Relaxed Weighted Path Order in Theorem Proving

Jan Jakubův and Cezary Kaliszyk

Mathematics Subject Classification (2010). Primary 68T15; Secondary 68Q42, 68Q60, 06F99.

Keywords. Automated Theorem Proving, First-order Logic, Term Orderings, Term Rewriting, Superposition Calculus, Weighted Path Order.

Abstract. We propose an extension of the automated theorem prover E by the weighted path ordering (WPO). Weighted path ordering is theoretically stronger than all the orderings used in E Prover, however its parametrization is more involved than those normally used in automated reasoning. In particular, it depends on a term algebra. We integrate the ordering in E Prover and perform an evaluation on the standard theorem proving benchmarks. The ordering is complementary to the ones used in E prover so far.

Furthermore, first-time presented here, we propose a relaxed variant of the weighted path order as an approximation of the standard WPO definition. A theorem prover strategy with a relaxed order can be incomplete, which is, however, not an issue as completeness can be easily regained by switching to a complete strategy. We show that the relaxed weighted path order can have a huge impact on an improvement of a theorem prover strategy.

1. Introduction

In the last two decades the superposition calculus has become one of the main foundations of automated theorem provers for first-order logic. Indeed the systems regularly winning the yearly CADE ATP Systems Competition, such as E Prover [9] and Vampire [4] are based on the superposition calculus. Also for the problems not previously solved by humans, superposition calculus based Prover9 has been most useful so far [7].

The use of powerful and efficient orderings is one of the major advantages of the superposition calculus for classical first-order theorem proving. Orderings allow provers to avoid redundant clauses, namely clauses which only differ in the order of literals, as well as permit orienting equations and therefore rewriting the clauses only in one direction. The three predominantly used orderings in automated theorem proving are LPO, KBO, and RPO. In fact, for the former two optimized implementations are known [6, 5].

However, term rewriting research has shown that there exist more powerful orderings, for example the *weighted path order* (WPO) [12] is one of the strongest known orderings. With carefully selected parameters is can subsume most known orderings including LPO, KBO, and RPO [13]. There are however two reasons, why such stronger orderings have not been tried for automated reasoning so far. First, they often rely on complicated parameters. For example WPO relies on an

This work was kindly supported by the ERC Consolidator grant no. 649043 *AI4REASON*, by ERC Starting grant no. 714034 *SMART*, and by Cost Action CA15123 *EUTypes*.

algebra on terms as an argument. Second, the efficiency of KBO, LPO, or even RPO has been optimized for the most common cases, whereas the more advanced orderings have been stated in a general manner, without optimizing their efficiency.

This paper extends our previous research [2] where we attempt to overcome both of these obstacles and propose an efficient way to implement WPO as part of an automated reasoning system. We also propose parameters that allow WPO to function efficiently within a state-of-the-art automated theorem prover and help with actual theorem proving problems. After discussing the preliminaries on term orderings in Section 2 and on their use in the superposition calculus in Section 3, the particular contributions of this paper are:

- We propose algebras that can be used efficiently for first-order theorem proving (Section 4).
- First-time presented in this paper, we propose a relaxed version of WPO based on approximation of the standard WPO definition (Section 5).
- We evaluate WPO against existing orderings in E Prover on parts of the TPTP library, the proofs stemming from the AIM conjecture [10], and on the CoqHammer proofs [1] in Section 6.
- We show that relaxed WPO can provide a huge benefit for a theorem proving strategy (Section 6.2).
- Additionally to our previous research, we provide an evaluation of effectiveness of our implementation based on real CPU time limits.

This work is an extended version of our ICMS 2018 paper [2]. In comparison with that work, the relaxed WPO, efficient implementation, and a more extensive evaluation including an evaluation based on CPU time limit are the main new contributions presented first-time in this paper.

2. Term Orderings and Rewriting

We work in first-order logic (FOL). A *signature* Σ is a collection of *symbols* with *arities*. The set of first-order *variables* is denoted $\mathcal{V}$, and $\mathcal{T}_\Sigma$ stands for the *terms* over signature Σ and variables $\mathcal{V}$. A *literal* is an atomic formula or its negation, and a *clause* is a disjunction of literals. In ATPs, clauses are used to describe both the input problem, and the knowledge inferred during the search. On occasion, *unit equality* clauses of the form $s = t$ are inferred. Such equalities can be used to simplify other clauses using $s \to t$ or $t \to s$ as a *rewriting rule*.

Rewriting systems, described by finite sets of rewriting rules, are often used inside ATPs to keep a set of clauses in *normal forms*. A crucial property for ATPs is the *termination* of every rewriting chain on any term. The termination of system $\mathcal{R}$ can be shown using a well-founded *term ordering* $>_\mathcal{T}$ on terms $\mathcal{T}$, that orients every rule $(s \to t) \in \mathcal{R}$, meaning $s >_\mathcal{T} t$. Terminating rewriting systems are called *reduction orders*. See [8, 13] for details.

Reduction orders are successfully used in many state-of-the-art ATPs. Common orders [8, 13] are lexicographic path order (LPO) and Knuth-Bendix order (KBO). LPO extends a *precedence* $>_\Sigma$ on symbols to a reduction order on $\mathcal{T}_\Sigma$ by a variety of subterm comparisons. KBO is generated by a precedence and symbol *weights*. Terms in KBO are first compared by weights and the subterm comparisons are necessary only if the weights differ. WPO further abstracts the idea of symbol weight comparisons to comparisons in *algebras*.

In this section, we remind the theoretical definitions of the orderings LPO and KBO used in E Prover, and remind the theoretical definition of WPO. We mostly follow [8] for LPO and KBO and [13] for WPO, and we refer the reader there for further details.

All the orderings will be defined on first-order terms $\mathcal{T}_\Sigma$, and rely on a precedence $>_\Sigma$, which needs to be a proper order on the symbols from signature Σ.

Definition 2.1 (LPO [8]). *Given a precedence on symbols $>_\Sigma$, we define the lexicographic path order (LPO) $>_{\text{lpo}}$ as follows: $s = f(s_1, \ldots, s_n) >_{\text{lpo}} t$ iff one of the following conditions holds:*

1. $t = f(t_1, \ldots, t_n)$ *and* $\exists i \in \{1, \ldots, n\}$ *such that*
   (i) $s_j = t_j$ *for all* j *such that* $1 \leqslant j < i$,
   (ii) $s_i >_{\text{lpo}} t_i$, *and*
   (iii) $s >_{\text{lpo}} t_j$ *for all* j *such that* $i < j \leqslant n$,
2. $t = g(t_1, \ldots, t_m)$, $f >_\Sigma g$, *and* $s >_{\text{lpo}} t_i$ *for all* i *such that* $1 \leqslant i \leqslant m$, *or*
3. $s_i \geq_{\text{lpo}} t$ *for all* i *such that* $1 \leqslant i \leqslant n$.

Where $\geq_{\text{lpo}}$ *is the reflexive closure of* $>_{\text{lpo}}$.

In order to define KBO, we additionally need a weight function induced by a pair (w, w_0) where w is symbol weight function and w_0 is a constant variable weight. The constant w_0 must be greater than zero, and the mapping w from the signature Σ to the natural numbers is defined such that $w(c) \geqslant w_0$ for any constant $c \in \Sigma$. The weight function w on symbols from Σ is naturally extended to the weight function on terms $\mathcal{T}_\Sigma$ as follows.

$$w(x) = w_0 \text{ for } x \in \mathcal{V} \qquad w(f(s_1, \ldots, s_n)) = w(f) + w(s_1) + \cdots + w(s_n)$$

Additionally if a unary function f has weight 0, than f is the greatest element wrt. the precedence. In the following, $|s|_x$ denotes the number of occurrences of variable $x \in \mathcal{V}$ in term $s \in \mathcal{T}_\Sigma$.

Definition 2.2 (KBO [8]). *Given a precedence* $>_\Sigma$ *and the weight function induced by* (w, w_0), *we define the Knuth-Bendix order (KBO)* $>_{\text{kbo}}$ *on* $\mathcal{T}_\Sigma$ *as follows:* $s >_{\text{kbo}} t$ *iff* $|s|_x \geqslant |t|_x$ *for all* $x \in \mathcal{V}$ *and one of the following conditions holds:*

1. $w(s) > w(t)$, *or*
2. $w(s) = w(t)$ *and one of the following conditions holds:*
   (i) $t \in \mathcal{V}$ *and* $s = f^n(t)$ *for an unary function symbol* f *and* $n > 0$,
   (ii) $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$, *and* $\exists i \in \{1, \ldots, n\}$ *such that* $s_j = t_j$ *for all* $1 \leqslant j < i$ *and* $s_i >_{\text{kbo}} t_i$, *or*
   (iii) $s = f(s_1, \ldots, s_n)$, $t = g(t_1, \ldots, t_m)$, *and* $f > g$.

Weighted path order (WPO) further abstracts the weight function to the notion of algebras on first-order terms defined as follows.

Definition 2.3. *An* algebra $\mathcal{A}$ *over* Σ *consists of a well-ordered carrier set and of an* interpretation $f_\mathcal{A} : \mathbb{N}^n \to \mathbb{N}$ *for every n-ary function symbol* f *from* Σ. *An algebra* $\mathcal{A}$ *is* weakly monotone *iff* $a \geq b$ *implies* $f(\ldots, a, \ldots) \geq f(\ldots, b, \ldots)$, *and* weakly simple *iff* $f(\ldots, a, \ldots) \geq a$ *for every* $f \in \Sigma$.

In this work, we consider the carrier set always to be $\mathbb{N}$ with the standard order on $\mathbb{N}$. Given a variable assignment $\sigma : \mathcal{V} \to \mathbb{N}$, we can structurally interpret every term $t \in \mathcal{T}_\Sigma$ using interpretations from algebra $\mathcal{A}$ as the number $\sigma_\mathcal{A}(t) \in \mathbb{N}$, formally as follows.

$$\sigma_\mathcal{A}(x) = \sigma(x) \qquad \sigma_\mathcal{A}(f(s_1, \ldots, s_n)) = f_\mathcal{A}(\sigma_\mathcal{A}(s_1), \ldots, \sigma_\mathcal{A}(s_n)))$$

Thus the algebra $\mathcal{A}$ induces the following ordering $>_\mathcal{A}$ on terms: $s >_\mathcal{A} t$ iff $\sigma_\mathcal{A}(s) > \sigma_\mathcal{A}(t)$ for every variable assignment σ. Similarly, we write $s \geq_\mathcal{A} t$ iff $\sigma_\mathcal{A}(s) \geq \sigma_\mathcal{A}(t)$ for every σ. The following defines WPO induced by $\mathcal{A}$.

Definition 2.4 (WPO [13]). *Given a precedence* $>_\Sigma$ *and an algebra* $\mathcal{A}$ *over* Σ, *the* weighted path order $>_{\text{wpo}}$ *on* $\mathcal{T}_\Sigma$ *is defined as follows:* $s = f(s_1, \ldots, s_n) >_{\text{wpo}} t$ *iff (1)* $s >_\mathcal{A} t$, *or (2)* $s \geq_\mathcal{A} t$ *and one of the following holds:*

2a. $\exists i \in \{1, \ldots, n\}$. $s_i \geq_{\text{wpo}} t$, *or*
2b. $t = g(t_1, \ldots, t_m)$, $\forall j \in \{1, \ldots, m\}$. $s >_{\text{wpo}} t_j$ *and either*
   (i) $f >_\Sigma g$, *or*
   (ii) $f = g$ *and* $(s_1, \ldots, s_n) >^{lex}_{\text{wpo}} (t_1, \ldots, t_n)$.

Only terms comparable in $\mathcal{A}$ are comparable in $>_{\text{wpo}}$. Strict order $s >_{\mathcal{A}} t$ alone implies $s >_{\text{wpo}} t$. Otherwise $s \geq_{\mathcal{A}}$ must hold and various subterm conditions are checked. In (2a), $\geq_{\text{wpo}}$ is the reflexive closure of $>_{\text{wpo}}$, while $>_{\mathcal{A}}$ and $\geq_{\mathcal{A}}$ are separately defined orders induced by $\mathcal{A}$. In (2b/ii) the lexicographical extension $>_{\text{wpo}}^{lex}$ of $>_{\text{wpo}}$ to n-tuples is used when the compared terms have the same head symbol.

If the WPO algebra $\mathcal{A}$ is weakly monotone and weakly simple, then $>_{\text{wpo}}$ is a reduction order [13, Theorem 13]. With different algebras, WPO is known to behave like LPO [13, Theorem 19], or like KBO [13, Theorem 16], or to subsume both [13, Theorem 20]. Instantiations of WPO with different algebras are discussed in Section 4.

3. Orderings in Superposition Calculus

Saturation based automated theorem provers, like E Prover [9], attempt to prove a first-order goal conjecture G in a theory T, that is, $T \vdash G$. First, theory axioms with the negated conjecture $T \cup \{\neg G\}$ are translated to a logically equivalent set of clauses. Then, a saturation process is initiated, which selects an unprocessed clause C and computes all possible inferences of C with all the previously processed clauses. Clause C is then marked as processed and another unprocessed clause is selected. This process continues until an empty clause (contradiction) is derived, or there are no more unprocessed clauses (the set of processed clauses becomes *saturated*), or the prover runs out of resources.

The saturation process uses term orderings for various purposes depending on the selected inference rules. The classical *resolution* rule allows to infer the clause $(C_1 \vee C_2)\sigma$ from clauses $(L_1 \vee C_1)$ and $(\neg L_2 \vee C_2)$ provided L_1 and L_2 are unifiable with the unifier σ. The *ordered resolution* restricts the classical resolution rule to literals maximal in each clause (w.r.t. a fixed term ordering $>_{\mathcal{T}}$). In *paramodulation*, inferred unit equality clauses of the form $s = t$, which can be oriented using the ordering (either $s >_{\mathcal{T}} t$ or $t >_{\mathcal{T}} s$), can be used as rewriting rules ($s \to t$ or $t \to s$, respectively). The processed clauses are then kept in their normal form with respect to the inferred rewriting rules (called *demodulators*). All these extensions restrict the number of possible inferences preserving completeness (that is, they do not prevent the inference of the empty clause). Clearly, the more terms are comparable, the more inferences are restricted, which leads to a more effective search space reduction.

E Prover implements LPO and KBO. The desired term ordering can be selected using a command-line option. E implements approximately ten signature-independent methods to generate the precedence on the symbols. In this work, we shall consider the following.

(arity/iarity). Symbols are sorted by arity or reverse arity. Symbols with higher arity are larger/smaller.

(freq/ifreq). Symbols are sorted by the frequency of their occurrence in the input problem. Frequently occurring symbols are larger/smaller. In the case of the same frequency, symbols are sorted by arity.

(ufirst). Same as **arity** but unary symbols are smaller. In the case of the same arity, symbols are sorted by frequency.

(ufreq). Same as **ifreq** but unary symbols are always smaller.

KBO is additionally parametrized by a weight function (w, w_0). E implements several ways of generating weights for a given problem. We shall consider the following. All of these set the variable weight w_0 to 1 and only differ in w.

(const). The weights of all the symbols are set to the constant 1.

(arity/iarity). The weight of an n-ary function symbol is set to $n + 1$ (respectively to $m - n + 1$, where m is the largest symbol arity).

(prec/iprec). Given a symbol precedence $<$, the weight of symbol f is the number of symbols smaller/larger than f increased by 1.

(fcount/ifcount). The weight of symbol f is the number of occurrences of f in the input problem (respectively m minus the number of occurrences, where m is the frequency of the most occurring symbol).

(frank/ifrank). Sort all function symbols by frequency of occurrence (which induces a total quasi-ordering). The weight of a symbol is the rank of it's equivalence class, with less frequent symbols getting lower/higher weights.

Additionally, E allows user-defined weights for all constant symbols, which override the weight assigned by the above weight generation schemes. Finally, E allows both a specific user-defined precedence and specific symbol weights. We do not, however, consider these specific settings as they depends on a signature. Our implementation of WPO in E Prover is described in the next Section 4.

4. Implementation of WPO in E Prover

This section describes our implementation of WPO in E Prover. We introduce two specific algebras from the literature [13]. Both algebras are weakly monotone and simple, and hence instantiate WPO to a reduction order. We discuss the implementation of algebra comparisons and provide several coefficient generation schemes for WPO. We conclude by a brief description of our main WPO comparison method. First we introduce $\mathcal{S}um$*-algebras* which sum the arguments with a positive multiplier.

Definition 4.1 ($\mathcal{S}um$-algebra). *A* $\mathcal{S}um$-algebra $\mathcal{A}$ *over* Σ *induced by* (w, c) *is an algebra over* Σ *where an* n*-ary function symbol* f *is interpreted as*

$$f_{\mathcal{A}}(a_1, \ldots, a_n) = w(f) + \sum_{i=1}^{n} c(f, i) * a_i$$

where $w(f) > 0$ *is the weight of* f *and* $c(f, i) > 0$ *is the coefficient of the* i*-th argument of* f *(called* subterm coefficient*).*

Both the weights and subterm coefficients can be zero under certain additional conditions [13, Theorems 5 & 13]. All E weight generation schemes used in this work produce non-zero weights, and hence we consider only positive coefficients, mainly to simplify the implementation. Experimenting with non-zero values is left as future work. The carrier set of $\mathcal{A}$ can be instantiated by a subset of $\mathbb{N}$ ($\{n \in \mathbb{N} : n \geq w_0\}$ for some $w_0 \in \mathbb{N}$). Note, that a restriction of such a $\mathcal{S}um$-algebra to $w_0 > 0$ and $c(f, i) = 1$ is equivalent to KBO [13, Theorem 16].

Given a $\mathcal{S}um$-algebra $\mathcal{A}$ over Σ, every term $s \in \mathcal{T}_\Sigma$ can be interpreted in $\mathcal{A}$ as an expression of the grammar "$E ::= \mathbb{N} \mid \mathcal{V} \mid (E + E) \mid (\mathbb{N} * E)$". This expression contains variables $\text{vars}(s) = \{x_1, \ldots, x_n\}$. The expression can transformed to the equivalent expression $s_{\mathcal{A}}$ of the following form, which we say *interprets* s in $\mathcal{A}$ (for appropriate $c_i \in \mathbb{N}$).

$$s_{\mathcal{A}}(x_1, \ldots, x_n) = c_0 + c_1 * x_1 + \cdots + c_n * x_n$$

Since the definitions of $>_{\mathcal{A}}$ and $\geq_{\mathcal{A}}$ involve an infinite number of variable assignments, it is necessary to provide an efficient algorithm to check the algebra comparisons in WPO. The following lemma helps us to achieve that. Note that, we take the liberty of reordering variables so that shared variables come first.

Lemma 4.1. *Given* $\mathcal{S}um$*-algebra* $\mathcal{A}$ *over* Σ *and terms* $s, t \in \mathcal{T}_\Sigma$*, let* $\text{vars}(t) \subseteq \text{vars}(s) = \{x_1, \ldots, x_n\}$ *and let* $\text{vars}(t) = \{x_1, \ldots, x_m\}$ *for some* $m \leq n$*. Let*

$$s_{\mathcal{A}}(x_1, \ldots, x_n) = c_0 + c_1 * x_1 + \cdots + c_n * x_n$$
$$t_{\mathcal{A}}(x_1, \ldots, x_m) = d_0 + d_1 * x_1 + \cdots + d_m * x_m$$

be the interpretations of s and t in $\mathcal{A}$. Then the following holds.

$$\begin{array}{lll} s >_{\mathcal{A}} t & \textit{iff} & \forall i \in \{1,\ldots,m\}.\ c_i \geq d_i \textit{ and } c_0 > d_0 \\ s \geq_{\mathcal{A}} t & \textit{iff} & \forall i \in \{0,\ldots,m\}.\ c_i \geq d_i \end{array}$$

Clearly, $s >_{\mathcal{A}} t$ (and also $s \geq_{\mathcal{A}} t$) implies $\mathrm{vars}(t) \subseteq \mathrm{vars}(s)$, hence the variable requirement is not a limitation. WPO requires algebras to be weakly monotone to generate a reduction order. Similarly, the notion of *strictly monotone* algebras can be defined (using strict comparisons instead of weak ones). $\mathcal{S}um$-algebras are strictly (and hence weakly) monotone. We next define the $\mathcal{M}ax$-algebras, which use max instead of addition, making them weakly monotone.

Definition 4.2 ($\mathcal{M}ax$-algebra). *A $\mathcal{M}ax$-*algebra $\mathcal{A}$ *over Σ induced by (w, c) is an algebra over Σ where an n-ary function symbol f is interpreted as*

$$f_{\mathcal{A}}(a_1,\ldots,a_n) = \max\left(\, w(f)\ ,\ \max_{i=1}^{n}(c(f,i) + a_i)\,\right)$$

where $w(f) > 0$ is the weight of f and $c(f,i) > 0$ is the coefficient of the i-th argument of f (called subterm penalty*).*

Again, zero weights and penalties are allowed under certain conditions, which we omit in this presentation. For example, setting all the weights and penalty coefficients to zeros makes WPO behave like LPO [13, Theorem 19]. Similarly to $\mathcal{S}um$-algebras, given a $\mathcal{M}ax$-algebra $\mathcal{A}$ over Σ, every term $s \in \mathcal{T}_\Sigma$ with $\mathrm{vars}(s) = \{x_1,\ldots,x_n\}$ can be interpreted by an expression $s_{\mathcal{A}}$ of the following form, which is said to interpret s in $\mathcal{A}$.

$$s_{\mathcal{A}}(x_1,\ldots,x_n) = \max(c_0, x_1 + c_1, \ldots, x_n + c_n)$$

The following allows efficiently comparing terms in $\mathcal{M}ax$-algebras.

Lemma 4.2. *Let $\mathcal{M}ax$-algebra $\mathcal{A}$ over Σ and terms $s,t \in \mathcal{T}_\Sigma$ be given. Let* $\mathrm{vars}(t) \subseteq \mathrm{vars}(s) = \{x_1,\ldots,x_n\}$ *and* $\mathrm{vars}(t) = \{x_1,\ldots,x_m\}$ *for some $m \leq n$. Let*

$$\begin{aligned} s_{\mathcal{A}}(x_1,\ldots,x_n) &= \max(c_0, x_1 + c_1, \ldots, x_n + c_n) \\ t_{\mathcal{A}}(x_1,\ldots,x_m) &= \max(d_0, x_1 + d_1, \ldots, x_m + d_m) \end{aligned}$$

interpret s and t in $\mathcal{A}$. Let $c_{\max} = \max(c_0,\ldots,c_n)$ and $d_{\max} = \max(d_0,\ldots,d_m)$. Then the following holds.

$$\begin{array}{lll} s >_{\mathcal{A}} t & \textit{iff} & c_{\max} > d_{\max} \textit{ and } \forall i \in \{1,\ldots,m\}.\ c_i > d_i \\ s \geq_{\mathcal{A}} t & \textit{iff} & c_{\max} \geq d_{\max} \textit{ and } \forall i \in \{1,\ldots,m\}.\ c_i \geq d_i \end{array}$$

Note that in $s >_{\mathcal{A}} t$, as opposed to Lemma 4.1, we require all the coefficients to be strictly greater. Otherwise $\max(x+2, y+1)$ would be strictly greater than $\max(x+1, y+1)$. We do not compare the constant coefficients c_0 and d_0, because, for example, $\max(1, x+3)$ is always greater than $\max(2, x+2)$ even though the constant coefficients are not. The proof of Lemma 4.2 follows from the observation that c_0 can be substituted by $c_{\max}$ without affecting the value of $s_{\mathcal{A}}$.

Inspired by precedence/weight generation schemes in E, we have implemented the following subterm coefficient generation schemes. These schemes generate coefficients $c(f,i)$ to be used both in $\mathcal{S}um$ and $\mathcal{M}ax$-algebras.

(constant). All coefficients are set to 1.
(arity). For an n-ary function symbol f we set $c(f,i) = n$.
(firstmax). For all f, the first coefficient $c(f,1)$ is set 2 while the others to 1.
(firstmin). For all f, the first coefficient $c(f,1)$ is set 1 while the others to 2.
(asc/desc). Set up ascending/descending coefficients. For an n-ary function symbol f we set $c(f,i) = i$ (respectively $c(f,i) = n - i + 1$).

To implement a new term ordering $>_\mathcal{T}$ in E, a term comparison method is required. The method takes two terms s and t as input and returns whether $s <_\mathcal{T} t$, or $s >_\mathcal{T} t$, or $s = t$, or the terms are incomparable. We have implemented the WPO comparison methods for $\mathcal{S}um$ and $\mathcal{M}ax$ algebras. Our implementation mostly follows Definition 2.4. At first we check strict algebra comparisons $>_\mathcal{A}$. To do that, we compute coefficients $\overline{c_i}$ and $\overline{d_i}$ from Lemma 4.1 or 4.2 by a traversal of s and t. If the coefficients are the same, we clearly have both $s \geq_\mathcal{A} t$ and $t \geq_\mathcal{A} s$. If $s >_\mathcal{A} t$, we return $s >_{\text{wpo}} t$ (and *vice versa*). For terms incomparable with $>_\mathcal{A}$, we proceed with the weak comparison $\geq_\mathcal{A}$. If they are weakly comparable, we proceed with the subterm checks.

5. Term Rewriting with Relaxed Algebras

Algebras $\mathcal{S}um$ and $\mathcal{M}ax$ from Section 4 have nice theoretical properties, in particular, they instantiate WPO to a reduction order. In this section we try to address the question, whether forsaking some of these theoretical properties might give us an advancement in a practical use of a theorem prover. We shall introduce several *relaxed algebras* which might instantiate WPO to a non-terminating order, or even to a relation which is not an order at all. To avoid infinite loops when rewriting terms, we impose an upper bound on the length of every rewriting chain. Proof strategies with this modification of rewriting might not be complete, however, correctness is preserved. It is a known fact in theorem proving, that incomplete strategies can still be useful in practice. Moreover, any proof search can be made complete by switching to a complete strategy once incomplete strategies fail to find a proof.

All of our relaxed algebras, just like the standard complete algebras from Section 4, are induced by (w, c) where w is a symbol weight function and c is a coefficient function. We define four relaxed $\mathcal{RS}um$-algebras and four relaxed $\mathcal{RM}ax$-algebras. Each of these algebras assign a numeric weight to a term. In the case of the $\mathcal{RS}um$-algebras, we denote the weight of term t by $\mathcal{RS}um(t)$, and all of the four algebras use the following recursive formula to compute the weight of a non-variable term.

$$\mathcal{RS}um\big(f(s_1, \ldots, s_n)\big) = w(f) + \sum_{i=1}^{n} c(f, i) * \mathcal{RS}um(s_i)$$

The four relaxed $\mathcal{RS}um$-algebras differ only in the value they assign to variables. Algebra $\mathcal{RS}um_0$ simply assigns 0 to every variable, while in algebra $\mathcal{RS}um_1$ we set $\mathcal{RS}um_1(x) = 1$ for every variable x. The remaining two algebras suppose that variables in a term are numbered (starting from 1) by their first occurrence in the term from left to right. Algebra $\mathcal{RS}um_+$ assigns to each variable its number, while algebra $\mathcal{RS}um_-$ assigns the opposite number. For example, given a term $f(g(x, y), x)$, we have $\mathcal{RS}um_+(x) = 1$ and $\mathcal{RS}um_+(y) = 2$, while $\mathcal{RS}um_-(x) = -1$ and $\mathcal{RS}um_-(y) = -2$.

Similarly, we define four relaxed $\mathcal{RM}ax$-algebras. Again, each of them assigns a weight to each term t, denoted $\mathcal{RM}ax(t)$. The following common formula is used to compute the weight of a non-variable term.

$$\mathcal{RM}ax\big(f(s_1, \ldots, s_n)\big) = \max\big(\, w(f)\,,\ \max_{i=1}^{n}(c(f, i) + \mathcal{RM}ax(s_i))\,\big)$$

The algebras differ in the value they assign to variables, and this gives us four $\mathcal{RM}ax$ algebras: $\mathcal{RM}ax_0$, $\mathcal{RM}ax_1$, $\mathcal{RM}ax_+$, and $\mathcal{RM}ax_-$. The variable weights are the same as in the case of the four $\mathcal{RS}um$ algebras.

Our relaxed algebras can easily used with WPO from Definition 2.4. The terms are at first compared by their weights, and only in the case of equal weights, subterms conditions (2a) and (2b) are checked. As opposed to the standard complete algebras, every two terms are comparable in our relaxed algebras. Hence more terms are strictly comparable in our relaxed algebras, thus, the computationally expensive subterms checks should be executed less often. Hence our relaxed algebras can be expected to perform more effectively.

	Auto	LPO		KBO		WPO		*union*	
	solved	solved	by	solved	by	solved	by	*solved*	*by*
TPTP/LAT	*27*	28	2	30	2	**34**	5	*36*	*5*
TPTP/REL	*49*	68	3	59	2	**75**	2	*77*	*3*
AIM	*35*	44	2	38	2	**54**	4	*54*	*4*
COQ	*22*	26	3	**27**	2	**27**	2	*27*	*2*

TABLE 1. Total number of problems solved by all LPO, KBO, and WPO instances with a fixed limit of 1000 processed clauses per problem.

The relaxed algebras can be seen as an approximation of the complete algebras in the following way. With the complete algebras, terms are represented by expressions with variables, and the expressions are compared with respect to every possible variable assignment (see Section 2). In the relaxed algebras, we just evaluate the expressions with a single fixed variable assignment, for example, $\sigma_0 = \{x \mapsto 0 : x \in \mathcal{V}\}$ in the case of $\mathcal{RS}um_0$ or $\mathcal{RM}ax_0$.

6. Experimental Evaluations

This section provides an evaluation of our experimental implementation of WPO in E Prover.[1] We use a single good-performing E strategy with the different term orders. The strategy was randomly selected and is provided in Appendix A. Section 6.1 describes previously published [2] evaluation of standard WPO. Section 6.2 provides evaluation of WPO with relaxed algebras, first published here.

We evaluate our experimental implementation on four complementary benchmarks with around 200 problems each. Benchmark problems are from two TPTP [11] categories (LAT and REL), from the *Abelian Inner Mappings* project (AIM) [10], and from CoqHammer [1]. As we evaluate a large number of different ordering instances on all of the benchmark problems, it is important to limit the number of problems, so that the evaluation can be done in a reasonable time.[2] We, however, believe that our collection of about 800 benchmark problems is reasonably orthogonal to allow us to perform an objective evaluation. All the selected benchmark domains rely heavily on equational reasoning, and hence can be expected to benefit from improvements in term rewriting.

6.1. Evaluation of Standard WPO Implementation

We evaluate all instances of LPO, KBO, and WPO induced by the generation schemes described above, in order to estimate the value of WPO for E. Altogether we have 1410 instances to be evaluated on all the benchmark problems. The limit of 1000 processed clauses, instead of time limit, is used for an evaluation independent on implementation effectiveness. Section 6.2, however, contains evaluation conducted with a fixed CPU time limit per instance and problem.

We have 6 instances of LPO, 108 instances KBO, and 1296 of WPO. The results for each benchmark are in Table 1. For each ordering, the column "*by*" shows the least number of instances necessary to solve the number in the column *solved*. Number of problems solved by E's automated term order selection is shown in column "*Auto*". The "*union*" columns show a combined performance. Table 2 shows the best-performing instance for every order type, measuring number problems *solved* and the number of problems solved additionally to *Auto* mode (column "*E+*"). The parameters of the instances select the generation schemes for precedence, weights, algebra, and coefficients.

[1] https://github.com/ai4reason/eprover/tree/WPO

[2] The evaluations took around 4 days employing 128 cores of Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz with 128 GB memory in total.

TPTP/REL	solved	E+
WPO(freq,prec,$\mathcal{S}um$,desc)	**63**	**14**
LPO(arity)	59	12
KBO(iarity,iarity)	57	8
E (Auto)	49	0

TPTP/LAT	solved	E+
KBO(iarity,iprec)	**29**	**3**
WPO(arity,iprec,$\mathcal{S}um$,const)	28	1
LPO(arity)	27	0
E (Auto)	27	0
WPO(ifreq,prec,$\mathcal{M}ax$,desc)	24	**3**

AIM	solved	E+
WPO(freq,fcount,$\mathcal{S}um$,desc)	**41**	**5**
LPO(arity)	**41**	4
KBO(freq,ifrank,c1)	37	1
E (Auto)	35	0

COQ	solved	E+
WPO(arity,fcount,$\mathcal{S}um$,desc)	**26**	**4**
KBO(arity,fcount)	25	3
LPO(ufreq)	24	**4**
E (Auto)	22	0

TABLE 2. Best instances of LPO, KBO, and WPO for each benchmark with a fixed limit of 1000 processed clauses per problem.

WPO helped to solve more problems for each benchmark. It also solved problems unsolved by *Auto*. Furthermore, the strongest WPO is usually equal or better than the strongest version of LPO and KBO. LPO(arity) is often the best of LPOs. As for WPO, $\mathcal{S}um$ often performs better than $\mathcal{M}ax$ overall but $\mathcal{M}ax$ can solve unique problems. The algebra coefficients generated by **desc** often perform best.

As stated above, we used a limit on processed clauses rather than on runtime, in order to abstract from implementation details. In order to assess the effectiveness of our implementation, we have additionally evaluated the best performing ordering instances from Table 2 on the benchmark problems with runtime limit of 5 seconds. For each benchmark category (AIM, COQ, etc.) we have computed the average runtime on the problems solved by all the instances. The results vary on different categories but LPO is usually the fastest and KBO is in average from 10% The speed of WPO varies, but in average it is from 40% LPO. However, for example on TPTP/REL, our implementation of WPO is in average faster than both LPO and KPO. We conclude that our implementation can be definitely made more effective, but even in the current state, it can provide a valuable gain. Section 6.2 provides additional evaluation with fixed CPU time limit instead of an abstract time limit.

6.2. Evaluation of WPO with Relaxed Algebras

This section provides experimental evaluation of WPO with both standard and relaxed algebras. We evaluate all LPO, KBO, and WPO instances with fixed CPU time limit of 1 second per problem. In this way we shall be able to estimate whether there are some WPO instances which enrich standard E Prover implementation. As before, we have 6 LPO instances, 108 KBO instances, and 1296 standard WPO instances. Additionally, we introduce 5184 instances of relaxed WPO, generated by 8 relaxed algebras from Section 5. Altogether we have 6594 instances to be evaluated on all of the benchmark problems. Hence a relatively small time limit of 1 second per instance and problem was chosen in order to make this evaluation possible. This is, however, not a limitation as a reasonable correlation between results obtained with higher time limits can be expected.

The results for each benchmark are in Table 3. The table is as in the previous section, that is, the column "*solved*" shows the total number of problems solved by all the instances of LPO, KBO, WPO, and by WPO instances with relaxed algebras (denoted "R-WPO"). Again, the column "*by*" shows the least number (more precisely, the size of a greedy coverage) of instances necessary to solve the number in the column "*solved*". A full listing of instances in greedy coverage are presented in Appendix B, as these data might provide additional insight about useful ordering parameters.

	Auto	LPO		KBO		WPO		R-WPO		*union*	
	solved	solved	by	solved	by	solved	by	solved	by	*solved*	*by*
TPTP/LAT	*42*	42	1	45	2	44	2	**48**	3	48	3
TPTP/REL	*88*	92	3	95	2	98	5	**115**	7	115	6
AIM	*43*	55	2	50	3	58	7	**70**	10	71	10
COQ	*36*	37	2	37	1	38	2	**42**	3	42	3

TABLE 3. Total number of problems solved by all LPO, KBO, WPO, and relaxed WPO instances with a fixed CPU time limit of 1 second per problem.

TPTP/REL	solved
WPO(ufreq,prec,$\mathcal{RS}um_0$,const)	**95**
KBO(ifreq,fcount,c1)	92
WPO(ufreq,ifcount,$\mathcal{S}um$,const,c1)	91
LPO(arity)	73
E (Auto)	88

TPTP/LAT	solved
KBO(ifreq,const)	**44**
WPO(freq,iprec,$\mathcal{RS}um_+$,const)	**44**
WPO(iarity,iprec,$\mathcal{S}um$,const)	43
LPO(ufirst)	42
E (Auto)	42

AIM	solved
WPO(ufreq,fcount,$\mathcal{RS}um_+$,desc)	**52**
LPO(freq)	50
KBO(ifreq,frank)	47
WPO(frirst,fcount,c1,$\mathcal{S}um$,desc)	46
E (Auto)	43

COQ	solved
WPO(arity,iarity,c1,$\mathcal{RS}um_-$,arity)	**39**
WPO(iarity,frank,$\mathcal{S}um$,firstmin)	37
KBO(ufirst,iprec,c1)	37
LPO(ufreq)	36
E (Auto)	36

TABLE 4. Best instances of LPO, KBO, WPO, and relaxed WPO for each benchmark with a fixed CPU time limit of 1 second per problem.

Number of problems solved by E's automated term order selection (`-tAuto`) is shown in column "*Auto*" as a reference. The "*union*" columns show a combined performance. Additionally, Table 4 shows the best-performing instance for every ordering and benchmark, together with the number of problems solved.

We can see that WPO with relaxed algebras outperforms other ordering types. From the combined performance in the column "*union*" of Table 3 we can furthermore conclude that WPO with relaxed algebras can solve all the problems as other orderings (with the exception of one AIM problem) and more. With a relatively big number of possible WPO instances, it is, however, a question whether one can arrive at the right instantiations easily. This is further discussed in Section 7.

7. Conclusion

In this paper we proposed efficient implementations of algebras that allow integrating more powerful orderings in the superposition calculus. The resulting E strategies are more precise, resulting in complementary proofs on the various corpora and have a potential to benefit E Prover and superposition calculus ATPs in general. Furthermore, first-time presented here, we proposed a relaxed version of WPO and experimentally evaluate its benefits, and thus also benefits of relaxed term orderings for ATPs in general.

We have experimentally evaluated our implementation of WPO with standard and relaxed algebras on a single good-performing E Prover strategy. State-of-the-art theorem provers, however, are not based on a single strategy, but rather on a portfolio of complementary strategies. It is often

the case, that even a large improvement of a single strategy from the portfolio has just a minimal effect on the overall portfolio performance. This is because the additionally solved problems are often already solved by another portfolio strategy. For example, we have shown that with the selected E strategy, there are problems solved only by WPO with relaxed algebras. Whether the same problems can be solved with another E strategy with LPO or KBO is not clear. We have experimentally tried to employ portfolio invention system BliStrTune [3] in order to invent two portfolios, one with and one without our WPO orderings (both standard and relaxed). So far we have been able to reach only a 1% Whether this behavior is caused by WPO, or by a wrong configuration or limitations of BliStrTune is left for further research.

As another future work, we would like to experiment with orderings that work modulo associativity and commutativity [14]. Additionally we would like to investigate other coefficient settings, and experiment with zero weights, as this might further reduce the number of derived clauses. We would also like to further optimize the efficiency of the algebra comparisons, as well as the computation of the ordering itself.

Appendix A. E Strategy Used For Experiments

The following is the E Prover strategy used in the experiments given as E Prover command line options. Additional command line options are introduced by the given term ordering selection. For example, KBO(ifreq,fcount,c1) adds "`-tKBO6 -Ginvfreq -wfreqcount -c1`".

```
--definitional-cnf=24 --destructive-er-aggressive --destructive-er
--prefer-initial-clauses -F1 --split-clauses=0
--forward-context-sr -WSelectComplexG --oriented-simul-paramod
-H'(1*ConjectureRelativeSymbolWeight(SimulateSOS,0.5,100,100,100,
100,1.5,1.5,1),4*ConjectureRelativeSymbolWeight(ConstPrio,0.1,100,
100,100,100,1.5,1.5,1.5),1*FIFOWeight(PreferProcessed),
1*ConjectureRelativeSymbolWeight(PreferNonGoals,0.5,100,100,100,100,
1.5,1.5,1),4*Refinedweight(SimulateSOS,3,2,2,1.5,2))'
```

Appendix B. Greedy Coverage

Table 5 gives a full list of order instances in greedy covers required to solve the number of problems listed in Table 3 for each benchmark. The first numeric column states how many new problems the corresponding instance adds to problems solved by the previous instances in the sequence (from top to down). The second column states how many problems the corresponding instance solves by itself. The following abbreviations are used: "fmin" stands for for "firstmin", and "fmax" for "firstmax". Other values might be abbreviated to a unique prefix (like "constant" to "con") for space restrictions. Furthermore, $\mathcal{S}um$ and $\mathcal{M}ax$ are abbreviated to $\mathcal{S}$ and $\mathcal{M}$.

References

1. Łukasz Czajka and Cezary Kaliszyk, *Hammer for Coq: Automation for dependent type theory*, J. Autom. Reasoning **61** (2018), no. 1-4, 423–453.
2. Jan Jakubuv and Cezary Kaliszyk, *Towards a unified ordering for superposition-based automated reasoning*, ICMS, Lecture Notes in Computer Science, vol. 10931, Springer, 2018, pp. 245–254.
3. Jan Jakubuv and Josef Urban, *Hierarchical invention of theorem proving strategies*, AI Commun. **31** (2018), no. 3, 237–250.
4. Laura Kovács and Andrei Voronkov, *First-order theorem proving and Vampire*, Computer-Aided Verification (CAV 2013), LNCS, vol. 8044, Springer, 2013, pp. 1–35.

LPO	adds	solves
AIM		
LPO(freq)	+50	50
LPO(ufirst)	+5	44
COQ		
LPO(ufreq)	+36	36
LPO(iarity)	+1	33
LAT		
LPO(ufirst)	+42	42
REL		
LPO(arity)	+73	73
LPO(ifreq)	+15	56
LPO(ufirst)	+4	71

KBO	adds	solves
AIM		
KBO(ifreq,frank,0)	+47	47
KBO(freq,iprec,0)	+2	43
KBO(arity,arity,0)	+1	46
COQ		
KBO(ufirst,iprec,1)	+37	37
LAT		
KBO(ifreq,const,0)	+44	44
KBO(freq,ifrank,0)	+1	43
REL		
KBO(ifreq,fcount,1)	+92	92
KBO(arity,const,0)	+3	74

WPO	adds	sol.
AIM		
WPO(ufirst,fcount,c1,$\mathcal{S}$,desc)	+46	46
WPO(freq,frank,c1,$\mathcal{M}$,fmax)	+5	43
WPO(iarity,iprec,c1,$\mathcal{S}$,con)	+2	42
WPO(freq,prec,$\mathcal{S}$,asc)	+2	43
WPO(ufirst,ifc,$\mathcal{M}$,con)	+1	36
WPO(freq,ifc,c1,$\mathcal{M}$,con)	+1	38
WPO(freq,fcount,$\mathcal{S}$,fmax)	+1	45
COQ		
WPO(iarity,frank,$\mathcal{S}$,fmin)	+37	37
WPO(arity,iprec,$\mathcal{M}$,asc)	+1	37
LAT		
WPO(iarity,iprec,$\mathcal{S}$,con)	+43	43
WPO(ufreq,con,$\mathcal{M}$,asc)	+1	36
REL		
WPO(ufreq,prec,c1,$\mathcal{S}$,con)	+91	91
WPO(ufirst,prec,c1,$\mathcal{M}$,con)	+2	46
WPO(ifreq,fcount,c1,$\mathcal{S}$,fmin)	+2	63
WPO(ufreq,prec,c1,$\mathcal{S}$,fmax)	+2	88
WPO(ufreq,iarity,$\mathcal{S}$,con)	+1	75

R-WPO	adds	sol.
AIM		
WPO(ufreq,fcount,$\mathcal{S}_+$,desc)	+52	52
WPO(ufirst,ifc,c1,$\mathcal{S}_0$,asc)	+5	46
WPO(ufreq,fcount,$\mathcal{M}_1$,asc)	+3	50
WPO(arity,iprec,$\mathcal{M}_+$,desc)	+2	38
WPO(arity,ifc,c1,$\mathcal{M}_0$,desc)	+2	48
WPO(ufirst,fcount,c1,$\mathcal{S}_1$,ari)	+2	47
WPO(arity,iprec,$\mathcal{S}_1$,asc)	+1	38
WPO(arity,arity,$\mathcal{S}_-$,desc)	+1	29
WPO(ufreq,ifc,c1,$\mathcal{M}_+$,fmin)	+1	45
WPO(ufirst,fcount,$\mathcal{S}_-$,fmax)	+1	45
COQ		
WPO(arity,iarity,c1,$\mathcal{S}_-$,arity)	+39	39
WPO(ufirst,con,c1,$\mathcal{S}_-$,fmin)	+2	34
WPO(arity,iprec,$\mathcal{S}_-$,desc)	+1	37
LAT		
WPO(freq,iprec,$\mathcal{S}_+$,fmin)	+44	44
WPO(iarity,ifrank,$\mathcal{M}_1$,con)	+3	40
WPO(ufreq,prec,c1,$\mathcal{M}_-$,asc)	+1	39
REL		
WPO(ufreq,prec,$\mathcal{S}_0$,con)	+95	95
WPO(ifreq,iprec,$\mathcal{M}_0$,fmax)	+13	86
WPO(freq,frank,c1,$\mathcal{S}_+$,desc)	+2	80
WPO(arity,iarity,$\mathcal{M}_1$,desc)	+2	57
WPO(ifreq,con,c1,$\mathcal{M}_0$,fmax)	+1	76
WPO(iarity,iarity,$\mathcal{M}_0$,fmax)	+1	61
WPO(ufreq,iarity,$\mathcal{M}_0$,fmax)	+1	60

TABLE 5. Full listing of order instances in greedy covers.

5. Bernd Löchner, *Things to know when implementing KBO*, J. Autom. Reasoning **36** (2006), no. 4, 289–310.
6. _____, *Things to know when implementing LPO*, International Journal on Artificial Intelligence Tools **15** (2006), no. 1, 53–80.
7. William McCune, *Solution of the Robbins problem*, J. Autom. Reasoning **19** (1997), no. 3, 263–276.
8. Aart Middeldorp, *Term rewriting lecture notes*, 9th International School on Rewriting (ISR 2017), 2017.
9. Stephan Schulz, *System description: E 1.8*, Logic for Programming, Artificial Intelligence (LPAR 2013), LNCS, vol. 8312, Springer, 2013, pp. 735–743.
10. Geoff Sutcliffe, *The 8th IJCAR automated theorem proving system competition - CASC-J8*, AI Communications **29** (2016), no. 5, 607–619.
11. _____, *The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0*, Journal of Automated Reasoning **59** (2017), no. 4, 483–502.
12. Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe, *Unifying the Knuth-Bendix, recursive path and polynomial orders*, PPDP, ACM, 2013, pp. 181–192.
13. _____, *A unified ordering for termination proving*, Sci. Comput. Program. **111** (2015), 110–134.
14. Akihisa Yamada, Sarah Winkler, Nao Hirokawa, and Aart Middeldorp, *AC-KBO revisited*, TPLP **16** (2016), no. 2, 163–188.

Jan Jakubův
Czech Technical University in Prague / CIIRC
Jugoslávských partyzánů 1580/3
160 00 Praha 6
Czech Republic
e-mail: jakubuv@gmail.com

Cezary Kaliszyk
University of Innsbruck
Technikerstraße 21a/2
6020 Innsbruck
Austria
e-mail: cezary.kaliszyk@uibk.ac.at