zantema_z112 with 449.01 seconds for a positive answer; sat(7) needs just 0.11 and pbc(7) 0.03 seconds. With a timeout of 60 seconds T$_\mathsf{T}$T proves KBO termination of 29 SRSs, without any timeout one more. Our implementations both prove KBO termination of 33 SRSs. The three SRSs that make up the difference (Trafo_dup11, Zantema_z069, Zantema_z070) derive from algebra (polyhedral groups). T$_\mathsf{T}$T and AProVE give up on these SRSs for the same reasons as mentioned in the preceding subsection for HM_t000.

Admitting quasi-precedences does not allow to prove KBO termination of more SRSs. On the contrary, a timeout occurs when using sat(10) on Trafo_dup11 whereas pbc(10) easily handles the system.

6 Assessment

In this section we compare the two approaches presented in this paper. Let us start with the most important measurements: power and run time. Here pbc is the clear winner. Not only is it faster on any kind of precedence; it also scales much better for larger numbers of bits used to represent the weights. Furthermore, the pseudo-boolean approach is less implementation work since additions are performed by the SAT solver and also the transformation to CNF is not necessary. We note that the implementation of pbc is exactly as described in the paper whereas sat integrates the optimizations described in Section 3.2.

A further advantage of the pseudo-boolean approach is the option of a *goal function* which should be minimized while preserving satisfiability of the constraints. Although the usage of such a goal function is not of computational interest it is useful for generating easily human readable proofs. We experimented with functions minimizing the weights for function symbols and reducing the comparisons in the precedence. The former has the advantage that one obtains a KBO proof with minimal weights which is nicely illustrated on the SRS Zantema_z113 consisting of the rules

$$\begin{array}{lll} 11 \to 43 & 33 \to 56 & 55 \to 62 \\ 12 \to 21 & 22 \to 111 & 34 \to 11 \\ 44 \to 3 & 56 \to 12 & 66 \to 21 \end{array}$$

TTT and AProVE produce the proof

$$w(1) = 32471712256 \qquad w(2) = 48725750528 \qquad w(3) = 43247130624$$
$$w(4) = 21696293888 \qquad w(5) = 44731872512 \qquad w(6) = 40598731520$$
$$3 > 1 > 2 \qquad\qquad 1 > 4$$

whereas pbc(6) produces

$$w(1) = 31 \qquad w(2) = 47 \qquad w(3) = 41$$
$$w(4) = 21 \qquad w(5) = 43 \qquad w(6) = 39$$
$$3 > 1 > 2 \qquad 3 > 5 > 6 > 2 \qquad 1 > 4$$

Regarding the goal function dealing with the minimization of comparisons in the precedence we detected that using two (three, four five, ten) bits to encode weights of function symbols 39 (45, 46, 47, 47) TRSs can be proved terminating in 16.7 (16.8, 17.0, 16.7, 16.9) seconds with empty precedence.

While running the experiments, sat and pbc produced different answers for the SRS Zantema_z13; pbc claimed KBO termination whereas sat answered "don't know". Chasing that discrepancy revealed a bug [6] in MiniSat+ (which has been corrected in the meantime).

An interesting (and probably computationally fast) extension will be the integration of the pseudo-boolean encoding of KBO into a dependency pair [1] setting [4,22]. Clearly the constraints will get more involved but we expect that the generalization to non-linear constraints in the input format for the PB track of the SAT 2007 competition will ease the work considerably.

7 Summary

In this paper we presented two logic-based encodings of KBO—pure SAT and PBC—which can be implemented more efficiently and with considerably less effort than the methods described in [5,15]. Especially the PBC encoding gives rise to a very fast implementation even without caring about possible optimizations in the encoding.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
3. M. Codish, V. Lagoon, and P. Stuckey. Solving partial order constraints for LPO termination. In *Proc. 17th International Conference on Rewriting Techniques and Applications*, volume 4098 of *LNCS*, pages 4–18, 2006.
4. M. Codish, P. Schneider-Kamp, V. Lagoon, R. Thiemann, and J. Giesl. SAT solving for argument filterings. In *Proc. 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 4246 of *LNAI*, pages 30–44, 2006.

5. J. Dick, J. Kalmus, and U. Martin. Automating the Knuth-Bendix ordering. *Acta Infomatica*, 28:95–119, 1990.
6. N. Eén. Personal conversation, 2007. Google Group on Minisat.
7. N. Eén and N Sörensson. An extensible SAT-solver. In *Proc. 6th International Conference on Theory and Applications of Satisfiability Testing*, volume 2919 of *LNCS*, pages 502–518, 2003.
8. N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.
9. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. In *Proc. 3rd International Joint Conference on Automated Reasoning*, volume 4130 of *LNAI*, pages 574–588, 2006.
10. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proc. 10th International Conference on Theory and Applications of Satisfiability Testing*, volume 4501 of *LNCS*, pages 340–354, 2007.
11. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. 3rd International Joint Conference on Automated Reasoning*, volume 4130 of *LNAI*, pages 281–286, 2006.
12. N. Hirokawa and A. Middeldorp. Tyrolean termination tool. In *Proc. 16th International Conference on Rewriting Techniques and Applications*, volume 3467 of *LNCS*, pages 175–184, 2005.
13. D. Hofbauer and J. Waldmann. Termination of string rewriting with matrix interpretations. In *Proc. 17th International Conference on Rewriting Techniques and Applications*, volume 4098 of *LNCS*, pages 328–342, 2006.
14. D.E. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.
15. K. Korovin and A. Voronkov. Orienting rewrite rules with the Knuth-Bendix order. *Information and Computation*, 183:165–186, 2003.
16. M. Kurihara and H. Kondo. Efficient BDD encodings for partial order constraints with application to expert systems in software verification. In *Proc. 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, volume 3029 of *LNAI*, pages 827–837, 2004.
17. V. Manquinho and O. Roussel. Pseudo-boolean evaluation, 2007. `www.cril.univ-artois.fr/PB07/`.
18. C. Marché. Termination problem data base (TPDB), version 3.2, June 2006. `www.lri.fr/~marche/tpdb`.
19. J. Steinbach. Extensions and comparison of simplification orders. In *Proc. 3rd International Conference on Rewriting Techniques and Applications*, volume 355 of *LNCS*, pages 434–448, 1989.
20. G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125. 1968.
21. H. Zankl. SAT techniques for lexicographic path orders. Seminar report, 2006. Available at `http://arxiv.org/abs/cs.SC/0605021`.
22. H. Zankl, N. Hirokawa, and A. Middeldorp. Constraints for argument filterings. In *Proc. 33rd International Conference on Current Trends in Theory and Practice of Computer Science*, volume 4362 of *LNCS*, pages 579–590, 2007.
23. H. Zankl and A. Middeldorp. KBO as a satisfaction problem. In *Proc. 8th International Workshop on Termination*, pages 55–59, 2006.