Improving Homeomorphic Embedding for Online Termination

Michael Leuschel*

Department of Electronics and Computer Science, University of Southampton, UK
Department of Computer Science, K.U. Leuven, Belgium
DIKU, University of Copenhagen, Denmark
mal@ecs.soton.ac.uk
www: http://www.ecs.soton.ac.uk/~mal

Abstract. Well-quasi orders in general, and homeomorphic embedding in particular, have gained popularity to ensure online termination of program analysis, specialisation and transformation techniques. It has been recently shown that the homeomorphic embedding relation is strictly more powerful than a large class of involved well-founded approaches. In this paper we provide some additional investigations on the power of homeomorphic embedding. We, however, also illustrate that the homeomorphic embedding relation suffers from several inadequacies in contexts where logical variables arise. We therefore present new, extended homeomorphic embedding relations to remedy this problem.
Keywords: Termination, Well-quasi orders, Program Analysis, Specialisation and Transformation, Logic Programming, Functional & Logic Programming.

1 Introduction

The problem of ensuring termination arises in many areas of computer science and a lot of work has been devoted to proving termination of term rewriting systems (e.g. [9–11, 40] and references therein) or of logic programs (e.g. [7, 41] and references therein). It is also an important issue within all areas of program analysis, specialisation and transformation: one usually strives for methods which are guaranteed to terminate. One can basically distinguish between two kinds of techniques for ensuring termination:

- *offline* (or *static*) techniques, which prove or ensure termination of a program or process *beforehand* without any kind of execution, and
- *online* (or *dynamic*) techniques, which ensure termination of a process *during* its execution.

Offline approaches have less information at their disposal but do not require runtime intervention (which might be impossible). Which of the two approaches is taken depends entirely on the application area. For instance, static termination analysis of logic programs [7, 41] falls within the former context, while termination of program specialisation, transformation or analysis is often ensured in an online manner.

* Part of the work was done while the author was Post-doctoral Fellow of the Fund for Scientific Research - Flanders Belgium (FWO).

This paper is primarily aimed at studying and improving online termination techniques. Let us examine the case of partial deduction [34, 12, 26] — an automatic technique for specialising logic programs. Henceforth we suppose some familiarity with basic notions in logic programming [4, 33].

Partial deduction based upon the Lloyd and Shepherdson framework [34] generates (possibly incomplete) SLDNF-trees for a set $\mathcal{A}$ of atoms. The specialised program is extracted from these trees by producing one clause (called a resultant) for every non-failing branch. The resolution steps within the SLDNF-trees — often referred to as *unfolding* steps — are those that have been performed beforehand, justifying the hope that the specialised program is more efficient.

Now, to ensure termination of partial deduction two issues arise [12, 39]. One is called the *local termination* problem, corresponding to the fact that each generated SLDNF-tree should be finite. The other is called the *global termination* problem, meaning that the set $\mathcal{A}$ should contain only a finite number of atoms. A similar classification can be done for most other program specialisation techniques (cf., e.g., [31]).

Below we mainly use local termination to illustrate our concepts. (As shown in [39] the atoms in $\mathcal{A}$ can be structured into a global tree and methods similar to the one for local termination can be used to ensure global termination. See also [45] for a very general, language independent, framework for termination.)

However, the discussions and contributions of the present paper are also (immediately) applicable in the context of analysis, specialisation and transformation techniques in general, especially when applied to computational paradigms, such as logic programming, constrained logic programming, conditional term rewriting, functional programming and functional & logic programming. We also believe that our discussions are relevant for other areas, such as infinite *model checking* or *theorem proving*, where termination has to be insured in non-trivial ways.

One, albeit ad-hoc, way to solve the local termination problem is to simply impose an arbitrary *depth bound*. Such a depth bound is of course not motivated by any property, structural or otherwise, of the program or goal under consideration. In the context of local termination, the depth bound will therefore typically lead either to too little or too much unfolding.

Another approach, often used in partial evaluation of functional programs [19], is to (only) expand a tree while it is *determinate* (i.e. it only has one non-failing branch). However, this approach can be very restrictive and in itself does not guarantee termination, as there can be infinitely failing determinate computations at specialisation time.

Well-founded Orders Luckily, more refined approaches to ensure local termination exist. The first non-ad-hoc methods [6, 38, 37, 36] in logic and [43, 52] functional programming were based on *well-founded orders*, inspired by their usefulness in the context of static termination analysis. These techniques ensure termination, while at the same time allowing unfolding related to the structural

aspect of the program and goal to be specialised, e.g., permitting the consumption of static input within the atoms of $\mathcal{A}$.

Definition 1. (wfo) *A* (strict) partial order $>_S$ *on a set* S *is an anti-reflexive, anti-symmetric and transitive binary relation on* $S \times S$*. A sequence of elements* $s_1, s_2, \ldots$ *in* S *is called* admissible wrt $>_S$ *iff* $s_i > s_{i+1}$*, for all* $i \geq 1$*.*
We call $>_S$ *a* well-founded order (wfo) *iff there are no infinite admissible sequences wrt* $>_S$*.*

Now, to ensure local termination for instance, one has to find a sensible well-founded order on atoms and then only allow SLDNF-trees in which the sequence of selected atoms is admissible wrt the well-founded order. If an atom that we want to select is not strictly smaller than its ancestors, we either have to select another atom or stop unfolding altogether.

Example 1. Let P be the *reverse* program using an accumulator:

$rev([\,], Acc, Acc) \leftarrow$
$rev([H|T], Acc, Res) \leftarrow rev(T, [H|Acc], Res)$

A simple well-founded order on atoms of the form $rev(t_1, t_2, t_3)$ might be based on comparing the termsize (i.e., the number of function and constant symbols) of the first argument. We then define the wfo on atoms by:

$$rev(t_1, t_2, t_3) > rev(s_1, s_2, s_3) \text{ iff } term\_size(t_2) > term\_size(s_2).$$

Based on that wfo, the goal $\leftarrow rev([a, b|T], [\,], R)$ can be unfolded into the goal $\leftarrow rev([b|T], [a], R)$ and further into $\leftarrow rev(T, [b, a], R)$ because the termsize of the first argument strictly decreases at each step (even though the overall termsize does not decrease). However, $\leftarrow rev(T, [b, a], R)$ cannot be further unfolded into $\leftarrow rev(T', [H', b, a], R)$ because there is no such strict decrease.

Much more elaborate techniques exist [6, 38, 37, 36], which, e.g., split the expressions into classes, use lexicographical ordering on subsequences of the arguments and even continuously refine the orders during the unfolding process. These works also present some further refinements on *how to apply* wfo's, especially in the context of partial deduction. For instance, instead of requiring a decrease wrt every ancestor, one can only request a decrease wrt the *covering ancestors*, i.e. one only compares with the ancestor atoms from which the current atom descends (via resolution). (Most of these refinements can also be applied to other approaches, notably the one we will present in the next section.)

However, it has been felt by several researchers that well-founded orders are sometimes too rigid or (conceptually) too complex in an online setting. Recently, *well-quasi orders* have therefore gained popularity to ensure online termination of program manipulation techniques [5, 44, 46, 30, 31, 14, 20, 2, 22, 50, 1, 8]. In [28] the reasons behind this move to well-quasi orders have been formally investigated. Notably, [28] shows that a rather simple well-quasi approach—the *homeomorphic embedding* relation— is strictly more powerful than a large class of involved well-founded approaches. Nonetheless, despite its power, we will show that the homeomorphic embedding is still unsatisfactory when it comes to variables. This paper aims at improving this situation by developing more adequate refinements of the homeomorphic embedding relation.

This paper is structured as follows. In Sections 2 and 3 we provide a gentle introduction to well-quasi orders and summarise the main results of [28]. In Section 4 we provide some additional investigation, discussing the concept of "near-foundedness" [36]. In Section 5 we show that, despite its power, the homeomorphic embedding is still unsatisfactory when it comes to variables. We provide a first solution, which we then improve in Section 6, notably to be able to cope with infinite alphabets.

2 Well-quasi orders and homeomorphic embedding

Formally, well-quasi orders can be defined as follows.

Definition 2. (quasi order) *A* quasi order $\geq_S$ *on a set* S *is a reflexive and transitive binary relation on* $S \times S$.

Henceforth, we will use symbols like $<, >$ (possibly annotated by some subscript) to refer to strict partial orders and $\leq, \geq$ to refer to quasi orders and binary relations. We will use either "directionality" as is convenient in the context. We also define an *expression* to be either a *term* (built-up from variables and function symbols of arity ≥ 0) or an *atom* (a predicate symbol applied to a, possibly empty, sequence of terms), and then treat predicate symbols as function symbols, but suppose that no confusion between function and predicate symbols can arise (i.e., predicate and function symbols are distinct).

Definition 3. (wbr,wqo) *Let* $\leq_S$ *be a binary relation on* $S \times S$. *A sequence of elements* $s_1, s_2, \ldots$ *in* S *is called* admissible wrt $\leq_S$ *iff there are no* $i < j$ *such that* $s_i \leq_S s_j$. *We say that* $\leq_S$ *is a* well-binary relation (wbr) *on* S *iff there are no infinite admissible sequences wrt* $\leq_S$. *If* $\leq_S$ *is a quasi order on* S *then we also say that* $\leq_S$ *is a* well-quasi order (wqo) *on* S.

Observe that, in contrast to wfo's, non-comparable elements are allowed within admissible sequences. An admissible sequence is sometimes called *bad* while a non-admissible one is called *good*. A well-binary relation is then such that all infinite sequences are good. There are several other equivalent definitions of well-binary relations and well-quasi orders. Higman [17] used an alternate definition of well-quasi orders in terms of the "finite basis property" (or "finite generating set" in [21]). Both definitions are equivalent by Theorem 2.1 in [17]. A different (but also equivalent) definition of a wqo is(e.g., [23, 51]): A quasi-order $\leq_V$ is a wqo iff for all quasi-orders $\preceq_V$ which contain $\leq_V$ (i.e. $v \leq_V v' \Rightarrow v \preceq_V v'$) the corresponding strict partial order $\prec_V$ is a wfo. This property has been exploited in the context of *static* termination analysis to dynamically construct well-founded orders from well-quasi ones and led to the initial use of wqo's in the offline setting [9, 10]. The use of well-quasi orders in an *online* setting has only emerged recently (it is mentioned, e.g., in [5] but also [44]) and [28] provides the first formal comparison.[1] Furthermore, in the online setting, transitivity of

[1] There has been some comparison between wfo's and wqo's in the offline setting, e.g., in [40] it is argued that (for "simply terminating" rewrite systems) approaches based upon quasi-orders are less interesting than ones based upon a partial orders.

a wqo is not really interesting (because one does not have to generate wfo's) and one can therefore drop this requirement, leading to the use of wbr's. Later on in Sections 5 and 6 we will actually develop wbr's which are not wqo's.

An interesting wqo is the *homeomorphic embedding* relation $\trianglelefteq$, which derives from results by Higman [17] and Kruskal [21]. It has been used in the context of term rewriting systems in [9, 10], and adapted for use in supercompilation [49] in [46]. Its usefulness as a stop criterion for partial evaluation is also discussed and advocated in [35]. Some complexity results can be found in [48] and [16] (also summarised in [35]).

The following is the definition from [46], which adapts the pure homeomorphic embedding from [10] by adding a rudimentary treatment of variables.

Definition 4. *The* (pure) homeomorphic embedding *relation* $\trianglelefteq$ *on expressions is inductively defined as follows (i.e.* $\trianglelefteq$ *is the least relation satisfying the rules):*

1. $X \trianglelefteq Y$ *for all variables* X, Y
2. $s \trianglelefteq f(t_1, \ldots, t_n)$ *if* $s \trianglelefteq t_i$ *for some* i
3. $f(s_1, \ldots, s_n) \trianglelefteq f(t_1, \ldots, t_n)$ *if* $\forall i \in \{1, \ldots, n\} : s_i \trianglelefteq t_i$.

The second rule is sometimes called the *diving* rule, and the third rule is sometimes called the *coupling* rule (notice that n is allowed to be 0). When $s \trianglelefteq t$ we also say that s is *embedded in* t or t is *embedding* s. By $s \triangleleft t$ we denote that $s \trianglelefteq t$ and $t \not\trianglelefteq s$.

The intuition behind the above definition is that $A \trianglelefteq B$ iff A can be obtained from B by "striking out" certain parts, or said another way, the structure of A reappears within B. Indeed, just applying the coupling rule 3 we get syntactic identity for ground expressions, rule 1 just confounds all variables, and the diving rule 2 allows to ignore a part (namely $f(t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n)$) of the right-hand term.

Example 2. We have $p(a) \trianglelefteq p(f(a))$ and indeed $p(a)$ can be obtained from $p(f(a))$ by "striking out" the f; see Fig. 1. Observe that the "striking out" corresponds to the application of the diving rule 2 and that we even have $p(a) \triangleleft p(f(a))$. We also have, e.g., that: $X \trianglelefteq X$, $p(X) \triangleleft p(f(Y))$, $p(X, X) \trianglelefteq p(X, Y)$ and $p(X, Y) \trianglelefteq p(X, X)$.

Proposition 1. $\trianglelefteq$ *is a wqo on the set of expressions over a finite alphabet.*

To ensure, e.g., local termination of partial deduction, we have to ensure that the constructed SLDNF-trees are such that the selected atoms do *not embed* any of their ancestors (when using a well-founded order as in Example 1, we had to require a *strict decrease* at every step). If an atom that we want to select embeds one of its ancestors, we either have to select another atom or stop unfolding altogether. For example, based on $\trianglelefteq$, the goal $\leftarrow rev([a, b|T], [\,], R)$ of Example 1 can be unfolded into $\leftarrow rev([b|T], [a], R)$ and further into $\leftarrow rev(T, [b, a], R)$ as $rev([a, b|T], [\,], R) \not\trianglelefteq rev([b|T], [a], R)$, $rev([a, b|T], [\,], R) \not\trianglelefteq rev(T, [b, a], R)$ and $rev([b|T], [a], R) \not\trianglelefteq rev(T, [b, a], R)$. However, $\leftarrow rev(T, [b, a], R)$ cannot be further unfolded into $\leftarrow rev(T', [H', b, a], R)$ as $rev(T, [b, a], R) \trianglelefteq rev(T', [H', b, a], R)$.

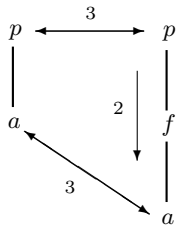

Fig. 1. Illustrating Example 2

Observe that, in contrast to Example 1, we did not have to choose how to measure which arguments. We further elaborate on the inherent flexibility of $\trianglelefteq$ in the next section.

The homeomorphic embedding relation is also useful for handling structures other than expressions. It has, e.g., been successfully applied in [30, 26, 31] to detect (potentially) non-terminating sequences of characteristic trees. Also, $\trianglelefteq$ seems to have the desired property that very often only "real" loops are detected and that they are detected at the earliest possible moment (see [35]).

3 Comparing wbr's and wfo's

In this section we summarise the main results of [28].

It follows from Definitions 1 and 3 that if $\leq_V$ is a wqo then $<_V$ (defined by $v_1 <_V v_2$ iff $v_1 \leq_V v_2 \wedge v_1 \not\geq_V v_2$) is a wfo, but not vice versa. The following shows how to obtain a wbr from a wfo.

Lemma 1. *Let $<_V$ be a well-founded order on V. Then $\preceq_V$, defined by $v_1 \preceq_V v_2$ iff $v_1 \not> _V v_2$, is a wbr on V. Furthermore, $<_V$ and $\preceq_V$ have the same set of admissible sequences.*

This means that, in an online setting, the approach based upon wbr's is in theory at least as powerful as the one based upon wfo's. Further below we will actually show that wbr's are strictly more powerful.

Observe that $\preceq_V$ is not necessarily a wqo: transitivity is not ensured as $t_1 \not> t_2$ and $t_2 \not> t_3$ do not imply $t_1 \not> t_3$. Let, e.g., $s < t$ denote that s is strictly more general than t. Then $<$ is a wfo [18] but $p(X, X, a) \not> p(X, Z, b)$ and $p(X, Z, b) \not> p(X, Y, a)$ even though $p(X, X, a) > p(X, Y, a)$.

Let us now examine the power of one particular wqo, the earlier defined $\trianglelefteq$.

The homeomorphic embedding $\trianglelefteq$ relation is very flexible. It will for example, when applied to the sequence of covering ancestors, permit the full unfolding of most terminating Datalog programs, the quicksort or even the mergesort program when the list to be sorted is known (the latter poses problems to some static termination analysis methods [41, 32]; for some experiments see [28]). Also, the

produce-consume example from [36] requires rather involved techniques (considering the context) to be solved by wfo's. Again, this particular example poses no problem to $\trianglelefteq$ (cf. [28]).

The homeomorphic embedding $\trianglelefteq$ is also very powerful in the context of metaprogramming. Notably, it has the ability to "penetrate" layers of (non-ground) meta-encodings (cf. [27] and [13] for further discussions on that matter; cf. also the appendix of [28] for some computer experiments). For instance, $\trianglelefteq$ will admit the following sequences (where, among others, Example 1 is progressively wrapped into "vanilla" metainterpreters counting resolution steps and keeping track of the selected predicates respectively):

Sequence
$rev([a,b\|T],[\,],R) \rightsquigarrow rev([b\|T],[a],R)$
$solve(rev([a,b\|T],[\,],R),0) \rightsquigarrow solve(rev([b\|T],[a],R),s(0))$
$solve'(solve(rev([a,b\|T],[\,],R),0),[\,]) \rightsquigarrow solve'(solve(rev([b\|T],[a],R),s(0)),[rev])$
$path(a,b,[\,]) \rightsquigarrow path(b,a,[a])$
$solve'(solve(path(a,b,[\,]),0),[\,]) \rightsquigarrow solve'(solve(path(b,a,[a]),s(0)),[rev])$

Again, this is very difficult for wfo's and requires refined and involved techniques (of which to our knowledge no implementation in the online setting exists). For example, to admit the third sequence we have to measure something like the "termsize of the first argument of the first argument of the first argument." For the fifth sequence this gets even more difficult.

We have intuitively demonstrated the usefulness of $\trianglelefteq$ and that it is often more flexible than wfo's. But can we prove some "hard" results? It turns out that we can and [28] establishes that — in the online setting — $\trianglelefteq$ is strictly more generous than a large class of refined wfo's containing the following:

Definition 5. *A well-founded order $\prec$ on expressions is said to be* monotonic *iff the following rules hold:*

1. $X \not\succ Y$ *for all variables* X, Y,
2. $s \not\succ f(t_1,\ldots,t_n)$ *whenever* f *is a function symbol and* $s \not\succ t_i$ *for some* i *and*
3. $f(s_1,\ldots,s_n) \not\succ f(t_1,\ldots,t_n)$ *whenever* $\forall i \in \{1,\ldots,n\} : s_i \not\succ f_i$.

Observe that point 2 need not hold for predicate symbols and that point 3 implies that $c \not\succ c$ for all constant and proposition symbols c. There is also a subtle difference between monotonic wfo's as of Definition 5 and wfo's which possess the replacement property (such orders are called rewrite orders in [40] and monotonic in [9]). More on that below.

[28] shows that most of the wfo's used in online practice are actually monotonic:

- Definitions 3.4 of [6], 3.2 of [38] and 2.14 of [37] all sum up the number of function symbols (i.e. termsize) of a subset of the argument positions of atoms. The algorithms only differ in the way of choosing the positions to measure. The early algorithms measure the input positions, while the later ones dynamically refine the argument positions to be measured. All these wfo's are monotonic.

– Definitions 3.2 of [37] as well as 8.2.2 of [36] use the lexicographical order on the termsizes of some selected argument positions. These wfo's are also monotonic, as proven in [28].

The only non-monotonic wfo in that collection of articles is the one defined specifically for metainterpreters in Definition 3.4 of [6] (also in Section 8.6 of [36]) which uses selector functions to focus on subterms to be measured.

We now adapt the class of simplification orderings from term rewriting systems. The power of this class of wfo's is also subsumed by $\trianglelefteq$.

Definition 6. *A* simplification ordering *is a wfo $\prec$ on expressions which satisfies*

1. $s \prec t \Rightarrow f(t_1, \ldots, s, \ldots, t_n) \prec f(t_1, \ldots, t, \ldots, t_n)$ (replacement property),
2. $t \prec f(t_1, \ldots, t, \ldots, t_n)$ (subterm property) *and*
3. $s \prec t \Rightarrow s\sigma \prec t\gamma$ *for all variable only substitutions σ and γ* (invariance under variable replacement).

The third rule of the above definition is new wrt term-rewriting systems and implies that all variables must be treated like a unique new constant. It turns out that a lot of powerful wfo's are simplification orderings [9, 40]: recursive path ordering, Knuth-Bendix ordering or lexicographic path ordering, to name just a few. However, not all monotonic wfo's are simplification orderings and there are wfo's which are simplification orderings but are not monotonic.

Proposition 2. *Let $\prec$ be a wfo on expressions. Then any admissible sequence wrt $\prec$ is also an admissible sequence wrt $\trianglelefteq$ if $\prec$ is* **a)** *monotonic or if it is* **b)** *a simplification ordering.*

This means that the admissible sequences wrt $\trianglelefteq$ are a superset of the union of all admissible sequences wrt simplification orderings and monotonic wfo's. In other words, no matter how much refinement we put into an approach based upon monotonic wfo's or upon simplification orderings we can only expect to approach $\trianglelefteq$ in the limit. But by a simple example we can even dispel that hope.

Example 3. Take the sequence $\delta = f(a), f(b), b, a$. This sequence is admissible wrt $\trianglelefteq$ as $f(a) \not\trianglelefteq f(b)$, $f(a) \not\trianglelefteq b$, $f(a) \not\trianglelefteq a$, $f(b) \not\trianglelefteq b$, $f(b) \not\trianglelefteq a$ and $a \not\trianglelefteq b$. However, there is no monotonic wfo $\prec$ which admits this sequence. More precisely, to admit δ we must have $f(a) \succ f(b)$ as well as $b \succ a$, i.e. $a \not\succ b$. But this violates rule 3 of Definition 5 and $\prec$ cannot be monotonic. This also violates rule 1 of Definition 6 and $\prec$ cannot be a simplification ordering.

These new results relating $\trianglelefteq$ to monotonic wfo's shed light on $\trianglelefteq$'s usefulness in the context of ensuring online termination.

But of course the admissible sequences wrt $\trianglelefteq$ are *not* a superset of the union of all admissible sequences wrt *any* wfo.[2] For instance the list-length norm $\|.\|_{llen}$ is not monotonic, and indeed we have for $t_1 = [1, 2, 3]$ and $t_2 = [[1, 2, 3], 4]$ that

[2] Otherwise $\trianglelefteq$ could not be a wqo, as *all* finite sequences without repetitions are admissible wrt some wfo (map last element to 1, second last element to 2, ...).

$\|t_1\|_{llen} = 3 > \|t_2\|_{llen} = 2$ although $t_1 \trianglelefteq t_2$. So there are sequences admissible wrt list-length but not wrt $\trianglelefteq$. The reason is that $\|.\|_{llen}$ in particular and non-monotonic wfo's in general can completely ignore certain parts of the term, while $\trianglelefteq$ will always inspect that part. E.g., if we have $s \succ f(\ldots t \ldots)$ and $\succ$ ignores the subterm t then it will also be true that $s \succ f(\ldots s \ldots)$ while $s \trianglelefteq f(\ldots s \ldots)$,[3] i.e. the sequence $s, f(\ldots s \ldots)$ is admissible wrt $\succ$ but not wrt $\trianglelefteq$.

Of course, for any wfo (monotonic or not) one can devise a wbr (cf. Lemma 1) which has the same admissible sequences. Still there are some feats that are easily attained, even by using $\trianglelefteq$, but which *cannot* be achieved by a wfo approach (monotonic or not). Take the sequences $S_1 = p([\,],[a]), p([a],[\,])$ and $S_2 = p([a],[\,]), p([\,],[a])$. Both of these sequences are admissible wrt $\trianglelefteq$. This illustrates the flexibility of using well-quasi orders compared to well-founded ones in an online setting, as there exists *no* wfo (monotonic or not) which will admit *both* these sequences. It, however, also illustrates why, when using a wqo in that way, one has to compare with every predecessor state of a process. Otherwise one can get infinite derivations of the form $p([a],[\,]) \rightarrow p([\,],[a]) \rightarrow p([a],[\,]) \rightarrow \ldots$.[4] In other words, for wqo's, the composition $s_1, \ldots, s_m$ of two admissible sequences $s_1, \ldots, s_n$ and $s_n, s_{n+1}, \ldots, s_m$ is not necessarily admissible. This is in contrast to wfo's.

Finally, one could argue that it is possible to extend the power of the wfo-approach by defining wfo's over histories (i.e., sequences) instead of the individual elements. This is, however, not the way that wfo's are used in practice and one is faced with the difficulty of defining a sensible order on sequences. Of course, one could always use a wqo $\preceq$ to define such an order: $s_1, \ldots, s_n > s_1, \ldots, s_n, s_{n+1}$ iff $\forall i \in \{1, \ldots, n\} : s_i \not\preceq s_{n+1}$. One would thus get an approach with *exactly* the same power and complexity as the wqo-approach.

4 Nearly-Foundedness and Over-Eagerness

In [37], as well as in Section 8.2.4 of [36], a technique for wfo's is formally introduced, based upon *nearly-foundedness*. As already mentioned earlier, some of the techniques in [6, 38, 37, 36] start out with a very coarse wfo $<_1$ which is then continuously refined, enabling a clever choice of weights for predicates and their respective arguments (deciding beforehand upon appropriate weights can be extremely difficult or impossible; see examples in Section 3). For example we might have that $<_1$ is based upon measuring the sum of the termsize of all arguments and the process of refining consists in dropping one or more arguments. For example, suppose that we have some sequence $s_1, s_2, \ldots, s_i$ which is

[3] Observe that if f is a predicate symbols then $f(\ldots s \ldots)$ is not a valid expression, which enabled us to ignore arguments to predicates.

[4] When using a wfo one has to compare only to the closest predecessor [37], because of the transitivity of the order and the strict decrease enforced at each step. However, wfo's are usually extended to incorporate variant checking and then require inspecting every predecessor anyway (though only when there is no strict weight decrease, see, e.g., [36, 37]).

admissible wrt the initial wfo $<_1$ but where $s_{i+1} \not<_1 s_i$ with $s_{i+1} = p(a, s(s(b)))$ and $s_i = p(s(a), b)$. In that case we can move to a refinement $<_2$ of $<_1$ in which only the termsize of the first argument is measured, enabling the move from s_i to s_{i+1} (as $s_{i+1} <_2 s_i$). This, however, does not guarantee that the *whole* sequence $s_1, s_2, \ldots, s_i, s_{i+1}$ is admissible wrt $<_2$. E.g., for the sequence $p(s(a), s(s(s(c)))), p(s(a), b), p(a, s(s(b)))$ with $i = 2$ we have $s_2 \not<_2 s_1$ even though $s_2 <_1 s_1$.

To solve this problem, the earlier algorithms verified that a refinement keeps the whole sequence admissible (otherwise it was disallowed). The problem with this approach is that re-checking the entire sequence can be expensive. [37, 36] therefore advocates another solution: not re-checking the entire sequence on the grounds that it does not threaten termination (provided that the refinements themselves are well-founded). This leads to sequences $s_1, s_2, \ldots$ which are not well-founded but *nearly-founded* [37, 36] meaning that $s_i \not< s_j$ only for a finite number of pairs (i, j) with $i > j$.

In summary, the motivation for nearly-foundedness lies in speeding up the construction of admissible sequences (not re-scanning the initial sequence upon refinement). As a side-effect, this approach will admit more sequences and, e.g., solve some of our earlier examples (as a suitably large depth bound would as well). However, from a theoretical point of view, we argue below that nearly-foundedness is difficult to justify and somewhat unsatisfactory.

First, we call a technique *over-eager* if it admits sequences which are not admitted by the variant test (i.e., it admits sequences containing variants). We call such a technique *strictly over-eager* if it admits sequences which contain more than 1 occurrence of the same syntactic term.

A depth bound based technique is strictly over-eager, which is obviously a very undesirable property indicating some ad-hoc behaviour. The same can (almost always)[5] also be said for over-eagerness. For instance, in the context of partial deduction or unfold/fold program transformation, over-eager unfolding will "hide" possibilities for perfect folding (namely the variants) and also lead to too large specialised programs. An approach based upon homeomorphic embedding is not over-eager nor is any other wfo/wqo based approach which does not distinguish between variants. However, as we show below, using nearly-foundedness leads to *strict* over-eagerness.

Let us first describe the way nearly-founded sequences are constructed in [37, 36]. First, we define a well-founded order $\prec_{\mathcal{W}}$ acting on a set $\mathcal{W}$ of well-founded orders on expressions. To construct admissible sequences of expressions wrt $\prec_{\mathcal{W}}$ we start by using one wfo $<_1 \in \mathcal{W}$ until the sequence can no longer be extended. Once this is the case we can use another wfo $<_2 \in \mathcal{W}$ which admits the offending step, provided that $<_2 \prec_{\mathcal{W}} <_1$. It is *not* required that the whole initial sequence is admissible wrt $<_2$, just the last step (not admitted by $<_1$). We can now continue expanding the sequence until we again reach an offending step, where we can then try to refine $<_2$ into some $<_3$ with $<_3 \prec_{\mathcal{W}} <_2$, and so on, until no further expansion is possible.

[5] See discussion in Section 5 concerning the variant test *on covering ancestors*.

Example 4. Take the following program.

$p([a,a|T],[a|Y]) \leftarrow p(T,Y)$
$p([b,b,b|T],Y) \leftarrow p(T,[b,b|Y])$
$p(T,[b,b|Y]) \leftarrow p([a,a,b,b,b|T],[a|Y])$

Let us now define the following well-founded orders, where $\|t\|_{ts}$ denotes the termsize of t:

$<_{\{1,2\}}$: $p(s_1,s_2) < p(t_1,t_2)$ iff $\|s_1\|_{ts} + \|s_2\|_{ts} < \|t_1\|_{ts} + \|t_2\|_{ts}$
$<_{\{1\}}$: $p(s_1,s_2) < p(t_1,t_2)$ iff $\|s_1\|_{ts} < \|t_1\|_{ts}$
$<_{\{2\}}$: $p(s_1,s_2) < p(t_1,t_2)$ iff $\|s_2\|_{ts} < \|t_2\|_{ts}$

We also define the wfo $\prec_{\mathcal{W}}$ on the above well-founded orders: $<_{\{1\}} \prec_{\mathcal{W}} <_{\{1,2\}}$ and $<_{\{2\}} \prec_{\mathcal{W}} <_{\{1,2\}}$. In other words we can refine $<_{\{1,2\}}$ into $<_{\{1\}}$ or $<_{\{2\}}$, which in turn cannot be further refined.

We can now construct the following admissible sequence in which two terms ($p([a,a,b,b,b],[a])$ and $p([b,b,b],[\,])$) appear twice:

$p([a,a,b,b,b],[a])$
$\quad\downarrow <_{\{1,2\}}$
$p([b,b,b],[\,])$
$\quad\downarrow <_{\{1,2\}}$
$p([\,],[b,b])$
$\quad\downarrow <_{\{2\}}$ (refinement)
$\underline{p([a,a,b,b,b],[a])}$
$\quad\downarrow <_{\{2\}}$
$\underline{p([b,b,b],[\,])}$

Example 4 thus proves that nearly-foundedness may result in strict over-eagerness. (As a side-effect, the example also shows that the nearly-foundedness approach cannot be mapped to a wfo-approach, however involved it might be.) Although it is unclear how often such situations will actually arise in practice, we believe that the strict over-eagerness is just one of the (mathematically) unsatisfactory aspects of nearly-foundedness.

5 A more refined treatment of variables

While $\trianglelefteq$ has a lot of desirable properties it still suffers from some drawbacks. Indeed, as can be observed in Example 2, the homeomorphic embedding relation $\trianglelefteq$ as defined in Definition 4 is rather crude wrt variables. In fact, all variables are treated as if they were the same variable, a practice which is clearly undesirable in a logic programming context. Intuitively, in the above example, $p(X,Y) \trianglelefteq p(X,X)$ can be justified (see, however, Definition 8 below), while $p(X,X) \trianglelefteq p(X,Y)$ is not. Indeed $p(X,X)$ can be seen as standing for something like $and(eq(X,Y),p(X,Y))$, which embeds $p(X,Y)$, but the reverse does not hold.

Secondly, $\trianglelefteq$ behaves in quite unexpected ways in the context of generalisation, posing some subtle problems wrt the termination of a generalisation process.

Example 5. Take for instance the following generalisation algorithm, which appears (in disguise) in a lot of partial deduction algorithms (e.g., [30, 26, 31]). (In that context $\mathcal{A}$ is the set of atoms for which SLDNF-trees have already been constructed while $\mathcal{B}$ are the atoms in the leaves of these trees. The goal of the algorithm is then to extend $\mathcal{A}$ such that all leaf atoms are covered.)

Input: two finite sets $\mathcal{A}, \mathcal{B}$ of atoms
Output: a finite set $\mathcal{A}' \supseteq \mathcal{A}$ s.t. every atom in $\mathcal{B}$ is an instance of an atom in $\mathcal{A}'$
Initialisation: $\mathcal{A}' := \mathcal{A}$, $\mathcal{B}' := \mathcal{B}$
while $\mathcal{B}' \neq \emptyset$ **do**
  remove an element B from $\mathcal{B}'$
  if B is not an instance of an element in $\mathcal{A}'$ **then**
    if $\exists A \in \mathcal{A}'$ such that $A \trianglelefteq B$ **then**
      add $msg(A, B)$ to $\mathcal{B}'$
    else add B to $\mathcal{A}'$

The basic idea of the algorithm is to use $\trianglelefteq$ to keep the set $\mathcal{A}'$ finite in the limit. However, although the above algorithm will indeed keep $\mathcal{A}'$ finite, it still does not terminate. Take for example $\mathcal{A} = \{p(X, X)\}$ and $\mathcal{B} = \{p(X, Y)\}$. We will remove $B = p(X, Y)$ from $\mathcal{B}' = \{p(X, Y)\}$ in the first iteration of the algorithm and we have that B is not an instance of $p(X, X)$ and also that $p(X, X) \trianglelefteq p(X, Y)$. We therefore calculate the $msg(\{p(X, X), p(X, Y)\}) = p(X, Y)$ and we have a loop (we get $\mathcal{B}' = \{p(X, Y)\}$).

To remedy these problems, [30, 26, 31] introduced the so called strict homeomorphic embedding as follows:

Definition 7. *Let A, B be expressions. Then B* (strictly homeomorphically) embeds *A, written as $A \trianglelefteq^+ B$, iff $A \trianglelefteq B$ and A is not a strict instance of B.*[6]

Example 6. We now still have that $p(X, Y) \trianglelefteq^+ p(X, X)$ but not $p(X, X) \trianglelefteq^+ p(X, Y)$. Note that still $X \trianglelefteq^+ Y$ and $X \trianglelefteq^+ X$.

A small experiment, specialising a query `rotate(X,X)` (using the ECCE system [25] with $\trianglelefteq$ and $\trianglelefteq^+$ respectively on conjunctions for global control; the `rotate` program, rotating a binary tree, can be found in [25]) demonstrates the interest of $\trianglelefteq$: when using $\trianglelefteq^+$ we obtain an overall speedup of 2.5 compared to "only" 2.0 using $\trianglelefteq$.

Notice that, if we replace $\trianglelefteq$ of Example 5 by $\trianglelefteq^+$ we no longer have a problem with termination (see [31, 26] for a termination proof of an Algorithm containing the one of Example 5).

The following is proven in [26, 31].

Theorem 1. *The relation $\trianglelefteq^+$ is a wbr on the set of expressions over a finite alphabet.*

[6] A is a strict instance of B iff there exists a substitution γ such that $A = B\gamma$ and there exists no substitution σ such that $B = A\sigma$.

Observe that $\trianglelefteq^+$ is not a wqo as it is not transitive: we have for example $p(X,X,Y,Y) \trianglelefteq^+ p(X,Z,Z,X)$ as well as $p(X,Z,Z,X) \trianglelefteq^+ p(X,X,Y,Z)$ but $p(X,X,Y,Y) \not\trianglelefteq^+ p(X,X,Y,Z)$. One might still feel dissatisfied with that definition for another reason. Indeed, although going from $p(X)$ to the instance $p(f(X))$ (and on to $p(f(f(X))),\ldots$) looks very dangerous, a transition from $p(X,Y)$ to $p(X,X)$ is often not dangerous, especially in a database setting. Take for example a simple Datalog program just consisting of the clause $p(a,b) \leftarrow p(X,X)$. Obviously P will terminate (i.e., fail finitely) for all queries but $\trianglelefteq^+$ will not allow the selection of $p(X,X)$ in the following derivation $\leftarrow \underline{p(X,Y)} \rightsquigarrow \leftarrow \underline{p(X,X)} \rightsquigarrow \leftarrow fail$ of $P \cup \{\leftarrow p(X,Y)\}$, hence preventing full unfolding and the detection of finite failure. On the practical side this means that neither $\trianglelefteq^+$ nor $\trianglelefteq$ will allow full unfolding of all terminating queries to Datalog programs (although they will allow full unfolding of terminating ground queries to range-restricted Datalog programs). To remedy this, we can develop the following refinement of $\trianglelefteq^+$.

Definition 8. *We define $s \trianglelefteq_{var} t$ iff $s \triangleleft t$ or s is a variant of t.*

We have $p(X) \trianglelefteq_{var} p(f(X))$ and $p(X,Y) \trianglelefteq_{var} p(Z,X)$ but $p(X,X) \not\trianglelefteq_{var} p(X,Y)$ and $p(X,Y) \not\trianglelefteq_{var} p(X,X)$.

It is obvious that $\trianglelefteq_{var}$ is strictly more powerful than $\trianglelefteq^+$ (if t is strictly more general than s, then it is not a variant of s and it is also not possible to have $s \triangleleft t$). It thus also solves the generalisation problems of $\trianglelefteq$ (i.e., if we produce a strict generalisation g of some expression t, then $t \not\trianglelefteq_{var} g$). In addition $\trianglelefteq_{var}$ has the following property: if we have a query $\leftarrow Q$ to a Datalog program which left-terminates then the LD-tree for $\leftarrow Q$ is admissible in the sense that, for every selected literal L, we have $L \not\trianglelefteq_{var} A$ for all covering ancestors A of L. Indeed, whenever we have that $L \trianglelefteq A$ for two Datalog atoms L and A then we must also have $A \trianglelefteq L$ (as the diving rule of $\trianglelefteq$ cannot be applied). Thus, for Datalog, $\trianglelefteq_{var}$ is equivalent to the variant test. Now, if a derivation from $\leftarrow A$ leads to a goal $\leftarrow L_1,\ldots,L_n$ where L_1 is a variant of a covering ancestor A, then it is possible to repeat this left-to-right derivation again and again and we have a real loop.

Theorem 2. *The relation $\trianglelefteq_{var}$ is a wqo on the set of expression over a finite alphabet.*

The proof can be found in [27]. Observe that $\trianglelefteq_{var}$ is, like $\trianglelefteq$ and $\trianglelefteq^+$, not a wqo over an infinite alphabet. More on that in Section 6.

Discussion

Observe that the variant test is (surprisingly) not complete for Datalog in general (under arbitrary computation rules). Take the program just consisting of $p(b,a) \leftarrow p(X,Z), p(Z,Y)$. Then the query $\leftarrow p(X,Y)$ is finitely failed as the following derivation shows:

$\underline{p(X,Y)} \rightsquigarrow \underline{p(X',Z')}, p(Z',Y') \rightsquigarrow p(X'',Z''), p(Z'',Y''), \underline{p(a,Y')} \rightsquigarrow fail.$

However, at the second step (no matter what we do) we have to select a variant of the covering ancestor $p(X,Y)$ and the variant test will prevent full unfolding.

An alternate approach to Definitions 7 and 8 — at least for the aspect of treating variables in a more refined way — might be based on numbering variables using some mapping $\#(.)$ and then stipulating that $X \trianglelefteq^{\#} Y$ iff $\#(X) \leq \#(Y)$. For instance in [35] a de Bruijn numbering of the variables is proposed. Such an approach, however, has a somewhat ad hoc flavour to it. Take for instance the terms $p(X,Y,X)$ and $p(X,Y,Y)$. Neither term is an instance of the other and we thus have $p(X,Y,X) \trianglelefteq^{+} p(X,Y,Y)$ and $p(X,Y,Y) \trianglelefteq^{+} p(X,Y,X)$. Depending on the particular numbering we will either have that $p(X,Y,X) \not\trianglelefteq^{\#} p(X,Y,Y)$ or that $p(X,Y,Y) \not\trianglelefteq^{\#} p(X,Y,X)$, while there is no apparent reason why one expression should be considered smaller than the other.[7]

6 Extended homeomorphic embedding

Although $\trianglelefteq^{+}$ from Definition 7 has a more refined treatment of variables and has a much better behaviour wrt generalisation than $\trianglelefteq$ of Definition 4, it is still somewhat unsatisfactory.

One point is the restriction to a finite alphabet. Indeed, for a lot of practical logic programs, using, e.g., arithmetic built-ins or even $= ../2$, a finite alphabet is no longer sufficient. Luckily, the fully general definition of homeomorphic embedding as in [21, 10] remedies this aspect. It even allows function symbols with variable arity.[8] We will show below how this definition can be adapted to a logic programming context.

However, there is another unsatisfactory aspect of $\trianglelefteq^{+}$ (and $\trianglelefteq_{var}$). Indeed, it will ensure that $p(X,X) \not\trianglelefteq^{+} p(X,Y)$ while $p(X,X) \trianglelefteq p(X,Y)$ but we still have that, e.g., $f(a,p(X,X)) \trianglelefteq^{+} f(f(a),p(X,Y))$. In other words, the more refined treatment of variables is only performed at the top, but not recursively within the structure of the expressions. For instance, this means that $\trianglelefteq^{+}$ will handle `rotate(X,X)` much better than $\trianglelefteq$ but this improvement will often vanish when we add a layer of metainterpretation.

The following, new and more refined embedding relation remedies this somewhat ad hoc aspect of $\trianglelefteq^{+}$.

Definition 9. *Given a wbr $\preceq_F$ on the function symbols and a wbr $\preceq_S$ on sequences of expressions, we define the* extended homeomorphic embedding *on expressions by the following rules:*

1. $X \trianglelefteq^{*} Y$ *if X and Y are variables*
2. $s \trianglelefteq^{*} f(t_1,\ldots,t_n)$ *if $s \trianglelefteq^{*} t_i$ for some i*
3. $f(s_1,\ldots,s_m) \trianglelefteq^{*} g(t_1,\ldots,t_n)$ *if $f \preceq_F g$ and $\exists 1 \leq i_1 < \ldots < i_m \leq n$ such that $\forall j \in \{1,\ldots,m\} : s_j \trianglelefteq^{*} t_{i_j}$ and $\langle s_1,\ldots,s_m\rangle \preceq_S \langle t_1,\ldots,t_n\rangle$*

[7] [35] also proposes to consider all possible numberings (but leading to $n!$ complexity, where n is the number of variables in the terms to be compared). It is unclear how such a relation compares to $\trianglelefteq^{+}$ and $\trianglelefteq_{var}$.

[8] Which can also be seen as associative operators.

Observe that for rule 3 both n and m are allowed to be 0, but we must have $m \leq n$. In contrast to Definition 4 for $\trianglelefteq$, the left- and right-hand terms in rule 3 do not have to be of the same arity. The above rule therefore allows to ignore $n - m$ arguments form the right-hand term (by selecting the m indices $i_1 < \ldots < i_m$).

Furthermore, the left- and right-hand terms in rule 3 do not have to use the same function symbol: the function symbols are therefore compared using the wbr $\preceq_F$. If we have a finite alphabet, then equality is a wqo on the function symbols (one can thus obtain the pure homeomorphic embedding as a special case). In the context of, e.g., program specialisation or analysis, we know that the function symbols occurring within the program (text) and call to be analysed are of finite number. One might call these symbols *static* and all others *dynamic*. A wqo can the be obtained by defining $f \preceq g$ if either f and g are dynamic or if $f = g$. For particular types of symbols a natural wqo or wbr exists (e.g., for numbers) which can be used instead. Also, for associative symbols (such as $\wedge$) one can represent $c_1 \wedge \ldots \wedge c_n$ by $\wedge(c_1, \ldots, c_n)$ and then use equality up to arities (e.g., $\wedge/2 = \wedge/3$) for $\preceq_F$.

Example 7. If we take $\preceq_F$ to be equality up to arities and ignore $\preceq_S$ (i.e., define $\preceq_S$ to be always true) we get all the embeddings of $\trianglelefteq$, e.g., $p(a) \trianglelefteq^* p(f(a))$. But we also get $p(a) \trianglelefteq^* p(b, f(a), c)$ (while $p(a) \trianglelefteq p(b, f(a), c)$ does not hold), $\wedge(p(a), q(b)) \trianglelefteq^* \wedge(s, p(f(a)), r, q(b))$ and $\wedge(a, b, c) \trianglelefteq^* \wedge(a, b, c, d)$. One can see that $\trianglelefteq^*$ provides a convenient way to handle associative operators such as the conjunction $\wedge$. (Such a treatment of $\wedge$ has been used in [14, 20, 8] to ensure termination of conjunctive partial deduction. It might prove equally beneficial for constrained partial deduction [29].) Indeed, in the context of $\trianglelefteq$ one cannot use $\wedge$ with all possible arities and one has to use, e.g., a binary representation. But then whether $\wedge(a, b, c)$ is embedded in $\wedge(a, b, c, d)$ (which, given associativity, it is) depends on the particular representation: $\wedge(a, \wedge(b, c)) \trianglelefteq \wedge(a, \wedge(\wedge(b, c), d))$ but $\wedge(a, \wedge(b, c)) \not\trianglelefteq \wedge(\wedge(a, b), \wedge(c, d))$.

In the above definition we can now instantiate $\preceq_S$ such that it performs a more refined treatment of variables, as discussed in Section 5. For example we can define: $\langle s_1, \ldots, s_m \rangle \preceq_S \langle t_1, \ldots, t_n \rangle$ iff $\langle t_1, \ldots, t_n \rangle$ is not strictly more general than $\langle s_1, \ldots, s_m \rangle$. (Observe that this means that if $m \neq n$ then $\preceq_S$ will hold.) This relation is a wbr (by Lemma 1, as the strictly more general relation is a wfo [18]). Then, in contrast to $\trianglelefteq^+$ and $\trianglelefteq_{var}$, this refinement will be applied *recursively* within $\trianglelefteq^*$. For example we now not only have $p(X, X) \not\trianglelefteq^* p(X, Y)$ but also $f(a, p(X, X)) \not\trianglelefteq^* f(f(a), p(X, Y))$ while $f(a, p(X, X)) \trianglelefteq^+ f(f(a), p(X, Y))$.

The reason why a recursive use of, e.g., the "not strict instance" test was not incorporated in [30, 26, 31] which use $\trianglelefteq^+$ was that the authors were not sure that this would remain a wbr (no proof was found yet). In fact, recursively applying the "not strict instance" looks very dangerous. Take, e.g., the following two atoms $A_0 = p(X, X)$ and $A_1 = q(p(X, Y), p(Y, X))$. In fact, although $A_0 \trianglelefteq^+ A_1$ we do not have $A_0 \trianglelefteq^* A_1$ (when, e.g., considering both q and p as static function symbols) and one wonders whether it might be possible to create an infinite sequence

of atoms by, e.g., producing $A_2 = p(q(p(X,Y),p(Y,Z)),q(p(Z,V),p(V,X)))$. We indeed have $A_1 \not\trianglelefteq^* A_2$, but luckily $A_0 \trianglelefteq^* A_2$ and $\trianglelefteq^*$ satisfies the wqo requirement of Definition 3. But can we construct some sequence for which $\trianglelefteq^*$ does not conform to Definition 3?

The following Theorem 3 shows that such a sequence *cannot* be constructed. However, if we slightly strengthen point 3 of Definition 9 by requiring that $\langle s_1,\ldots,s_m\rangle$ is not a strict instance of the selected subsequence $\langle t_{i_1},\ldots,t_{i_m}\rangle$, we actually no longer have a wqo, as the following sequence of expression shows: $A_0 = f(p(X,X))$, $A_1 = f(p(X,Y),p(Y,X))$, $A_2 = f(p(X,Y),p(Y,Z),p(Z,X))$, ... Using the slightly strengthened embedding relation no A_i would be embedded in any A_j with $j \neq i$, while using Definition 9 unmodified we have, e.g., $A_1 \trianglelefteq^* A_2$ (but not $A_0 \trianglelefteq^* A_1$ or $A_0 \trianglelefteq^* A_2$).

Theorem 3. *$\trianglelefteq^*$ is a wbr on expressions. Additionally, if $\preceq_F$ and $\preceq_S$ are wqo's then so is $\trianglelefteq^*$.*

The proof can be found in [27].

7 Discussion and Conclusion

Of course $\trianglelefteq^*$ is not the ultimate relation for ensuring online termination. Although it has proven to be extremely useful superimposed, e.g., on determinate unfolding, on its own in the context of local control of partial deduction, $\trianglelefteq^*$ (as well as $\trianglelefteq^+$ and $\trianglelefteq$) will sometimes allow too much unfolding than desirable for efficiency concerns: more unfolding does not always imply a better specialised program. We refer to the solutions developed in, e.g., [31, 20]. Similar problems can arise in the setting of global control and we again refer to [31, 20] for discussions and experiments. Also, the issue of an efficient implementation of the homeomorphic embedding relation still remains open. (However, in Section 4 we have shown that the efficient way to use wfo's, which avoids re-scanning the entire sequence upon refinement, has very undesirable properties.)

For some applications, $\trianglelefteq$ as well as $\trianglelefteq^+$ and $\trianglelefteq^*$ remain too restrictive. In particular, they do not always deal satisfactorily with fluctuating structure (arising, e.g., for certain metainterpretation tasks) [50]. The use of characteristic trees [26, 31] remedies this problem to some extent, but not totally. A further step towards a solution is presented in [50]. In that light, it might be of interest to study whether the extensions of the homeomorphic embedding relation proposed in [42] and [24] (in the context of static termination analysis of term rewrite systems) can be useful in an online setting.

In summary, we have discussed the relation between wqo's and wfo's. We have illustrated that $\trianglelefteq$, despite its simplicity, is strictly more generous than the class of monotonic wfo's and simplification orderings combined. As all the wfo's used for automatic online termination (so far) are actually monotonic, this formally establishes the interest of $\trianglelefteq$ in that context. We have also compared to techniques based upon nearly-foundedness, and have shown that such techniques—contrary to $\trianglelefteq$— can lead to the undesirable property of strict over-eagerness.

We have also presented new embedding relations $\trianglelefteq^+$, $\trianglelefteq_{var}$ and $\trianglelefteq^*$, which inherit all the good properties of $\trianglelefteq$ while providing a refined treatment of (logical) variables. We believe that these refinements can be of value in other contexts and for other languages (such as in the context of partial evaluation of functional-logic programs [3, 2, 22, 1] or of supercompilation [49, 15, 47] of functional programming languages, where — at specialisation time – variables also appear). For instance, one can simply plug $\trianglelefteq^*$ into the language-independent framework of [45].

We also believe that $\trianglelefteq^*$ provides both a theoretically and practically more satisfactory basis than $\trianglelefteq^+$ or $\trianglelefteq$. We also believe that $\trianglelefteq^*$ can play a contributing role in other areas, such as controlling abstraction and ensuring termination of infinite model checking.

Acknowledgements

I would like to thank Danny De Schreye, Robert Glück, Jesper Jørgensen, Bern Martens, Maurizio Proietti, Jacques Riche and Morten Heine Sørensen for all the discussions and joint research which led to this paper. Anonymous referees, Patrick Cousot, Renaud Marlet, and Bern Martens provided extremely useful feedback on this paper.

References

1. E. Albert, M. Alpuente, M. Falaschi, P. Julián, and G. Vidal. Improving control in functional logic program specialization. In G. Levi, editor, Static Analysis. *Proceedings of SAS'98*, LNCS 1503, pages 262–277, Pisa, Italy, September 1998. Springer-Verlag.
2. M. Alpuente, M. Falaschi, P. Julián, and G. Vidal. Spezialisation of lazy functional logic programs. In *Proceedings of PEPM'97, the ACM Sigplan Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 151–162, Amsterdam, The Netherlands, 1997. ACM Press.
3. M. Alpuente, M. Falaschi, and G. Vidal. Narrowing-driven partial evaluation of functional logic programs. In H. Riis Nielson, editor, *Proceedings of the 6th European Symposium on Programming, ESOP'96*, LNCS 1058, pages 45–61. Springer-Verlag, 1996.
4. K. R. Apt. Introduction to logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 10, pages 495–574. North-Holland Amsterdam, 1990.
5. R. Bol. Loop checking in partial deduction. *The Journal of Logic Programming*, 16(1&2):25–46, 1993.
6. M. Bruynooghe, D. De Schreye, and B. Martens. A general criterion for avoiding infinite unfolding during partial deduction. *New Generation Computing*, 11(1):47–79, 1992.
7. D. De Schreye and S. Decorte. Termination of logic programs: The never ending story. *The Journal of Logic Programming*, 19 & 20:199–260, May 1994.
8. D. De Schreye, R. Glück, J. Jørgensen, M. Leuschel, B. Martens, and M. H. Sørensen. Conjunctive partial deduction: Foundations, control, algorithms and experiments. To appear in *The Journal of Logic Programming*, 1999.

9. N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.
10. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 243–320. Elsevier, MIT Press, 1990.
11. N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.
12. J. Gallagher. Tutorial on specialisation of logic programs. In *Proceedings of PEPM'93, the ACM Sigplan Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 88–98. ACM Press, 1993.
13. R. Glück and J. Hatcliff, John Jørgensen. Generalization in hierarchies of online program specialization systems. In *this volume*.
14. R. Glück, J. Jørgensen, B. Martens, and M. H. Sørensen. Controlling conjunctive partial deduction of definite logic programs. In H. Kuchen and S. Swierstra, editors, *Proceedings of the International Symposium on Programming Languages, Implementations, Logics and Programs (PLILP'96)*, LNCS 1140, pages 152–166, Aachen, Germany, September 1996. Springer-Verlag.
15. R. Glück and M. H. Sørensen. A roadmap to supercompilation. In O. Danvy, R. Glück, and P. Thiemann, editors, *Proceedings of the 1996 Dagstuhl Seminar on Partial Evaluation*, LNCS 1110, pages 137–160, Schloß Dagstuhl, 1996. Springer-Verlag.
16. J. Gustedt. *Algorithmic Aspects of Ordered Structures*. PhD thesis, Technische Universität Berlin, 1992.
17. G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 2:326–336, 1952.
18. G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.
19. N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.
20. J. Jørgensen, M. Leuschel, and B. Martens. Conjunctive partial deduction in practice. In J. Gallagher, editor, *Proceedings of the International Workshop on Logic Program Synthesis and Transformation (LOPSTR'96)*, LNCS 1207, pages 59–82, Stockholm, Sweden, August 1996. Springer-Verlag.
21. J. B. Kruskal. Well-quasi ordering, the tree theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95:210–225, 1960.
22. L. Lafave and J. Gallagher. Constraint-based partial evaluation of rewriting-based functional logic programs. In N. Fuchs, editor, *Proceedings of the International Workshop on Logic Program Synthesis and Transformation (LOPSTR'97)*, LNCS 1463, pages 168–188, Leuven, Belgium, July 1998.
23. P. Lescanne. Rewrite orderings and termination of rewrite systems. In A. Tarlecki, editor, *Mathematical Foundations of Computer Science 1991*, LNCS 520, pages 17–27, Kazimierz Dolny, Poland, September 1991. Springer-Verlag.
24. P. Lescanne. Well rewrite orderings and well quasi-orderings. Technical Report N^o 1385, INRIA-Lorraine, France, January 1991.
25. M. Leuschel. The ECCE partial deduction system and the DPPD library of benchmarks. Obtainable via `http://www.cs.kuleuven.ac.be/~dtai`, 1996.
26. M. Leuschel. *Advanced Techniques for Logic Program Specialisation*. PhD thesis, K.U. Leuven, May 1997. Accessible via `http://www.ecs.soton.ac.uk/~mal`.
27. M. Leuschel. Homeomorphic embedding for online termination. Technical Report DSSE-TR-98-11, Department of Electronics and Computer Science, University of Southampton, UK, October 1998.

28. M. Leuschel. On the power of homeomorphic embedding for online termination. In G. Levi, editor, Static Analysis. *Proceedings of SAS'98*, LNCS 1503, pages 230–245, Pisa, Italy, September 1998. Springer-Verlag.
29. M. Leuschel and D. De Schreye. Constrained partial deduction and the preservation of characteristic trees. *New Generation Computing*, 16(3):283–342, 1998.
30. M. Leuschel and B. Martens. Global control for partial deduction through characteristic atoms and global trees. In O. Danvy, R. Glück, and P. Thiemann, editors, *Proceedings of the 1996 Dagstuhl Seminar on Partial Evaluation*, LNCS 1110, pages 263–283, Schloß Dagstuhl, 1996. Springer-Verlag.
31. M. Leuschel, B. Martens, and D. De Schreye. Controlling generalisation and polyvariance in partial deduction of normal logic programs. *ACM Transactions on Programming Languages and Systems*, 20(1):208–258, January 1998.
32. N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. Unfolding the mystery of mergesort. In N. Fuchs, editor, *Proceedings of the International Workshop on Logic Program Synthesis and Transformation (LOPSTR'97)*, LNCS 1463, pages 206–225, Leuven, Belgium, July 1998.
33. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
34. J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *The Journal of Logic Programming*, 11(3& 4):217–242, 1991.
35. R. Marlet. *Vers une Formalisation de l'Évaluation Partielle*. PhD thesis, Université de Nice - Sophia Antipolis, December 1994.
36. B. Martens. *On the Semantics of Meta-Programming and the Control of Partial Deduction in Logic Programming*. PhD thesis, K.U. Leuven, February 1994.
37. B. Martens and D. De Schreye. Automatic finite unfolding using well-founded measures. *The Journal of Logic Programming*, 28(2):89–146, August 1996.
38. B. Martens, D. De Schreye, and T. Horváth. Sound and complete partial deduction with unfolding based on well-founded measures. *Theoretical Computer Science*, 122(1–2):97–117, 1994.
39. B. Martens and J. Gallagher. Ensuring global termination of partial deduction while allowing flexible polyvariance. In L. Sterling, editor, *Proceedings ICLP'95*, pages 597–613, Kanagawa, Japan, June 1995. MIT Press.
40. A. Middeldorp and H. Zantema. Simple termination of rewrite systems. *Theoretical Computer Science*, 175(1):127–158, 1997.
41. L. Plümer. *Termination Proofs for Logic Programs*. LNCS 446. Springer-Verlag, 1990.
42. L. Puel. Using unavoidable set of trees to generalize Kruskal's theorem. *Journal of Symbolic Computation*, 8:335–382, 1989.
43. E. Ruf. *Topics in Online Partial Evaluation*. PhD thesis, Stanford University, March 1993.
44. D. Sahlin. Mixtus: An automatic partial evaluator for full Prolog. *New Generation Computing*, 12(1):7–51, 1993.
45. M. H. Sørensen. Convergence of program transformers in the metric space of trees. In *Mathematics of Program Construction, Proceedings of MPC'98*, LNCS 1422, pages 315–337. Springer-Verlag, 1998.
46. M. H. Sørensen and R. Glück. An algorithm of generalization in positive supercompilation. In J. W. Lloyd, editor, *Proceedings of ILPS'95, the International Logic Programming Symposium*, pages 465–479, Portland, USA, December 1995. MIT Press.
47. M. H. Sørensen, R. Glück, and N. D. Jones. A positive supercompiler. *Journal of Functional Programming*, 6(6):811–838, 1996.

48. J. Stillman. *Computational Problems in Equational Theorem Proving*. PhD thesis, State University of New York at Albany, 1988.
49. V. F. Turchin. The concept of a supercompiler. *ACM Transactions on Programming Languages and Systems*, 8(3):292–325, 1986.
50. W. Vanhoof and B. Martens. To parse or not to parse. In N. Fuchs, editor, *Proceedings of the International Workshop on Logic Program Synthesis and Transformation (LOPSTR'97)*, LNCS 1463, pages 322–342, Leuven, Belgium, July 1997.
51. A. Weiermann. Complexity bounds for some finite forms of Kruskal's theorem. *Journal of Symbolic Computation*, 18(5):463–488, November 1994.
52. D. Weise, R. Conybeare, E. Ruf, and S. Seligman. Automatic online partial evaluation. In *Proceedings of the Conference on Functional Programming Languages and Computer Architectures*, LNCS 523, pages 165–191, Harvard University, 1991. Springer-Verlag.