On the Power of Homeomorphic Embedding for Online Termination

Michael Leuschel*

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

Abstract. Recently well-quasi orders in general, and homeomorphic embedding in particular, have gained popularity to ensure the termination of program analysis, specialisation and transformation techniques. In this paper we investigate and clarify for the first time, both intuitively and formally, the advantages of such an approach over one using well-founded orders. Notably we show that the homeomorphic embedding relation is strictly more powerful than a large class of involved well-founded approaches.

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. [7,8,9,37] and references therein) or of logic programs (e.g. [6,38] 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:

- *static* techniques, which prove or ensure termination of a program or process *beforehand* (i.e. *off-line*) without any kind of execution, and
- *online* (or *dynamic*) techniques, which ensure termination of a process *during* its execution. (The process itself can of course be, e.g., performing a static analysis.)

Static 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 [6,38] falls within the former context, while termination of program specialisation, transformation or analysis is often ensured in an online manner.

This paper is primarily aimed at studying and improving online termination techniques. Let us examine the case of partial deduction [29,10,23] — an automatic technique for specialising logic programs. Henceforth we suppose some familiarity with basic notions in logic programming [3,28].

* Part of the work was done while the author was Post-doctoral Fellow of the Fund for Scientific Research - Flanders Belgium (FWO) and visiting DIKU, University of Copenhagen.

G. Levi (Ed.): SAS'98, LNCS 1503, pp. 230–245, 1998.

Partial deduction based upon the Lloyd and Shepherdson framework [29] generates (possibly incomplete) SLDNF-trees for a set $\mathcal{A}$ of atoms. The specialised program is extracted from these trees by producing one clause 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 [10,34]. 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. [26]).

Below we mainly use local termination to illustrate our concepts. (As shown in [34] 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.)

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. For instance, abstract interpretation techniques usually analyse a set of abstract calls to which new call patterns are continuously added. One thus faces a problem very similar to global termination of partial deduction.

Depth Bounds. 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.

Determinacy. Another approach, often used in partial evaluation of functional programs [17] 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 [5,33,32,31] in logic and [40,47] 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 is no infinite admissible sequence wrt* $>_S$

To ensure local termination, 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, 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, exist and we refer the reader to [5,33,32,31] for precise details. 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). Other refinements consist in allowing the wfo's not only to depend upon the selected atom but on the *context* as well [32] or to ignore calls to *non-recursive* predicates. [32] also discusses a way to relax the condition of a "strict decrease" when refining a wfo. (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 [4,41,42,25,26,11,18,1,20,46]. Unfortunately, this move to well-quasi orders has never been formally justified nor has the relation to well-founded approaches been investigated. We strive to do so in this paper and will actually prove that a rather simple well-quasi approach is strictly more powerful than a large class of involved well-founded approaches.

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. 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 functors, 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 [14] used an alternate definition of well-quasi orders in terms of the "finite basis property" (or "finite generating set" in [19]). A different (but also equivalent) definition of a wqo is: 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 [7,8]. The use of well-quasi orders in an *online* setting has only emerged recently (it is mentioned, e.g., in [4] but also [41]) and has never been compared to well-founded approaches. There has been some comparison between wfo's and wqo's in the offline setting, e.g., in [37] it is argued that (for "simply terminating" rewrite systems) approaches based upon quasi-orders are less interesting than ones based upon a partial orders. In this paper we will show that the situation is somewhat reversed in an online setting. Furthermore, in the online setting, transitivity of a wqo is not really interesting and one can therefore drop this requirement, leading to the use of wbr's. [24] contains some useful wbr's which are not wqo's.

An interesting wqo is the homeomorphic embedding relation $\trianglelefteq$, which derives from results by Higman [14] and Kruskal [19]. It has been used in the context of term rewriting systems in [7,8], and adapted for use in supercompilation [45] in [42]. Its usefulness as a stop criterion for partial evaluation is also discussed and

advocated in [30]. Some complexity results can be found in [44] and [13] (also summarised in [30]).

The following is the definition from [42], which adapts the pure homeomorphic embedding from [8] by adding a rudimentary treatment of variables.

Definition 4. **($\trianglelefteq$)** *The* (pure) homeomorphic embedding *relation $\trianglelefteq$ on expressions is defined inductively 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. 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 $t \not\trianglelefteq s$.

Example 2. 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. For instance we have $p(a) \trianglelefteq p(f(a))$ and indeed $p(a)$ can be obtained from $p(f(a))$ by "striking out" the f. 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. *The relation $\trianglelefteq$ is a wqo on the set of expressions over a finite alphabet.*

For a complete proof, reusing Higman's and Kruskal's results [14,19] in a very straightforward manner, see, e.g., [23]. Extensions to infinite alphabets and improved treatment of variables can be found in [24].

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 $rev(T, [b, a], R) \trianglelefteq rev(T', [H', b, a], R)$. 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 [25,23,26] 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 [30]).

3 Comparing wbr's and wfo's

3.1 General Comparison

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. All missing proofs can be found in [24].

Lemma 1. (wbr from wfo) *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 (see below) 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 $\trianglelefteq$.

3.2 Homeomorphic Embedding and Monotonic wfo's

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 [38,27]; for some experiments see Appendix A). Also, the *produce-consume* example from [31] requires rather involved techniques (considering the context) to be solved by wfo's. Again, this example poses no problem to $\trianglelefteq$ (cf. Appendix A). 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. also Appendix A). For instance, $\trianglelefteq$ will admit the following sequences (where Example 1 is progressively wrapped into "vanilla" metainterpreters counting resolution steps and keeping track of the selected predicates respectively):

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

Again, this is very difficult for wfo's and requires refined and involved techniques (of which to our knowledge no implementation exists).

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 we now establish that — in the online setting — $\trianglelefteq$ is strictly more generous than a large class of refined wfo's.

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

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

Note the similarity of structure with the definition of $\trianglelefteq$ (but, contrary to $\trianglelefteq$, $\not\prec$ does not have to be the least relation satisfying the rules). This similarity of structure will later enable us to prove that any sequence admissible wrt $\prec$ must also be admissible wrt $\trianglelefteq$ (by showing that $s \trianglelefteq t \Rightarrow s \not\prec t$). Also observe that point 2 need not hold for predicate symbols and that point 3 implies that $c \not\prec c$ for all constant and proposition symbols c. Finally, there is 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 [37] and monotonic in [7]). More on that below.

Similarly, we say that a *norm* $\|.\|$ (i.e. a mapping from expressions to $\mathbb{N}$) is said to be monotonic iff the associated wfo $\prec_{\|.\|}$ is monotonic ($t_1 \prec_{\|.\|} t_2$ iff $\|t_1\| < \|t_2\|$).

For instance the termsize norm (see below) is trivially monotonic. More generally, any semi-linear norm of the following form is monotonic:

Proposition 2. *Let the norm $\|.\| : Expr \rightarrow \mathbb{N}$ be defined by:*

- $\|t\| = c_f + \sum_{i=1}^{n} c_{f,i}\|t_i\|$ *if $t = f(t_1, \ldots, t_n)$, $n \geq 0$*
- $\|t\| = c_v$ *otherwise (i.e. t is a variable)*

Then $\|.\|$ is monotonic if all coefficients $c_v, c_f, c_{f,i}$ are ≥ 0 and $c_{f,i} \geq 1$ for all function symbols f of arity ≥ 1 (but not necessarily for all predicate symbols).

Proof. As $<$ on $\mathbb{N}$ is total we have that $s \not> t$ is equivalent to $s \leq t$. The proof proceeds by induction on the structure of the expressions and examines every rule of Definition 5 separately:

1. $X \leq Y$ for all variables X, Y
   this trivially holds as we use the same constant c_v for all variables.
2. $s \leq f(t_1, \ldots, t_n)$ whenever $s \leq t_i$ for some i
   This holds trivially if all coefficients are ≥ 0 and if $c_{f,i} \geq 1$. This is verified, as the rule only applies if f is a function symbol.
3. $f(s_1, \ldots, s_n) \leq f(t_1, \ldots, t_n)$ whenever $\forall i \in \{1, \ldots, n\} : s_i \leq f_i$
   This holds trivially, independently of whether f is a function or predicate symbol, as all coefficients are positive (and the same coefficient is applied to s_i and t_i).

By taking $c_v = 0$ and $c_{f,i} = c_f = 1$ for all f we get the *termsize* norm $\|.\|_{ts}$, which by the above proposition is thus monotonic. Also, by taking $c_v = 1$ and $c_{f,i} = c_f = 1$ for all f we also get a monotonic norm, counting symbols. Finally, a *linear norm* can always be obtained [38] by setting $c_v = 0$, $c_{f,i} = 1$ and $c_f \in \mathbb{N}$ for all f. Thus, as another corollary of the above, any linear norm is monotonic.

Proposition 3. *Let $\|.\|_1, \ldots, \|.\|_k$ be monotonic norms satisfying Proposition 2. Then the lexicographical ordering $\prec_{lex}$ defined by $s \prec_{lex} t$ iff*

$\exists i \in \{1, \ldots, k\}$ such that $\|s\|_i < \|t\|_i$ and $\forall j \in \{1, \ldots, i-1\}$: $\|s\|_j = \|t\|_j$ is a monotonic wfo.

Proof. By standard results (see, e.g., [8]) we know that $\prec_{lex}$ is a wfo (as $<$ is a wfo on $\mathbb{N}$). We will prove that $\prec_{lex}$ satisfies all the rules of Definition 5.
1. First, rule 1 is easy as $\|X\|_i = \|Y\|_i$ for all i and variables X, Y and therefore we never have $X \prec_{lex} Y$.
2. Before examining the other rules, let us note that $s \not\succ_{lex} t$ is equivalent to saying that either

a) $\forall j \in \{1, \ldots, k\}$ $\|s\|_j = \|t\|_j$ or
b) there exists an $j \in \{1, \ldots, k\}$ such that $\|s\|_j < \|t\|_j$ and $\forall l \in \{1, \ldots, j-1\}$: $\|s\|_l = \|t\|_l$.

Let us now examine rule 2 of Definition 5. We have to prove that whenever $s \not\succ_{lex} t_i$ the conclusion of the rule holds.
Let us first examine case a) for $s \not\succ_{lex} t_i$. We have $\|s\|_j = \|t_i\|_j$ and thus we know that $\|s\|_j \leq \|f(t_1, \ldots, t_n)\|_j$ by monotonicity of $\|.\|_j$ (as $<$ on $\mathbb{N}$is total we have that $s \not> t$ is equivalent to $s \leq t$). As this holds for all $\|.\|_j$ we cannot have $s_j \succ_{lex} f(t_1, \ldots, t_n)$.
Let us now examine the second case b) for $s \not\succ_{lex} t_i$. Let $j \in \{1, \ldots, k\}$ such that $\|s\|_j < \|t_i\|_j$ and $\forall l \in \{1, \ldots, j-1\}$: $\|s\|_l = \|t_i\|_l$. For all l we can deduce as above that $\|s\|_l \leq \|f(t_1, \ldots, t_n)\|_l$. However, we still have to prove that $\|s\|_j < \|f(t_1, \ldots, t_n)\|_j$. By monotonicity of $\|.\|_j$ we only know that $\|s\|_j \leq \|f(t_1, \ldots, t_n)\|_j$. But we can also apply monotonicity of $\|.\|_j$ to deduce that $\|t_i\|_j \leq \|f(t_1, \ldots, t_n)\|_j$ and hence we can infer the desired property (as $\|s\|_j < \|t_i\|_j$).

3. Now, for rule 3 we have to prove that whenever $s_i \not\succ_{lex} t_i$ for all $i \in \{1, \ldots, n\}$ the conclusion of the rule holds. There are again two cases.
a) We can have $\|s_i\|_j = \|t_i\|_j$ for all i, j. By monotonicity of each $\|.\|_j$ we know that $\|f(s_1, \ldots, s_n)\|_j \leq \|f(t_1, \ldots, t_n)\|_j$ for all $j \in \{1, \ldots, k\}$. Hence, we cannot have $f(s_1, \ldots, s_n) \succ_{lex} f(t_1, \ldots, t_n)$.
b) In the other case we know that there must be a value $j' \in \{1, \ldots, k\}$ such that for some i: $\|s_i\|_{j'} < \|t_i\|_{j'}$ and $\forall l \in \{1, \ldots, j'-1\}$: $\|s_i\|_l = \|t_i\|_l$. I.e., by letting j denote the minimum value j' for which this holds, we know that for some i: $\|s_i\|_j < \|t_i\|_j$ and for all i': $\forall l \in \{1, \ldots, j\}$: $\|s_{i'}\|_l \leq \|t_{i'}\|_l$. By monotonicity of each $\|.\|_l$ we can therefore deduce that $\forall l \in \{1, \ldots, j\}$: $\|f(s_1, \ldots, s_n)\|_l \leq \|f(t_1, \ldots, t_n)\|_l$. We can also deduce by monotonicity of $\|.\|_j$ that $\|f(s_1, \ldots, s_n)\|_j \leq \|f(t_1, \ldots, t_n)\|_j$. We can even deduce that $\|f(s_1, \ldots, s_n)\|_j \leq \|f(t_1, \ldots, t_{i-1}, s_i, t_{i+1}, \ldots, t_n)\|_j \leq \|f(t_1, \ldots, t_n)\|_j$. Now, we just have to prove that:
$\|f(t_1, \ldots, t_{i-1}, s_i, t_{i+1}, \ldots, t_n)\|_j < \|f(t_1, \ldots, t_n)\|_j$ in order to affirm that
$\|f(s_1, \ldots, s_n)\|_j \not\succ_{lex} \|f(t_1, \ldots, t_n)\|_j$. This does not hold for all monotonic norms, but as we know that $\|.\|_j$ satisfies Proposition 2, this can be deduced by the fact that the coefficient $c_{f,i}$ in $\|.\|_j$ must be ≥ 1.

It is important that the norms $\|.\|_1, \ldots, \|.\|_k$ satisfy Proposition 2. Otherwise, a counterexample would be as follows. Let $\|a\|_1 = 1$, $\|b\|_1 = 2$ and $\|f(a)\|_1 = \|f(b)\|_1 = 5$. Also let $\|a\|_2 = 2$, $\|b\|_2 = 1$ and $\|f(a)\|_2 = 3$, $\|f(b)\|_2 = 2$. Now we have $a \prec_{lex} b$, i.e. $a \not\succ_{lex} b$, but also $f(a) \succ_{lex} f(b)$ and condition 3 of monotonicity for $\prec_{lex}$ is violated.

One could make Proposition 3 slightly more general, but the current version is sufficient to show the desired result, namely that most of the wfo's used in online practice are actually monotonic. For example almost all of the refined wfo's defined in [5,33,32,31] are monotonic:

- Definitions 3.4 of [5], 3.2 of [33] and 2.14 of [32] all sum up the number of function symbols (i.e. termsize) of a subset of the argument positions of

atoms. These wfo's are therefore immediately covered by Proposition 2. The algorithms only differ in the way of choosing the positions to measure. The early algorithms simply measure the input positions, while the later ones dynamically refine the argument positions to be measured (but which are still measured using the termsize norm).
- Definitions 3.2 of [32] as well as 8.2.2 of [31] use the lexicographical order on the termsizes of some selected argument positions. These wfo's are therefore monotonic as a corollary to Propositions 2 and 3.

The only non-monotonic wfo in that collection of articles is the one defined specifically for metainterpreters in Definition 3.4 of [5] (also in Sect. 8.6 of [31]) which uses selector functions to focus on subterms to be measured. We will return to this approach below.

Also, as already mentioned, some of the techniques in [32,31] (in Sects. 3.4 and 8.2.4 respectively) do not require the *whole* sequence to be admissible wrt a unique wfo, i.e. one can split up a sequence into a (finite) number of subsequences and apply different (monotonic) wfo's on these subsequences. Similar refinements can also be developed for wqo's and the formal study of these refinements are (thus) not the main focus of the paper.

Before showing that $\trianglelefteq$ is strictly more powerful than the union of all monotonic wfo's, we adapt the class of simplification orderings from term rewriting systems. It will turn out that the power of this class 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 powerful wfo's are simplification orderings [7,37]: recursive path ordering, Knuth-Bendix ordering or lexicographic path ordering, to name just a few. However, not all wfo's of Proposition 2 are simplification orderings: e.g., for $c_f = 0, c_a = 1$ we have $\|a\| = \|f(a)\|$ and the subterm property does not hold (for the associated wfo). In addition, Proposition 2 allows a special treatment for predicates. On the other hand, there are wfo's which are simplification orderings but are not monotonic according to Definition 5.

Proposition 4. *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.*

Proof. First, let us observe that for a given wfo $\prec$ on expressions, any admissible sequence wrt $\prec$ is also an admissible sequence wrt $\trianglelefteq$ iff $s \succ t \Rightarrow s \not\trianglelefteq t$. Indeed ($\Rightarrow$), whenever $s \trianglelefteq t$ then $s \not\succ t$, and this trivially implies (by transitivity of $\prec$) that any sequence not admissible wrt $\trianglelefteq$ cannot be strictly descending wrt $\prec$. On the other hand

($\Leftarrow$), let us assume that for some s and t $s \trianglelefteq t$ but $s \succ t$. This means that the sequence s, t is admissible wrt $\succ$ but not wrt $\trianglelefteq$ and we have a contradiction.

a) The proof that for a monotonic wfo $\prec$ we have $s \trianglelefteq t \Rightarrow s \not\succ t$ is by straightforward induction on the structure of s and t. The only "tricky" aspect is that the second rule for monotonicity only holds if f is a function symbol. But if f is a predicate symbol, then $s \trianglelefteq t_i$ cannot hold because we supposed that predicate and function symbols are distinct.

b) If $\prec$ is a simplification ordering then we can apply Lemma 3.3 of [37] to deduce that $\prec$ is the superset of the strict part of $\trianglelefteq$ (i.e., $\prec \supseteq \triangleleft$). Let us examine the two possibilities for $s \trianglelefteq t$. First, we can have $s \triangleleft t$. In that case we can deduce $s \prec t$ and thus $s \not\succ t$. Second, we can have $s \trianglelefteq t$ and $t \trianglelefteq s$. In that case s and t are identical, except for the variables. If we now take the substitution σ which assigns all variables in s and t to a unique variable we have $s\sigma = t\sigma$, i.e., $s\sigma \not\succ t\sigma$. This means that $s \succ t$ cannot hold (because $\succ$ is invariant under variable replacement).

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.[1] 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 $\|t_1\|_{llen} = 3 > \|t_2\|_{llen} = 2$ 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 $s \trianglelefteq f(\ldots s \ldots)$,[2] i.e. the sequence $s, f(\ldots s \ldots)$ is admissible wrt $\succ$ but not wrt $\trianglelefteq$.

For that same reason the wfo's for metainterpreters defined in Definition 3.4 of [5] mentioned above are not monotonic, as they are allowed to completely focus on subterms, fully ignoring other subterms. However, automation of that technique is not addressed in [5]. E.g., for this wfo one cannot immediately apply

[1] 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, ...).

[2] 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 in e.g. Proposition 2.

the idea of continually refining the measured subterms, because otherwise one might simply plunge deeper and deeper into the terms and termination would not be ensured. A step towards an automatic implementation is presented in Sect. 8.6 of [31] and it will require further work to formally compare it with wqo-based approaches and whether the ability to completely ignore certain parts of an expression can be beneficial for practical programs. But, as we have seen earlier, $\trianglelefteq$ alone is already very flexible for metainterpreters, even more so when combined with characteristic trees [26] (see also [46]).

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 $p([a],[]) \rightarrow p([],[a]) \rightarrow p([a],[]) \rightarrow \ldots$.[3]

Short Note on Offline Termination. This example also shows why $\trianglelefteq$ (or well-quasi orders in general) cannot be used *directly* for static termination analysis. Let us explain what we mean. Take, e.g., a program containing the clauses $C_1 = p([a],[]) \leftarrow p([],[a])$ and $C_2 = p([],[a]) \leftarrow p([a],[])$. Then, in both cases the body is not embedding the head, but still the combination of the two clauses leads to a non-terminating program. However, $\trianglelefteq$ can be used to *construct well-founded* orders for static termination analysis. Take the clause C_1. The head and the body are incomparable according to $\trianglelefteq$. So, we can simply extend $\trianglelefteq$ by stating that $p([a],[]) \rhd p([],a)$ (thus making the head strictly larger than the body atom). As already mentioned, for any extension $\leq$ of a wqo we have that $<$ is a wfo. Thus we know that the program just consisting of C_1 is terminating. If we now analyse C_2 we have that, according to the extended wqo, the body is strictly larger than the head and (luckily) we cannot prove termination (i.e. there is no way of extending $\trianglelefteq$ so that for both C_1 and C_2 the head is strictly larger than the body).

4 Discussion and Conclusion

Of course $\trianglelefteq$ is not the ultimate relation for ensuring online termination. On its own in the context of local control of partial deduction, $\trianglelefteq$ will sometimes allow

[3] When using a wfo one has to compare only to the closest predecessor [32], 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., [31,32]).

too much unfolding than desirable for efficiency concerns (i.e. more unfolding does not always imply a better specialised program) and we refer to the solutions developed in, e.g., [26,18].

For some applications, $\trianglelefteq$ remains too restrictive. In particular, it does not always deal satisfactorily with fluctuating structure (arising, e.g., for certain meta-interpretation tasks) [46]. The use of characteristic trees [23,26] remedies this problem to some extent, but not totally. A further step towards a solution is presented in [46]. In that light, it might be of interest to study whether the extensions of the homeomorphic embedding relation proposed in [39] and [21] (in the context of static termination analysis of term rewrite systems) can be useful in an online setting. As shown in [24] the treatment of variables of $\trianglelefteq$ is rather rudimentary and several ways to remedy this problem are presented.

In summary, we have shed new light on the relation between wqo's and wfo's and have formally shown (for the first time) why wqo's are more interesting than wfo's for ensuring termination in an online setting (such as program specialisation or analysis). We have illustrated the inherent flexibility of $\trianglelefteq$ and proved that, despite its simplicity, it is strictly more generous than the class of monotonic wfo's. 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.

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, comments and joint research which led to this paper. Anonymous referees as well as Bern Martens provided extremely useful feedback on this paper.

References

1. M. Alpuente, M. Falaschi, P. Julián, and G. Vidal. Spezialisation of lazy functional logic programs. In *Proceedings PEPM'97*, pages 151–162, Amsterdam, The Netherlands, 1997. ACM Press.
2. M. Alpuente, M. Falaschi, and G. Vidal. Narrowing-driven partial evaluation of functional logic programs. In H. Riis Nielson, editor, *Proceedings ESOP'96*, LNCS 1058, pages 45–61. Springer-Verlag, 1996.
3. 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.
4. R. Bol. Loop checking in partial deduction. *The Journal of Logic Programming*, 16(1&2):25–46, 1993.
5. 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.
6. 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.
7. N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

8. 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.
9. N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.
10. J. Gallagher. Tutorial on specialisation of logic programs. In *Proceedings PEPM'93*, pages 88–98. ACM Press, 1993.
11. 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 PLILP'96*, LNCS 1140, pages 152–166, Aachen, Germany, September 1996. Springer-Verlag.
12. 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.
13. J. Gustedt. *Algorithmic Aspects of Ordered Structures*. PhD thesis, Technische Universität Berlin, 1992.
14. G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 2:326–336, 1952.
15. G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.
16. N. D. Jones. An introduction to partial evaluation. *ACM Computing Surveys*, 28(3):480–503, September 1996.
17. N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.
18. J. Jørgensen, M. Leuschel, and B. Martens. Conjunctive partial deduction in practice. In J. Gallagher, editor, *Proceedings LOPSTR'96*, LNCS 1207, pages 59–82, Stockholm, Sweden, August 1996. Springer-Verlag.
19. J. B. Kruskal. Well-quasi ordering, the tree theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95:210–225, 1960.
20. L. Lafave and J. Gallagher. Constraint-based partial evaluation of rewriting-based functional logic programs. In N. Fuchs, editor, *Proceedings LOPSTR'97*, LNCS, Leuven, Belgium, July 1997. Springer-Verlag.
21. P. Lescanne. Well rewrite orderings and well quasi-orderings. Technical Report N^o 1385, INRIA-Lorraine, France, January 1991.
22. M. Leuschel. The ECCE partial deduction system and the DPPD library of benchmarks. 1996–1998. Obtainable via `http://www.cs.kuleuven.ac.be/~dtai`.
23. M. Leuschel. *Advanced Techniques for Logic Program Specialisation*. PhD thesis, K.U. Leuven, May 1997.
Accessible via `http://www.cs.kuleuven.ac.be/~michael`.
24. M. Leuschel. Homeomorphic Embedding for Online Termination. Technical Report, Department of Electronics and Computer Science, University of Southampton, 1998.
25. 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.
26. 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.

27. N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. Unfolding the mystery of mergesort. In N. Fuchs, editor, *Proceedings LOPSTR'97*, LNCS, Leuven, Belgium, July 1997. Springer-Verlag.
28. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
29. J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *The Journal of Logic Programming*, 11(3& 4):217–242, 1991.
30. R. Marlet. *Vers une Formalisation de l'Évaluation Partielle*. PhD thesis, Université de Nice - Sophia Antipolis, December 1994.
31. B. Martens. *On the Semantics of Meta-Programming and the Control of Partial Deduction in Logic Programming*. PhD thesis, K.U. Leuven, February 1994.
32. B. Martens and D. De Schreye. Automatic finite unfolding using well-founded measures. *The Journal of Logic Programming*, 28(2):89–146, August 1996.
33. 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.
34. 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.
35. J. Martin. Sonic partial deduction. Technical Report, Department of Electronics and Computer Science, University of Southampton, 1998.
36. P.-A. Melliès. On a duality between Kruskal and Dershowitz theorems. In K. G. Larsen, editor, *Proceedings of ICALP'98*, LNCS, Aalborg, Denmark, 1998. Springer-Verlag.
37. A. Middeldorp and H. Zantema. Simple termination of rewrite systems. *Theoretical Computer Science*, 175(1):127–158, 1997.
38. L. Plümer. *Termination Proofs for Logic Programs*. LNCS 446. Springer-Verlag, 1990.
39. L. Puel. Using unavoidable set of trees to generalize Kruskal's theorem. *Journal of Symbolic Computation*, 8:335–382, 1989.
40. E. Ruf. *Topics in Online Partial Evaluation*. PhD thesis, Stanford University, March 1993.
41. D. Sahlin. Mixtus: An automatic partial evaluator for full Prolog. *New Generation Computing*, 12(1):7–51, 1993.
42. M. H. Sørensen and R. Glück. An algorithm of generalization in positive supercompilation. In J. W. Lloyd, editor, *Proceedings ILPS'95*, pages 465–479, Portland, USA, December 1995. MIT Press.
43. M. H. Sørensen, R. Glück and N.D. Jones. A positive supercompiler. *Journal of Functional Programming*, 1996.
44. J. Stillman. *Computational Problems in Equational Theorem Proving*. PhD thesis, State University of New York at Albany, 1988.
45. V. F. Turchin. The concept of a supercompiler. *ACM Transactions on Programming Languages and Systems*, 8(3):292–325, 1986.
46. W. Vanhoof and B. Martens. To parse or not to parse. In N. Fuchs, editor, *Proceedings LOPSTR'97*, LNCS, Leuven, Belgium, July 1997. Springer-Verlag.
47. 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.

A Small Experiments with the ECCE System

The purpose of this appendix is to illustrate the flexibility which the homeomorphic embedding relation provides straight "out of the box" (other more intricate well-quasi orders, like the one used by Mixtus [41], can handle some of the examples below as well).

For that we experiment with the ECCE partial deduction system [22] using an unfolding rule based on $\trianglelefteq$ which allows the selection of either determinate literals or left-most literals within a goal, given that no covering ancestor [5] is embedded (via $\trianglelefteq$). To ease readability, the specialised programs are sometimes presented in unrenamed form.

First, let us take the *mergesort* program, which is somewhat problematic for a lot of static termination analysis methods [38,27].

```
mergesort([],[]).
mergesort([X],[X]).
mergesort([X,Y|Xs],Ys) :-
       split([X,Y|Xs],X1s,X2s),
       mergesort(X1s,Y1s),mergesort(X2s,Y2s),
       merge(Y1s,Y2s,Ys).

split([],[],[]).
split([X|Xs],[X|Ys],Zs) :- split(Xs,Zs,Ys).

merge([],Xs,Xs).
merge(Xs,[],Xs).
merge([X|Xs],[Y|Ys],[X|Zs]) :- X =< Y, merge(Xs,[Y|Ys],Zs).
merge([X|Xs],[Y|Ys],[Y|Zs]) :- X>Y, merge([X|Xs],Ys,Zs).
```

The partial evaluation query:

```
?- mergesort([3,1,2],X).
```

As the following resulting specialised program shows, homeomorphic embedding allowed the full unfolding of *mergesort*:

```
mergesort([3,1,2],[1,2,3]).
```

It took ECCE less than 0.5 s on a Sparc Classic to produce the above program (including post-processing and writing to file).

We can even achieve this same feat even if we interpose one or more levels of metainterpretation! Take a vanilla solve metainterpreter with mergesort at the object-level:

```
solve([]).
solve([H|T]) :-
    claus(H,Bdy),solve(Bdy),solve(T).

claus(mergesort([],[]), []).
claus(mergesort([X],[X]), []).
```

```
claus(mergesort([X,Y|Xs],Ys),
       [split([X,Y|Xs],X1s,X2s),
        mergesort(X1s,Y1s),mergesort(X2s,Y2s),
        merge(Y1s,Y2s,Ys) ]).
claus(split([],[],[]), []).
claus(split([X|Xs],[X|Ys],Zs) , [ split(Xs,Zs,Ys) ]).
claus(merge([],Xs,Xs), []).
claus(merge(Xs,[],Xs), []).
claus(merge([X|Xs],[Y|Ys],[X|Zs]) ,
       [ X =< Y, merge(Xs,[Y|Ys],Zs) ]).
claus(merge([X|Xs],[Y|Ys],[Y|Zs]) ,
       [ X>Y, merge([X|Xs],Ys,Zs)]).
claus('=<'(X,Y),[]) :- X =< Y.
claus('>'(X,Y),[]) :- X > Y.

mergesort_test(X) :- solve([mergesort([3,1,2],X)]).
```

The partial evaluation query:

```
?- mergesort_test(X).
```

Again homeomorphic embedding allowed the full unfolding:

```
mergesort_test([1,2,3]).
```

It took ECCE 2.86 s on a Sparc Classic to produce the above program (including post-processing and writing to file).

The following example is taken from [31].

```
produce([],[]).
produce([X|Xs],[X|Ys]) :- produce(Xs,Ys).

consume([]).
consume([X|Xs]) :- consume(Xs).
```

The partial evaluation query:

```
?- produce([1,2|X],Y), consume(Y).
```

To solve it in the setting of unfolding based upon wfo's one needs both partition based measure functions as well as taking the context into account. The same adequate unfolding can simply be achieved by $\trianglelefteq$ based determinate unfolding, as illustrated by the following specialised program derived by ECCE (default settings):

```
produce_conj__1([],[1,2]).
produce_conj__1([X1|X2],[1,2,X1|X3]) :-
   produce_conj__2(X2,X3).
produce_conj__2([],[]).
produce_conj__2([X1|X2],[X1|X3]) :-
   produce_conj__2(X2,X3).
```