Towards a Unified Ordering for Superposition-based Automated Reasoning*

Jan Jakubův[1] and Cezary Kaliszyk[2]

[1] Czech Technical University in Prague, Prague, Czech Republic
`jakubuv@gmail.com`
[2] University of Innsbruck, Innsbruck, Austria
`cezary.kaliszyk@uibk.ac.at`

Abstract. We propose an extension of the automated theorem prover E by the weighted path ordering. 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 discuss how the parameters for the ordering can be proposed automatically for particular theorem proving problem strategies. 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.

Keywords: automated reasoning, term orderings, weighted path order, superposition calculus

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 [7] and Vampire [2] 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 [5].

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 [4,3].

However, term rewriting research has shown that there exist more powerful orderings, for example the *weighted path order* (WPO) [10] is one of the strongest known orderings. With carefully selected parameters is can subsume most known

* Supported by the ERC Consolidator grant no. 649043 *AI4REASON*, ERC Starting grant no. 714034 *SMART*, and FWF grant P26201.

orderings including LPO, KBO, and RPO [11]. 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 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.

In this paper 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),
- We present an optimized pseudocode for WPO in terms of typical ATP structures and implement an extension of E-prover that supports WPO (Sect. 4).
- We evaluate WPO against existing orderings in E-prover on parts of the TPTP library, the proofs stemming from the AIM conjecture [9], and on the CoqHammer proofs [1] in Section 5.

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 [6,11] for details.

Reduction orders are successfully used in many state-of-the-art ATPs. Common orders [6,11] 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* as follows.

Definition 1. *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 (WPO [11]). *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 $>^{lex}_{\text{wpo}}$ 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 [11, Theorem 13]. With different algebras, WPO is known to behave like LPO [11, Theorem 19], or like KBO [11, Theorem 16], or to subsume both [11, 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 [7], 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 ($s \rightarrow t$ or $t \rightarrow s$, 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 [11]. 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 3 ($\mathcal{S}um$-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 [11, 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 [11, 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 $\mathrm{vars}(s) = \{x_1, \ldots, x_n\}$. 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 1. *Given* $\mathcal{S}um$*-algebra* $\mathcal{A}$ *over* Σ *and terms* $s, t \in \mathcal{T}_\Sigma$*, let* $\mathrm{vars}(t) \subseteq \mathrm{vars}(s) = \{x_1, \ldots, x_n\}$ *and let* $\mathrm{vars}(t) = \{x_1, \ldots, x_m\}$ *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 $\text{vars}(t) \subseteq \text{vars}(s)$, 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 ($\mathcal{M}ax$-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 [11, 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 $\text{vars}(s) = \{x_1, \ldots, x_n\}$ 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 2. *Let $\mathcal{M}ax$-algebra $\mathcal{A}$ over Σ and terms $s, t \in \mathcal{T}_{\Sigma}$ be given. Let* $\text{vars}(t) \subseteq \text{vars}(s) = \{x_1, \ldots, x_n\}$ *and* $\text{vars}(t) = \{x_1, \ldots, x_m\}$ *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 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 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. At first we check strict algebra comparisons $>_{\mathcal{A}}$. To do that, we compute coefficients $\overline{c_i}$ and $\overline{d_i}$ from Lemma 1 or 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 Experimental Evaluation

We evaluate our experimental implementation[3] of WPO in E Prover on four complementary benchmarks with 200 problems each. Benchmark problems are from two TPTP [8] categories (LAT and REL), from the *Abelian Inner Mappings* project (AIM) [9], and from CoqHammer [1]. 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. This gives us a collection of about 800 benchmark problems which we believe are reasonably orthogonal to allow us to perform an objective evaluation. As we evaluate around 1400 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.[4] The limit of 1000 processed clauses, instead of time limit, is used for an evaluation independent on implementation effectiveness. We use a single good-performing E strategy with the different term orders.

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 "*all*" columns show 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.

[3] `https://github.com/ai4reason/eprover/tree/WPO`
[4] The evaluation took around 2 days employing 32 cores of Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz with 128 GB memory in total.

	Auto	LPO		KBO		WPO		*all*	
	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.

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.

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 asses 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% to 40% slower. The speed of WPO varies, but in average it is from 40% to 140% slower than 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.

6 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.

As future work, we would like to experiment with further algebras, additional coefficient settings, and 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, as well as perform more thorough evaluations.

References

1. Czajka, Ł., Kaliszyk, C.: Hammer for Coq: Automation for dependent type theory. J. Autom. Reasoning (2018)
2. Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: Computer-Aided Verification (CAV 2013). LNCS, vol. 8044, pp. 1–35. Springer (2013)
3. Löchner, B.: Things to know when implementing KBO. J. Autom. Reasoning 36(4), 289–310 (2006)
4. Löchner, B.: Things to know when implementing LPO. International Journal on Artificial Intelligence Tools 15(1), 53–80 (2006)
5. McCune, W.: Solution of the Robbins problem. J. Autom. Reasoning 19(3), 263–276 (1997)
6. Middeldorp, A.: Term rewriting lecture notes (2017), 9th International School on Rewriting (ISR 2017)
7. Schulz, S.: System description: E 1.8. In: Logic for Programming, Artificial Intelligence (LPAR 2013). LNCS, vol. 8312, pp. 735–743. Springer (2013)
8. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. Journal of Automated Reasoning 59(4), 483–502 (2017)
9. Sutcliffe, G.: The 8th IJCAR automated theorem proving system competition - CASC-J8. AI Communications 29(5), 607–619 (2016)
10. Yamada, A., Kusakari, K., Sakabe, T.: Unifying the Knuth-Bendix, recursive path and polynomial orders. In: PPDP. pp. 181–192. ACM (2013)
11. Yamada, A., Kusakari, K., Sakabe, T.: A unified ordering for termination proving. Sci. Comput. Program. 111, 110–134 (2015)