KBO Constraint Solving Revisited

Yasmine Briefs, Hendrik Leidinger, Christoph Weidenbach

KBO Constraint Solving Revisited

Yasmine Briefs[1,2(✉)], Hendrik Leidinger[1,2], and Christoph Weidenbach[1]

[1] Max Planck Institute for Informatics, Saarbrücken, Germany
{ybriefs,hleiding,weidenbach}@mpi-inf.mpg.de
[2] Graduate School of Computer Science,
Saarland Informatics Campus, Saarbrücken, Germany

Abstract. KBO constraint solving is very well-known to be an NP-complete problem. Motivated by the needs of the family of SCL calculi, we consider the particular case where all terms occurring in a constraint are bound by a (single) ground term. We show that this problem and variants of this problem remain NP-complete even if the form of atoms in the constraint is further restricted. In addition, for a non-strict, partial term ordering solely based on symbol counting constraint solving remains NP-complete. Nevertheless, we provide a new simple algorithm testing KBO constraint solvability that performs well on benchmark examples.

Keywords: KBO Constraint Solving · NP-complete problem · Weight Ordering Constraint Solving

1 Introduction

The family of SCL calculi (Clause Learning from Simple Models) [2,5,13] perform reasoning on a set of first-order clauses. They develop a trail of ground literals with respect to a ground term (atom) bound β and an ordering $\prec$. All ground literals on the trail are $\prec$ (or $\preceq$) smaller than the ground term (atom) β and $\prec$ should in particular have the property that for any term t there are only finitely many literals s such that $s \prec t$. In case SCL does not detect a conflict with respect to a finite, exhaustive trail of ground literals, they constitute a model candidate for the clause set [4]. If SCL detects a conflict it learns a new first-order non-ground clause. It is derived by resolution and factoring with guidance from the trail. A natural choice for the ordering $\prec$ is the Knuth-Bendix (KBO) ordering [9]. For the ground case, a KBO relation can be efficiently computed [14]. All SCL calculi propagate literals from clauses with respect to the trail. For example, given a trail $[P(a)]$ and a clause $\neg P(x) \vee R(x, y)$ the literal $R(a, y)$ could be propagated. The SCL theory only enables ground literals on the trail, however, in practice it is not affordable to put all groundings of $R(a, y)$ on the trail that are $\prec$ smaller than β. Therefore, we already considered trail literals with variables when we developed a two-watched literal scheme for SCL [3]. Recall that this propagation situation is not exceptional as typically not all literals in a clause carry all occurring variables. The consequence of this

U. Sattler and M. Suda (Eds.): FroCoS 2023, LNAI 14279, pp. 81–98, 2023.
https://doi.org/10.1007/978-3-031-43369-6_5

extension is that for SCL we now need to decide solvability of conjunctions of inequations $t_i \prec \beta$ where the t_i may contain (shared) variables, i.e., we have to decide solvability of a particular form of KBO constraints if $\prec$ is the KBO.

For the SCL(EQ) calculus [13] the requirements on constraint solving get more sophisticated. Now the trail is a sequence of unit (in)equalities and propagation and conflicting clauses are decided with respect to the resulting congruence. For an extended congruence closure algorithm [6,8,15,16] we need now in addition to inequations $t_i \prec \beta$ to consider inequalities $t_i \neq s_i$ in order to separate congruence classes. In its simplest form, constraints consist of inequations $t_i \prec \beta$ and inequalities $t_i \neq s_i$ where β and the s_i are ground, so called *simple right-ground* constraints, Definition 4. In a more general setting, the s_i carry variables and then a quantifier alternation on variables occurring in s_i but not in t_i needs to be considered. Such constraints are called *alternating*, Definition 27.

In this paper we investigate the complexity of all these variants with respect to a KBO $<$, Definitions 3, 4, 25, 27, but also a weaker non-strict ordering based on pure symbol counting, Definition 22. Except for constraints bound by a single ground term, Proposition 26, all problems are NP-hard, Propositions 5, 21, 24, 28.

Korovin and Voronkov developed a decision procedure [10] for KBO constraints consisting of inequations $s_j < t_j$ only and refined it to an NP algorithm [11]. According to Löchner [14], these results are "of more theoretical interest" because they are "too involved to be implemented with reasonable effort". In fact, to the best of our knowledge we present the first implemented algorithm for KBO constraint solving in this paper. Later, Korovin and Voronkov [12] showed that checking satisfiability of a KBO constraint consisting of a single inequation $s < t$ can be done in polynomial time. For the special case of a right-ground constraint consisting of a single inequation $s < t$, what their algorithm essentially does is assigning the minimal constant to every variable.

To the best of our knowledge the problem of simple right-ground KBO constraints has never been studied before. We are also not aware of any implementation of a KBO constraint solving algorithm. The paper is now organized as follows: In Sect. 3 we prove the NP-completeness of this problem and present an algorithm to solve it. In Sect. 4 we study the complexity of variants of this problem including alternating constraints. We also consider a non-strict, partial ordering based on symbol counting and weaker than a KBO. The algorithm for right-ground constraints is extended to alternating constraints. In Sect. 5 we put the algorithm developed in Sects. 3 and 4 to practice and end the paper with a discussion of the obtained results, Sect. 6.

2 Preliminaries

In the following let Σ be a *signature*, i.e., a finite set of function symbols. Every function symbol f has an associated *arity* which we denote by $\mathrm{arity}(f)$. Function symbols c with $\mathrm{arity}(c) = 0$ are called *constants*. We denote the set of all *terms* by $T(\Sigma, \mathcal{X})$ where $\mathcal{X}$ is an infinite set of variables. $Vars(t)$ denotes the set of

variables occurring in the term t. A term t is called *ground* if it contains no variables, i.e., $Vars(t) = \emptyset$. The set of all *ground terms* is denoted by $T(\Sigma)$. We assume that Σ contains at least one non-constant function and at least one constant, i.e., that $T(\Sigma)$ is infinite. For otherwise, constraint solving becomes trivial. A *substitution* is a mapping $\sigma : \mathcal{X} \to T(\Sigma, \mathcal{X})$ such that $\sigma(x) \neq x$ for only finitely many $x \in \mathcal{X}$. The application $t\sigma$ of a substitution σ to a term $t \in T(\Sigma, \mathcal{X})$ is defined in the usual way. We call a substitution *grounding* for some term $t \in T(\Sigma, \mathcal{X})$ if $t\sigma$ is ground. A substitution σ is a *matcher* from s to t if $s\sigma = t$. We consider the following version of the Knuth-Bendix ordering (KBO) on ground terms:

Definition 1 (KBO on Ground Terms [9]). *Let $\succ$ be a strict total ordering (a precedence) on Σ, and $w : \Sigma \to \mathbb{N}^+$ a weight function. w is extended to terms recursively by $w(f(t_1, \ldots, t_n)) = w(f) + \sum_{i=1}^{n} w(t_i)$. The Knuth-Bendix ordering $>_{KBO}$ induced by $\succ$ and w is defined by $s >_{KBO} t$ iff*

1. *$w(s) > w(t)$, or*
2. *$w(s) = w(t)$, and*
   (a) *$s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$ and $f \succ g$, or*
   (b) *$s = f(s_1, \ldots, s_m)$, $t = f(t_1, \ldots, t_m)$ and $(s_1, \ldots, s_m) >_{KBO}^{lex} (t_1, \ldots, t_m)$.*

In particular, the precedence is strict and total, no unary function f with $w(f) = 0$ is allowed and all weights are natural numbers. It can be shown that $>_{\mathrm{KBO}}$ is a strict, total and well-founded ordering on ground terms. In the following, we simply write $>$ for $>_{\mathrm{KBO}}$.

Definition 2. *A KBO constraint C is a finite set of atoms $t\#s$ where $t, s \in T(\Sigma, \mathcal{X})$ and $\# \in \{<, >, \neq, \leq, \geq, =\}$. We say that $C = \{t_1 \#_1 s_1, \ldots, t_n \#_n s_n\}$ is satisfiable if there exists a substitution σ that is grounding for all t_j, s_j such that*

$$\bigwedge_{j=1}^{n} t_j\sigma \ \#_j \ s_j\sigma.$$

Such a grounding substitution σ is called a solution.

Definition 3. *A right-ground KBO constraint C is a KBO constraint where $s_1, \ldots, s_n \in T(\Sigma)$, i.e., only the t_j may contain variables.*

Definition 4. *A simple right-ground KBO constraint C is a right-ground KBO constraint where $\# \in \{<, \neq\}$.*

For simple right-ground KBO constraints, we prefer more explicit notation: We now assume $t_1, \ldots, t_n, l_1, \ldots, l_m \in T(\Sigma, \mathcal{X})$, $s_1, \ldots, s_n, r_1, \ldots, r_m \in T(\Sigma)$ and call C satisfiable if there exists a substitution σ that is grounding for all t_j, l_j such that

$$\left(\bigwedge_{j=1}^{n} t_j\sigma < s_j\right) \wedge \left(\bigwedge_{j=1}^{m} l_j\sigma \neq r_j\right).$$

3 Simple, Right-Ground KBO Constraints

We start by investigating the complexity of simple, right-ground KBO constraint solving.

Proposition 5. *Checking satisfiability for simple right-ground KBO constraints is NP-hard.*

Proof. We reduce from MONOTONE 3SAT which is NP-complete by [7]. Let $N \uplus M$ be a set of clauses where N consists of the clauses with only positive literals and M consists of the clauses with only negative literals. We consider a signature with a constant a, a ternary function f and a unary function g. We use a KBO instance where all weights are 1 and $f \succ g \succ a$. For every propositional variable P occurring in $N \uplus M$, we introduce a variable x_P. Then the equation $x_P = a$ stands for P is *true* and $x_P \neq a$ stands for P is *false*.

Now every positive clause $(P \vee Q \vee R) \in N$ is encoded as an inequation $f(x_P, x_Q, x_R) < f(g(a), g(a), g(a))$. Obviously, this inequation can only be satisfied by a grounding that maps at least one of these variables to a, i.e., that sets at least one of P, Q, R to *true*.

Every negative clause $(\neg P \vee \neg Q \vee \neg R) \in M$ is encoded as an inequality $f(x_P, x_Q, x_R) \neq f(a, a, a)$. Obviously, this can only be satisfied if not all of these variables are mapped to a, i.e., if at least one of P, Q, R is *false*.

Now the clause set has a solution iff there is a solution to the constructed simple right-ground KBO constraint. Assume $N \uplus M$ is satisfiable by a valuation β. Then for every propositional variable P map x_P to a if $\beta(P) = 1$ and to $g(a)$ otherwise. As explained above, this grounding will satisfy the constraint. Now let σ be a solution to the constraint. Then the valuation β where $\beta(P) = 1$ if $\sigma(x_P) = a$ and $\beta(P) = 0$ otherwise satisfies $N \uplus M$.

We have added $|M|$ inequalities and $|N|$ inequations which can be constructed in polynomial time, so the reduction works in polynomial time. □

Proposition 6. *Checking satisfiability for simple right-ground KBO constraints is in NP.*

Proof. Let $C = \{t_1 < s_1, \ldots, t_n < s_n, l_1 \neq r_1, \ldots, l_m \neq r_m\}$ be a constraint. If for some inequality $l_j \neq r_j$, there is no matcher from l_j to r_j, we can ignore this inequality since it is true for every grounding. If for some inequality $l_j \neq r_j$, it actually holds that $l_j = r_j$, then this inequality is impossible to satisfy, so we are done. After sorting out these two cases, as r_j is ground, every inequality $l_j \neq r_j$ has a unique matcher τ_j which has linear size with respect to r_j. In the following, we say that the term $\tau_j(x)$ is restricted by the inequality $l_j \neq r_j$. The inequality $l_j \neq r_j$ then signifies

$$\bigvee_{x \in Vars(l_j)} \sigma(x) \neq \tau_j(x).$$

For the inequations $t_j < s_j$, it is obviously optimal to assign the smallest possible term to every variable. Larger terms only have to be considered due to

the inequalities $l_j \neq r_j$. If there is a grounding σ that satisfies $t_j < s_j$, then any grounding σ' with $\sigma'(x) \leq \sigma(x)$ for all variables x satisfies $t_j < s_j$. Hence, if there exists a solution, then there also exists a solution that only uses the $m+1$ smallest terms for every variable. This is because every inequality $l_j \neq r_j$ only restricts at most one term for every variable, so for every variable the $m+1$ smallest terms contain the smallest term that is not restricted for that variable.

As we only have to consider the $m+1$ smallest terms for every variable, the size of the groundings we have to consider is polynomially bounded by the input size. Let f be the function with the maximal arity and let $p = \text{arity}(f)$. Let a be the smallest constant. We claim that every of the $m+1$ smallest terms has at most $mp+1$ symbols. Proof by contradiction: Assume t_0 is one of the $m+1$ smallest terms with $\#t_0 > mp+1$. Perform the following m times: Obtain t_{i+1} by replacing any subterm $g(s_1, \ldots, s_n)$, where the s_i are constants, by a. The number of symbols decreases by at most p, so $\#t_i > (m-i)p+1$. As none of the t_i is a constant, such a subterm always exists. After m steps, we obtain terms $t_0 > t_1 > \cdots > t_m$ with $\#t_m > (m-m)p+1 = 1$, i.e., t_m is not a constant, so $t_m > a$. This contradicts the fact that t_0 was one of the $m+1$ smallest terms since at least $m+1$ terms are smaller than t_0. Thus, we can guess a grounding and check in polynomial time whether it is a solution. □

Next we propose an algorithm for testing satisfiability of simple right-ground KBO constraints. Of course, by Proposition 6, there already exists an algorithm, but we expect that the following algorithm performs better in practice. Let C be a simple right-ground KBO constraint with n inequations $t_j < s_j$ and m inequalities $l_j \neq r_j$.

Assume that $\mathit{Vars}(\{t_j \mid 1 \leq j \leq n\} \cup \{l_j \mid 1 \leq j \leq m\}) = \{x_1, \ldots, x_k\}$. As explained in the proof of Proposition 6, we only have to consider the $m+1$ smallest terms for the grounding, so to begin, we generate an ordered list S of the $m+1$ smallest terms. This way, a grounding substitution σ corresponds to a vector $\vec{v} \in \mathbb{N}^k$ where $v_i < m+1$ is the index of the term $\sigma(x_i)$ in S, i.e., $S[v_i] = \sigma(x_i)$. Let $\sigma(\vec{v})$ with $\sigma(\vec{v})(x_i) := S[v_i]$ denote the grounding corresponding to the vector $\vec{v}$. Later on, we give a dynamic programming algorithm to compute the k smallest terms for some number k. Actually, we do not directly generate the $m+1$ smallest terms, but start with a constant number of terms and generate more terms as needed.

The algorithm is given by three inference rules that are represented by an abstract rewrite system. They operate on a state which is either $\bot$ or a four-tuple $(T; \vec{v}; F; C)$ where T is a sequence of variables, the *trace*; $\vec{v} \in \mathbb{N}^k$ is a grounding substitution in vector notation, the *current grounding*; F is a set of *forbidden* groundings; and C is a *simple right-ground KBO constraint*. The initial state for a constraint C is $(\varepsilon; (0, \ldots, 0); \emptyset; C)$, i.e., the *trace* is empty, every variable is mapped to the smallest constant and there are no *forbidden* groundings.

We use the following partial ordering $\leq_F$ on groundings: $\vec{v} \leq_F \vec{u}$ iff for all $i \in \{1, \ldots, k\}$ we have $v_i \leq u_i$. By $\mathit{inc}(\vec{v}, i)$ we denote the grounding $\vec{v}'$ with $v'_i = v_i + 1$ and $v'_l = v_l$ for all $l \in \{1, \ldots, k\}$ with $l \neq i$, i.e., the grounding where we increase the term for the variable x_i by one. Analogously, we define $\mathit{dec}(\vec{v}, i)$,

where we instead decrease the term for the variable x_i by one, i.e., $v_i' = v_i - 1$. The two operations *inc* and *dec* are only used when they are well-defined, i.e., they yield a grounding $\vec{v} \in \mathbb{N}^k$ where $v_i < m+1$. The operation *inc* is only used when an inequality $l_j \neq r_j$ is not satisfied, and this can happen at most m times without intermediate Backtrack steps. The operation $dec(\vec{v}, i)$ is only used for Backtrack, and by Lemma 15, in this case $v_i > 0$.

The role of F is that we want to keep the algorithm from considering wrong groundings again. For all $\vec{u} \in F$, we do not visit states with grounding $\vec{v}$ if $\vec{v} \geq_F \vec{u}$. When we Backtrack, we insert the current grounding into F. The trace T records the last updated variables so Backtrack is able to undo the last Increase operation. As will be proven in Theorem 18, the algorithm terminates in $\bot$ iff there exists no solution, and if there exists a solution, then it terminates in a state where the current grounding $\vec{v}$ is a solution.

Increase $\quad (T; \vec{v}; F; C) \quad \Rightarrow_{\text{KCS}} \quad (Tx_i; \vec{v}'; F; C)$

provided $\vec{v}' = inc(\vec{v}, i)$, $l_j\sigma(\vec{v}) = r_j$ for some $l_j \neq r_j \in C$, $l_j\sigma(\vec{v}') \neq r_j$ and there is no $\vec{u} \in F$ with $\vec{v}' \geq_F \vec{u}$

Backtrack $\quad (Tx_i; \vec{v}; F; C) \quad \Rightarrow_{\text{KCS}} \quad (T; \vec{v}'; F \cup \{\vec{v}\}; C)$

provided $\vec{v}' = dec(\vec{v}, i)$ and either

1. $l_j\sigma(\vec{v}) = r_j$ for some $l_j \neq r_j \in C$, but for all $l \in \{1, \ldots, k\}$, we have that $l_j\sigma(inc(\vec{v}, l)) \neq r_j$ implies that there is a $\vec{u} \in F$ with $inc(\vec{v}, l) \geq_F \vec{u}$, or
2. $t_j\sigma(\vec{v}) \geq s_j$ for some $t_j < s_j \in C$

Fail $\quad (\varepsilon; \vec{v}; F; C) \quad \Rightarrow_{\text{KCS}} \quad \bot$

provided either

1. $l_j\sigma(\vec{v}) = r_j$ for some $l_j \neq r_j \in C$, but for all $l \in \{1, \ldots, k\}$, we have that $l_j\sigma(inc(\vec{v}, l)) \neq r_j$ implies that there is a $\vec{u} \in F$ with $inc(\vec{v}, l) \geq_F \vec{u}$, or
2. $t_j\sigma(\vec{v}) \geq s_j$ for some $t_j < s_j \in C$

Informally, Increase is applicable if some inequality $l_j \neq r_j$ is not fulfilled and we can fix this with the new grounding $inc(\vec{v}, i)$ which is not forbidden by F. Backtrack undoes an operation and is applicable if either some inequality $l_j \neq r_j$ is not fulfilled, but Increase is not applicable, or if some inequation $t_j < s_j$ is not fulfilled. Fail is applicable if Backtrack would be applicable on an empty *trace*, i.e., there is no operation to undo.

Obviously, there is no state on which we can apply both Backtrack and Fail.

Definition 7. *A reasonable strategy is a strategy that prefers Backtrack and Fail over Increase.*

Example 8. Consider a signature with constants a, b, c and a binary function f. We set $w(a) = 1; w(b) = w(c) = 2; w(f) = 3$ and $a \prec b \prec c \prec f$. We consider the constraint

$$C = \{x_1 \neq a, f(x_1, x_2) < f(a, c)\}.$$

The $m+1$ smallest terms, where $m = 1$, are a, b. This is the unique execution of the algorithm. In order to increase readability, for $\vec{v}$, we write the terms instead of the indices.

$$\begin{aligned}
& (\varepsilon; (a, a); \emptyset; C) \\
\Rightarrow_{\text{KCS}}^{\text{Increase}} \quad & (x_1; (b, a); \emptyset; C) \\
\Rightarrow_{\text{KCS}}^{\text{Backtrack}} \quad & (\varepsilon; (a, a); \{(b, a)\}; C) \\
\Rightarrow_{\text{KCS}}^{\text{Fail}} \quad & \bot
\end{aligned}$$

The algorithm terminates in $\bot$, so there is no solution.

Example 9. Consider a signature with constants a, b, a binary function g and a ternary function f. Let $w(a) = 1, w(b) = w(f) = w(g) = 2$ and $a \prec b \prec g \prec f$. The constraint is

$$C = \{x_1 < b, g(x_2, a) < g(b, b), f(x_1, x_2, x_3) \neq f(a, a, a), g(x_1, x_2) \neq g(a, b)\}.$$

The $m + 1$ smallest terms, where $m = 2$, are $a, b, g(a, a)$.

$$\begin{aligned}
& (\varepsilon; (a, a, a); \emptyset; C) \\
\Rightarrow_{\text{KCS}}^{\text{Increase}} \quad & (x_1; (b, a, a); \emptyset; C) \\
\Rightarrow_{\text{KCS}}^{\text{Backtrack}} \quad & (\varepsilon; (a, a, a); \{(b, a, a)\}; C) \\
\Rightarrow_{\text{KCS}}^{\text{Increase}} \quad & (x_2; (a, b, a); \{(b, a, a)\}; C) \\
\Rightarrow_{\text{KCS}}^{\text{Increase}} \quad & (x_2x_2; (a, g(a, a), a); \{(b, a, a)\}; C) \\
\Rightarrow_{\text{KCS}}^{\text{Backtrack}} \quad & (x_2; (a, b, a); \{(b, a, a), (a, g(a, a), a)\}; C) \\
\Rightarrow_{\text{KCS}}^{\text{Backtrack}} \quad & (\varepsilon; (a, a, a); \{(b, a, a), (a, g(a, a), a), (a, b, a)\}; C) \\
\Rightarrow_{\text{KCS}}^{\text{Increase}} \quad & (x_3; (a, a, b); \{(b, a, a), (a, g(a, a), a), (a, b, a)\}; C)
\end{aligned}$$

The algorithm has found a solution, so no rule is applicable and it terminates. Note that after the third and fifth operation, we cannot increase x_1 because $(b, b, a) \geq_F (b, a, a) \in F$.

Next we prove the correctness of the algorithm.

Lemma 10. *If* $(\varepsilon; (0, \ldots, 0); \emptyset; C) \Rightarrow_{KCS}^{l} (T; \vec{v}'; F; C)$*, then there is no* $\vec{u} \in F$ *with* $\vec{v}' \geq_F \vec{u}$*.*

Proof. We prove this by induction on l. For $l = 0$, this holds since $F = \emptyset$. For $l > 0$, the last applied rule must have been either Increase or Backtrack. If the last applied rule was Increase, then there cannot be such a $\vec{u}$ because Increase does not modify F and because this is part of the condition of the Increase rule. Now assume the last applied rule was Backtrack, so the previous state was $(Tx_i; \vec{v}; F'; C)$ with $\vec{v}' = dec(\vec{v}, i)$ and $F = F' \cup \{\vec{v}\}$. If there was some $\vec{u}$ in F such that $\vec{v}' \geq_F \vec{u}$, then, since $\vec{v}' <_F \vec{v}$, we have $\vec{v} >_F \vec{u}$. Hence, by the induction hypothesis, $\vec{u} \notin F'$, so as $F = F' \cup \{\vec{v}\}$, it must hold that $\vec{u} = \vec{v}$, contradiction to $\vec{v} >_F \vec{u}$. □

Lemma 11. *If* $(\varepsilon; (0, \ldots, 0); \emptyset; C) \Rightarrow^i_{KCS} (T; \vec{u}; F; C) \Rightarrow^l_{KCS} (T'; \vec{u}'; F'; C)$ *for* $l > 0$*, then* $\vec{u} \neq \vec{u}'$ *or* $F \neq F'$*.*

Proof. If all l rule applications are applications of the Increase rule, then clearly $\vec{u} <_F \vec{u}'$, so in particular, $\vec{u} \neq \vec{u}'$. There is no rule that removes elements from F, so $F \subseteq F'$. If there is at least one application of the Backtrack rule among the l rule applications, the current assignment $\vec{v}$ is added to F, and by Lemma 10, $\vec{v} \notin F$, so F is modified and $F \neq F'$. □

Proposition 12. $\Rightarrow_{KCS}$ *is well-founded, i.e., the algorithm always terminates.*

Proof. By Lemma 11, we can reach every combination of $\vec{v}$ and F at most once. For $\vec{v}$, there are $(m+1)^k$ possibilities. We only add occurring groundings to F, so the number of possibilities for F is upper bounded by the number of subsets of all possible groundings which is $2^{(m+1)^k}$. Thus, the number of reached states is finite (it is at most $(m+1)^k 2^{(m+1)^k}$), so the algorithm terminates. □

Of course, the upper bounds in the proof of Proposition 12 are far too high and the algorithm will run much faster in practice.

Lemma 13. *If* $(\varepsilon; (0, \ldots, 0); \emptyset; C) \Rightarrow^l_{KCS} (T; \vec{v}; F; C)$ *and* $\vec{u} \in F$*, then for all* $\vec{u}' \geq_F \vec{u}$ *it holds that* $\vec{u}'$ *cannot be a solution.*

Proof. The proof is by induction on l. For $l = 0$, we have $F = \emptyset$, so this holds. For $l > 0$, if the last applied rule was Increase, the statement follows by the induction hypothesis since F is not modified. Now assume that the last applied rule was Backtrack. Let $(T'; \vec{v}'; F'; C)$ be the previous state. We only have to show that all $\vec{u}' \geq_F \vec{v}'$ cannot be solutions, for all other elements of $F = F' \cup \{\vec{v}'\}$, this follows by the induction hypothesis. First assume that Backtrack is applicable because of condition (1). Then $\vec{v}'$ cannot be a solution since $l_j\sigma(\vec{v}') = r_j$. For $\vec{u}' >_F \vec{v}'$, if $l_j\sigma(\vec{u}') = r_j$, then $\vec{u}'$ clearly cannot be a solution. Otherwise, there is a variable x_i such that $\vec{u}' \geq_F inc(\vec{v}', i)$ and $l_j\sigma(inc(\vec{v}', i)) \neq r_j$. However, it is part of condition (1) that then, there is an element $\vec{u}'' \in F'$ with $\vec{u}'' \leq_F inc(\vec{v}', i) \leq_F \vec{u}'$, so by the induction hypothesis, $\vec{u}'$ cannot be a solution. If Backtrack is applicable because of condition (2), then $t_i\sigma(\vec{v}') \geq s_i$ for some $i \in \{1, \ldots, n\}$. Clearly, if $\vec{u}' \geq_F \vec{v}'$, then also $t_i\sigma(\vec{u}') \geq s_i$, so $\vec{u}'$ cannot be a solution. □

Corollary 14. *If* $(\varepsilon; (0, \ldots, 0); \emptyset; C) \Rightarrow^{l}_{KCS} (T; \vec{v}; F; C)$ *and condition (1) or condition (2) of Fail is fulfilled for* $(T; \vec{v}; F)$*, then for all* $\vec{u} \geq_F \vec{v}$*,* $\vec{u}$ *cannot be a solution.*

Proof. The conditions for Fail are the same as the conditions for Backtrack, so this follows by the proof of Lemma 13. □

Lemma 15. *If* $(\varepsilon; (0, \ldots, 0); \emptyset; C) \Rightarrow^{l}_{KCS} (T; \vec{v}; F; C)$*, then for all* $i \in \{1, \ldots, k\}$ *the number of occurrences of* x_i *on the trace* T *equals* v_i*.*

Proof. In the following, we denote the number of occurrences of x_i in T by $C(T, x_i)$. The proof is by induction on l. If $l = 0$, the statement trivially holds. For $l > 0$ let $(T'; \vec{v}'; F'; C)$ be the previous state. If the last applied rule was Increase, then $T = T'x_i$ and $\vec{v} = inc(\vec{v}', i)$, so

$$C(T, x_i) = C(T', x_i) + 1 \stackrel{\text{IH}}{=} v'_i + 1 = v_i.$$

For $j \neq i$, $C(T, x_j) = C(T', x_j)$ and $v_j = v'_j$, so the statement follows by the induction hypothesis. If the last applied rule was Backtrack, then $T' = Tx_i$ and $\vec{v} = dec(\vec{v}', i)$, so

$$C(T, x_i) = C(T', x_i) - 1 \stackrel{\text{IH}}{=} v'_i - 1 = v_i.$$

Again, for $j \neq i$, $C(T, x_j) = C(T', x_j)$ and $v_j = v'_j$, so the statement follows by the induction hypothesis. □

Lemma 16. *If* $(\varepsilon; (0, \ldots, 0); \emptyset; C) \Rightarrow^{l}_{KCS} (T; \vec{v}; F; C) \Rightarrow^{Fail}_{KCS} \bot$*, then there exists no solution.*

Proof. Since Fail is applicable on $(T; \vec{v}; F; C)$, $T = \varepsilon$, so by Lemma 15, $\vec{v} = (0, \ldots, 0)$. Hence, by Corollary 14, for all $\vec{u} \geq_F (0, \ldots, 0)$, $\vec{u}$ cannot be a solution, so there exists no solution. □

Lemma 17. *If* $(\varepsilon; (0, \ldots, 0); \emptyset; C) \Rightarrow^{l}_{KCS} (T; \vec{v}; F; C)$ *and no rule is applicable on* $(T; \vec{v}; F; C)$ *then* $\vec{v}$ *is a solution.*

Proof. Assume that for some $j \in \{1, \ldots, n\}$, we had $t_j\sigma(\vec{v}) \geq s_j$. Then, either Backtrack or Fail would be applicable. Now assume that for some $j \in \{1, \ldots, m\}$, we had $l_j\sigma(\vec{v}) = r_j$. Then, either Increase or Backtrack would be applicable. □

Theorem 18. *The algorithm is correct: If there exists a solution, then starting from* $(\varepsilon; (0, \ldots, 0); \emptyset; C)$*, the algorithm terminates in a state* $(T; \vec{v}; F; C)$ *where* $\vec{v}$ *is a solution. If there is no solution, the algorithm terminates in* $\bot$*.*

Proof. Follows by Proposition 12, Lemma 16 and Lemma 17. □

We have implemented the above algorithm in the context of the SPASS reasoning workbench. The efficiency of the algorithm depends on the respective variables we choose for Increase. If there exists a solution, then there exists an execution using only the rule Increase. The following criteria might be useful to select the best variable for Increase:

- We prefer variables that do not occur in "critical" inequations, or in a minimal number of inequations. A "critical" inequation is one where the weight difference is 0 or close to 0.
- We prefer variables x_i for which the next term is not restricted by any inequality $l_j \neq r_j$.
- We prefer variables x_i for which the next term does not have a larger weight, or for which the increase in weight is minimal.
- We prefer variables that fix multiple inequalities $l_j \neq r_j$ instead of just one.

It is possible to calculate and maintain some score for every variable here and decide based on this score. The exact selection criteria still need to be further explored.

A remaining problem from the presentation of the algorithm is how to compute the k smallest terms. If the occurring weights are rather small, the following dynamic programming algorithm might be useful in practice. The idea is to compute all terms of a specific weight for increasing weights until we generated at least k terms. Unfortunately, there may be exponentially many terms of a specific weight where the exponent is the maximal arity of a function and the base is the number of terms of smaller weights. However, k is bounded above by the number of inequalities m, the number of terms with smaller weights is bounded above by k and the maximal arity is probably small, so it is to be expected that this is not a big problem.

As it is probably hard to find the next possible weight, we simply always increase the weight by 1 starting by the weight of the smallest constant. Our DP array is two-dimensional, one dimension having the weight and the other dimension having the size of the tuple from 1 to *max_ arity*. Actually, it is four-dimensional since every entry is a list of tuples of terms and every tuple is a list of its entries. A tuple of size 1 is just a term of the specific weight. The tuples of larger size are needed for the DP transitions where they serve as argument tuples for the functions. We maintain an array *smallest_ terms* that will in the end contain the at least k smallest terms.

We iterate over the weights starting at the weight of the minimal constant. Let *curweight* denote the current weight. The idea is to compute all terms of weight *curweight*, sort them, add them to *smallest_ terms*, and proceed with weight $curweight + 1$ if $|smallest\_\ terms|$ is still smaller than k. To do so, if *curweight* is not the smallest weight, we first compute the tuples of size 2 to *max_ arity* for the previous weight. This is done via DP: For tuple size i we iterate over the terms $s \in smallest\_\ terms$. Then we iterate over the tuples t of size $i-1$ and weight $curweight-1-w(s)$ using the DP array and add (s,t) to the current DP entry. Afterwards, we calculate all terms of weight *curweight* by iterating over all symbols f and all tuples t of size $\text{arity}(f)$ and weight $curweight - w(f)$ using the DP array. Then, the term $f(t)$ has weight *curweight*.

We finish this section by a discussion of potential heuristics, sufficient conditions for a simple right-ground KBO constraint to have a solution. As explained before, every inequality $l_j \neq r_j$ rules out any assignment that satisfies τ_j, the matcher from l_j to r_j. Now assume we have m inequalities and know that there

are more than m solutions for the inequation $t < s$, then one might think that there is a grounding that solves all inequalities $l_j \neq r_j$ and the inequation $t < s$. However, this is not true.

Example 19. Consider a signature with constants a, b and c and a binary function f. The weights are $w(a) = 1; w(b) = w(c) = 2; w(f) = 3$ and we use $a \prec b \prec c \prec f$ as a precedence. Now consider the constraint

$$C = \{x \neq a, f(x, y) < f(a, c)\}.$$

The inequation has two solutions, namely $\{x \mapsto a, y \mapsto a\}$ and $\{x \mapsto a, y \mapsto b\}$. However, it has no solution where x is not mapped to a, so for the overall problem, there is no solution.

So the above sufficient condition needs to refined in order to be correct. However, calculating the number of solutions is again NP-hard.

Proposition 20. *Calculating the number of solutions σ for some right-ground inequation $t < s$ is NP-hard.*

Proof. We reduce from the Unbounded Subset Sum Problem (USSP) which is NP-complete by [7]. Let $s_1, \ldots, s_n, T \in \mathbb{N}^+$. We have to find out whether there are $x_1, \ldots, x_n \in \mathbb{N}$ such that $\sum_{i=1}^{n} x_i s_i = T$, i.e., whether there is a multiset of values from $\{s_1, \ldots, s_n\}$ that sums up to T. Assume we had an oracle that could compute the number of solutions for any inequation $l < r$ where r is ground. We will use this oracle twice.

For both uses, we use a signature with constants c and d and unary functions $f_1, \ldots, f_n$. We have $w(c) = 1$, $w(f_i) = s_i$ for $i \in \{1, \ldots, n\}$ and $d \prec c \prec f_1 \prec \cdots \prec f_n$. For the first case, set $w(d) = T+2$. Using the oracle with the inequation $x < d$, we get the number of terms smaller than d. Since d is the smallest term of weight $T + 2$, this is exactly the number of terms with weight $\leq T + 1$. For the second case, set $w(d) = T + 1$. Again, using the oracle with the inequation $x < d$, we get the number of terms smaller than d. This time, this is the number of terms with weight $\leq T$. If we now subtract those values, we get the number of terms with weight exactly $T + 1$.

Now the USSP has a solution iff the number of terms with weight exactly $T+1$ is not 0. Every term t of weight $T+1$ must have the constant c as subterm since the weight of d is too large. The rest of t must consist of the unary functions. Hence, the weights of the unary functions used sum up to $T + 1 - 1 = T$. Since the weights of the unary functions correspond to the numbers from the USSP, this yields a solution for the USSP. Conversely, given a solution to the USSP, we can construct a term of weight $T + 1$ analogously. □

The problem with the aforementioned insufficient condition is that an inequality $l_j \neq r_j$ does not necessarily rule out only one grounding, but possibly infinitely many groundings. This happens if there are variables that are not restricted by the matcher τ_j of l_j and r_j. However, the criterion can be refined to a correct sufficient condition. If we restrict ourselves to the $m + 1$

smallest terms again, we would again at least have a finite number of groundings that $l_j \neq r_j$ rules out. If we now sum up these numbers over all inequalities, we have an upper bound on the total number of ruled out groundings. For the inequation $t < s$, the same problem with variables that do not occur arises (there may be infinitely many solutions), so here, we restrict ourselves to the $m + 1$ smallest terms again. If now, the number of solutions for $t < s$ is larger than the upper bound on the total number of ruled out groundings, we can actually be sure that there is a solution. However, this correct sufficient condition is hard to compute and therefore seems to be not very useful in practice.

4 Further Constraint Variants and Ordering Relaxation

In this section we study further variants of constraint problems and eventually extend the algorithm of Sect. 3 to alternating KBO constraints.

Proposition 21. *Checking satisfiability for right-ground KBO constraints restricted to strict inequations is NP-hard.*

Proof. The proof strategy is the same as the one used in the proof of Proposition 5. The encoding for positive clauses stays the same as $<$ is still allowed. For negative clauses $\neg P \vee \neg Q \vee \neg R$ we encode them as $f(x_P, x_Q, x_R) > f(a, a, a)$. This inequation can only be satisfied by a grounding that does not map all of these variables to a, and is trivially satisfied by any such grounding. □

In particular, we have seen that only having constraints of the form $t_i < s_i$ and $t_i > s_i$ suffices to make the problem NP-hard. Next we turn to a weaker term ordering $\leq_{\text{sym}}$ solely based on symbol counting. Even for this ordering constraint solving remains NP-hard.

Definition 22. *For ground terms $t, s \in T(\Sigma)$, we define $t \leq_{sym} s :\iff |sym(t)| \leq |sym(s)|$, i.e., t does not contain more symbols than s.*

Definition 23. *A right-ground symbol constraint C is a finite set of atoms $t\#s$ with $t \in T(\Sigma, \mathcal{X})$, $s \in T(\Sigma)$ and $\# \in \{\leq_{sym}, \neq\}$. Satisfiability is defined analogously to the satisfiability of KBO constraints.*

Proposition 24. *Checking satisfiability for right-ground symbol constraints is NP-hard.*

Proof. The proof strategy is the same as the one used in the proof of Proposition 5. We encode positive clauses $P \vee Q \vee R$ as $f(x_P, x_Q, x_R) \leq f(g(a), g(a), a)$. The only way to satisfy this inequation is to map at least one of these variables to a. Negative clauses $\neg P \vee \neg Q \vee \neg R$ are encoded as $f(x_P, x_Q, x_R) \neq f(a, a, a)$. □

In particular, the NP-hardness of these problems is not caused by the complicated structure of the KBO since the problem is already NP-hard for a comparison as simple as counting the number of symbols.

Our next variants are motivated by the definition of congruence classes with respect to terms with variables. For the first variant, all instances of the defining term t have to be smaller than a single ground term β and different from ground terms $s_1, \ldots, s_n$.

Definition 25. *A simple, single ground KBO constraint C consists of terms $t \in T(\Sigma, \mathcal{X})$ and $s_1, \ldots, s_n, \beta \in T(\Sigma)$. We say that C is satisfiable if there exists a substitution σ that is grounding for t such that*

$$\left(\bigwedge_{j=1}^{n} t\sigma \neq s_j \right) \wedge t\sigma < \beta.$$

Proposition 26. *Assuming that we are given the $n+1$ smallest terms, checking satisfiability of simple, single right-ground KBO constraints is in P.*

Proof. Actually, for this problem, if a reasonable strategy (Definition 7) is used, the algorithm from Sect. 3 runs in polynomial time. The key difference to the other problems is that here, every variable occurs in every inequality, so every inequality rules out at most one grounding. We first show that we can only reach polynomially many states. First, consider states $(T; \vec{v}; F; C)$ where $t\sigma(\vec{v}) < \beta$. If $\vec{v}$ does not violate any inequality $t \neq s_i$, then the algorithm terminates, so there is at most one such state. For every inequality $t \neq s_i$, there is at most one grounding $\vec{v}$ that violates it. We claim that we reach at most $k+1$ states with current grounding $\vec{v}$ where k is the number of variables. $\vec{v}$ is reached at most once using Increase because otherwise, there must be an intermediate Backtrack, so $\vec{v}$ would be inserted into F. If we reach $\vec{v}$ using Backtrack for some variable x_i, then $inc(\vec{v}, i)$ was inserted into F, so for every variable x_i, we can reach $\vec{v}$ using Backtrack at most once. Hence, $\vec{v}$ is reached at most $k+1$ times.

Now, consider states $(T; \vec{v}; F; C)$ where $t\sigma(\vec{v}) \geq \beta$. Since a reasonable strategy is used, we must have reached this state from a state that does not violate $t < \beta$, so by the argumentation before, at most k such states can be reached for every inequality, so at most $n \cdot k$ in total where n is the number of inequalities.

Hence, in total, there are at most $(k+1) \cdot n + n \cdot k + 1$ states. The state transitions can be done in polynomial time because we only need to iterate over all inequalities and inequations and over all entries in F. Since there are only polynomially many states and every rule application inserts at most one element into F, F has polynomially many entries. □

Definition 27 (Alternating KBO Constraint). *An alternating KBO constraint C consists of terms $t, s_1, \ldots, s_n \in T(\Sigma, \mathcal{X})$ and $\beta \in T(\Sigma)$. We say that C is satisfiable if there exists a substitution σ that is grounding for t such that for all substitutions τ that are grounding for all s_j we have*

$$\left(\bigwedge_{j=1}^{n} t\sigma \neq s_j\tau \right) \wedge t\sigma < \beta.$$

Proposition 28. *Checking satisfiability for alternating KBO constraints is NP-hard.*

Proof. We reduce from SAT. Let N be a set of clauses and $X_1, \ldots, X_k$ be the variables occurring in N. We use a signature with a k-ary function f, two constants a and b, variables $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$. Set $t = f(x_1, \ldots, x_k)$. Now for every clause $C_j \in N$ we introduce an inequality $f(x_1, \ldots, x_k) \not\approx f(s_{j,1}, \ldots, s_{j,k})$ where we set $s_{j,i} = b$ if X_i occurs positively in C_j, $s_{j,i} = a$ if X_i occurs negatively in C_j and $s_{j,i} = y_i$ if X_i does not occur in C_j. The idea is that $x_i = a$ stands for X_i is set to $\top$ and $x_i = b$ stands for X_i is set to $\bot$. $\forall \tau. x_i\sigma \not\approx y_i\tau$ is obviously impossible to satisfy, so the inequality must be made true by setting some positive variable to a or some negative variable to b.

To ensure that the x_i are only mapped to a or b, we do the following: We first introduce a new constant c and set $\beta = c$. Then we set $w(f) = w(a) = w(b) = 1$, $w(c) = k + 2$ and $c \prec a \prec b \prec f$. If all variables x_i are mapped to a or b, we have $w(t\sigma) = k + 1$, i.e., $t\sigma < \beta$. Any grounding where some x_i is not mapped to a or b results in $t\sigma \geq \beta$.

Now there is a solution σ iff there is a satisfying valuation for N. □

If a reasonable strategy is used, satisfiability of alternating KBO constraints can be checked using the algorithm from Sect. 3. Any solution σ must be such that $t\sigma < \beta$, so we only have to consider instances of the $s_j\tau$ with $s_j\tau < \beta$. What we can now do is to calculate for all s_j all groundings τ with $s_j\tau < \beta$ and add the inequality $t \not\approx s_j\tau$ to the constraint. There are only finitely many such groundings because we did not allow unary functions f with $w(f) = 0$. This way, we obtain a simple right-ground KBO constraint, so we can apply the algorithm. A more efficient possibility to do this is to add the groundings of the s_j implicitly, i.e., to change the condition of Increase (and the first case of Backtrack and Fail) to whether there exists a matcher τ such that $l_j\sigma(\vec{v}) = r_j\tau$. Also, the condition for the next grounding for Increase changes: It is not that we fix the inequality anymore, but that we change a variable that occurs on the left side of the inequality.

Example 29. Consider the signature $\Sigma = \{f^{(2)}, g^{(1)}, a^{(0)}\}$, where the superscript numbers denote the function arities, together with the following alternating KBO constraint C:

$$t = f(x_1, x_2) \qquad s_1 = f(g(y_1), y_2)$$
$$\beta = f(f(a, a), a) \qquad s_2 = f(a, a)$$

We set $w(a) = w(g) = w(f) = 1$ and $a \prec g \prec f$. The few smallest terms are

$$a, g(a), g(g(a)), f(a, a).$$

Note that for alternating KBO constraints, it does not suffice anymore to consider the $n + 1$ smallest terms only since an inequality may rule out more than

one term for a variable. However, as mentioned in Sect. 3, we calculate the smallest terms as needed, so this is not a problem. For shorter notation, for F, we omit groundings $\vec{u}$ if there is a grounding $\vec{v} \in F$ with $\vec{v} <_F \vec{u}$. A possible run of the algorithm looks as follows:

$$
\begin{array}{lll}
 & (\varepsilon; (a,a); \emptyset; C) & \\
\Rightarrow^{\text{Increase}}_{\text{KCS}} & (x_1; (g(a),a); \emptyset; C) & s_2,\ \tau = \{\} \\
\Rightarrow^{\text{Increase}}_{\text{KCS}} & (x_1x_1; (g(g(a)),a); \emptyset; C) & s_1,\ \tau = \{y_1 \mapsto a, y_2 \mapsto a\} \\
\Rightarrow^{\text{Increase}}_{\text{KCS}} & (x_1x_1x_1; (f(a,a),a); \emptyset; C) & s_1,\ \tau = \{y_1 \mapsto g(a), y_2 \mapsto a\} \\
\Rightarrow^{\text{Backtrack}}_{\text{KCS}} & (x_1x_1; (g(g(a)),a); \{(f(a,a),a)\}; C) & \beta \\
\Rightarrow^{\text{Backtrack}}_{\text{KCS}} & (x_1; (g(a),a); \{(g(g(a)),a)\}; C) & s_1 \\
\Rightarrow^{\text{Backtrack}}_{\text{KCS}} & (\varepsilon; (a,a); \{(g(a),a)\}; C) & s_1 \\
\Rightarrow^{\text{Increase}}_{\text{KCS}} & (x_2; (a,g(a)); \{(g(a),a)\}; C) & s_2,\ \tau = \{\}
\end{array}
$$

5 Experiments

We implemented the algorithm of Sect. 3 and its extension to constraints with right hand side variables, Definition 27, and tested it in the context of an extended congruence closure (CC) algorithm with variables [6,8,15,16]. We implemented a rather naive variant of [8] with the only goal to generate KBO constraints in order to test our new algorithm on KBO constraints. In contrast to [8] our algorithm considers a finite signature, as usual for first-order logic problems. All experiments were carried out on a Debian Linux server equipped with AMD EPYC 7702 64-Core CPUs running at 3.35GHz and an overall memory of 2TB. The result of all runs as well as all input files and binaries can be found at https://nextcloud.mpi-klsb.mpg.de/index.php/s/BAwd99cxFpSJmSp.

As a first test case we considered all eligible UEQ problems from CASC-J11 [17]. We consider equations and all inequalities for the congruence closure algorithm. The equations generate the congruence and for the inequalities we compute the congruence classes for the respective right and left side term of the inequality. For each example, the KBO function weight was always set to one and the precedence is generated with respect to the occurrence of symbols in the input file in ascending order. For β we chose a fixed nesting depth of 4 and build for each input file a nested term of exactly this depth using function symbols in the order of occurrence in the input, starting with a non-constant function symbol. Out of all eligible problems our CC algorithm terminated on 186 problems within a time limit of 30 min. Please note that although our CC implementation is rather naive, in contrast to the classical ground CC algorithm it does not need a complete grounding; for the examples where our naive algorithm runs out of time a complete grounding is not affordable. The below table shows some typical runs on the UEQ domain. All timings are presented in hundredths of a second and

if they take less than one hundredth of a second we write zero. The below table shows the problem name, the number of ground terms smaller than β indicating the solution space for the constraint, the summed up time of all calls to the KBO constraint solver during the CC run, the number of calls to the KBO constraint solver, and the results of these calls. The three selected examples are typical: most of the problems are satisfiable and the constraint solving algorithm needs almost no time. Note that for the first example all 8014 calls to the constraint solver needed in sum 3 hundreds of a second. The LAT143-1 is the example showing the worst constraint solving performance, i.e., still less than a hundreth of a second per call.

Problem	$< \beta$	Time KBO Constraint Solver	#calls	# true	# false
GRP183-3	9969	3	8014	7946	68
LAT143-1	29720	8797	31033	29554	1479
GRP409-1	103565	0	6	6	0

For the SMT-LIB examples of the UF domain [1], we expanded let operators, removed the typing, coded predicates as equations, did a CNF transformation and then took the first literal of each clause as input for the CC algorithm. Nesting depth was set to 2, the rest done as for the UEQ examples. Removing types means that the number of smaller terms increases, i.e., the problems get potentially more difficult for the constraint solver, in particular for unsatisfiable constraints. The below table again shows some typical results. 1112 examples could be performed by the CC algorithm inside 30 min. The UF domain contains larger examples compared to the UEQ domain, but the characteristics remain. Constraint solving itself takes almost no time. Again all timings are presented in hundredths of a second.

Problem	$< \beta$	Time KBO Constraint Solver	#calls	# true	# false
00336	2120806	0	131	131	0
uf.926761	138397692	0	3882	1988	1894
uf.555113	254939	134	5120	3306	1814

Here uf.555113 is the worst example on constraint solving time with 1.34 s for 5120 calls. Although alternating KBO constraint solving is NP-hard, in practice there are typically only a few inequalities meaning that out of the overall number of terms smaller β, only a few need to be considered.

6 Discussion

We have studied a number of specific KBO constraint solving problems motivated by the SCL calculus and established their complexity. Except for simple,

single right-ground KBO constraints all studied problems are proven NP-hard. We propose an algorithm that eventually runs for alternating KBO constraints which include a quantifier alternation. The algorithm shows nice performance on benchmark problems. Our next step is to turn our naive CC implementation with variables into a robust algorithm.

Acknowledgments. We thank our reviewers for their constructive comments that helped us improve the paper.

References

1. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB) (2016). https://www.smt-lib.org/
2. Bromberger, M., Fiori, A., Weidenbach, C.: Deciding the Bernays-Schoenfinkel fragment over bounded difference constraints by simple clause learning over theories. In: Henglein, F., Shoham, S., Vizel, Y. (eds.) VMCAI 2021. LNCS, vol. 12597, pp. 511–533. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67067-2_23
3. Bromberger, M., Gehl, T., Leutgeb, L., Weidenbach, C.: A two-watched literal scheme for first-order logic. In: Konev, B., Schon, C., Steen, A. (eds.) Proceedings of the Workshop on Practical Aspects of Automated Reasoning Co-located with the 11th International Joint Conference on Automated Reasoning (FLoC/IJCAR 2022), Haifa, Israel, 11–12 August 2022. CEUR Workshop Proceedings, vol. 3201. CEUR-WS.org (2022)
4. Bromberger, M., Schwarz, S., Weidenbach, C.: Exploring partial models with SCL. In: Piskac, R., Voronkov, A. (eds.) Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning. EPiC Series in Computing, vol. 94, pp. 48–72. EasyChair (2023). https://doi.org/10.29007/8br1
5. Bromberger, M., Schwarz, S., Weidenbach, C.: SCL(FOL) revisited (2023). https://doi.org/10.48550/ARXIV.2302.05954. https://arxiv.org/abs/2302.05954
6. Downey, P.J., Sethi, R., Tarjan, R.E.: Variations on the common subexpression problem. J. ACM **27**(4), 758–771 (1980)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Mathematical Sciences Series. Freeman, New York (1979)
8. Hurd, J.: Congruence classes with logic variables. Log. J. IGPL **9**(1), 53–69 (2001)
9. Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebras. In: Leech, I. (ed.) Computational Problems in Abstract Algebra, pp. 263–297. Pergamon Press (1970)
10. Korovin, K., Voronkov, A.: A decision procedure for the existential theory of term algebras with the Knuth-Bendix ordering. In: Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No. 99CB36332), pp. 291–302. IEEE (2000)
11. Korovin, K., Voronkov, A.: Knuth-Bendix constraint solving is NP-complete. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 979–992. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-48224-5_79
12. Korovin, K., Voronkov, A.: Orienting rewrite rules with the Knuth-Bendix order, vol. 183, pp. 165–186. Elsevier (2003)

13. Leidinger, H., Weidenbach, C.: SCL(EQ): SCL for first-order logic with equality. In: Blanchette, J., Kovács, L., Pattinson, D. (eds.) IJCAR 2022. LNCS, vol. 13385, pp. 228–247. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-10769-6_14
14. Löchner, B.: Advances in Equational Theorem Proving-Architecture, Algorithms, and Redundancy Avoidance. Dissertation, Fachbereich Informatik, TU Kaiserslautern (2005)
15. Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. J. ACM **27**(2), 356–364 (1980)
16. Shostak, R.E.: Deciding combinations of theories. J. ACM **31**(1), 1–12 (1984)
17. Sutcliffe, G.: The CADE ATP system competition - CASC. AI Mag. **37**(2), 99–101 (2016)