Extending Homeomorphic Embedding in the Context of Logic Programming

Michael Leuschel

Report CW 252, June 1997

(Revised August 1997)

Department of Computing Science, K.U.Leuven

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. However, as we illustrate in the paper, the homeomorphic embedding relation as it is usually defined suffers from several inadequacies which make it less suitable in a logic programming context. We present several increasingly refined ways to remedy this problem by providing more sophisticated treatments of variables and present a new, extended homeomorphic embedding relation.

Keywords : Termination, Well-quasi orders, Logic Programming, Program Transformation, Program Analysis, Partial Deduction.
CR Subject Classification : I.2.3, I.2.2, F.4.1, D.1.6.

Extending Homeomorphic Embedding in the Context of Logic Programming

Michael Leuschel*

Departement Computerwetenschappen
Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
e-mail : michael@cs.kuleuven.ac.be
www : `http://www.cs.kuleuven.ac.be/~michael`
Tel.: ++32 (0)16 - 32 7555 Fax: ++32 (0)16 - 32 7996

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. However, as we illustrate in the paper, the homeomorphic embedding relation as it is usually defined suffers from several inadequacies which make it less suitable in a logic programming context. We present several increasingly refined ways to remedy this problem by providing more sophisticated treatments of variables and present a new, extended homeomorphic embedding relation.

1 Introduction

The problem of ensuring termination arises in a lot of different contexts in computer science. For instance a lot of work has been devoted to proving termination of term rewriting systems (e.g. [7] and references therein) or of logic programs (e.g. [5, 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
- *on-line* (or *dynamic*) techniques, which ensure termination of a process *during* its execution.

For instance static termination analysis of logic programs [5, 38] falls within the former context, while termination of e.g. partial deduction — an automatic technique for specialising logic programs — is usually ensured in an on-line manner.

Let us examine the case of partial deduction in more detail. Partial deduction based upon the Lloyd and Shepherdson framework [30] 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 also referred to as *unfolding* steps — are those that have been performed beforehand, justifying the hope that the specialised program be more efficient.

*Supported by the Belgian GOA "Non-Standard Applications of Abstract Interpretation"

Now, to ensure termination of partial deduction two issues arise [9, 35] (cf. Figure 1). 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.

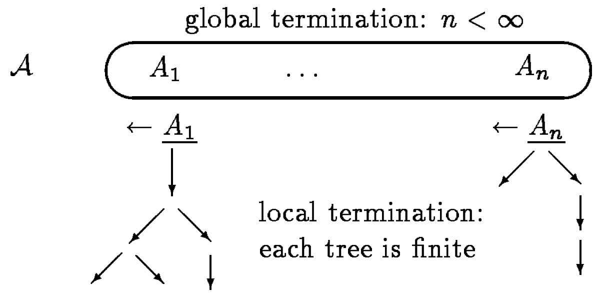
global termination: $n < \infty$
$\mathcal{A}$
A_1
...
A_n
$\leftarrow \underline{A_1}$
$\leftarrow \underline{A_n}$
local termination:
each tree is finite

Figure 1: Global and local termination

Below, in a first approach, we concentrate on local termination. But as shown in [35] 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

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. The depth bound will therefore lead either to too little or too much unfolding in a lot of interesting cases.

Another approach, often used in partial evaluation of functional programs [17, 16], is to only expand a tree while it is *determinate* (i.e. it only has one non-failing branch). However, this in itself does not guarantee termination, as there can be infinitely failing determinate computations. In (strict) functional programs such a condition can be seen as an error in the original program (the corresponding run-time computation will not terminate). In logic programming the situation is somewhat different: a goal can infinitely fail (in a deterministic way) at partial deduction time while its run-time instances finitely fail. In applications like theorem proving, even infinite failures at run-time do not necessarily indicate a programmer's error: they might simply be due to unprovable statements. This is why, contrary to maybe functional programming, measures in addition to determinacy have to be adopted to ensure local termination.

Luckily, more refined approaches to ensure termination of unfolding exist. The methods in [4, 34, 33, 32] are based on *well-founded orders*, inspired by their usefulness in the context of static termination analysis (see e.g. [8, 5]). These techniques ensure termination, while at the same time allowing unfolding related to the structural aspect of the program and goal to be partially deduced, e.g. permitting the consumption of static input within the atoms of $\mathcal{A}$.

Formally, well-founded sets and orders are defined as follows:

Definition 1.1 (s-poset) A *strict partial order* on a set S is an anti-reflexive, anti-symmetric and transitive binary relation on $S \times S$. A couple $S, >_S$ consisting of a set S and a strict partial order $>_S$ on S is called an *s-poset* or partially strictly ordered set.

Definition 1.2 (wfo) An s-poset $S, >_S$ is called *well-founded* iff there is no infinite sequence of elements $s_1, s_2, \ldots$ in S such that $s_i > s_{i+1}$, for all $i \geq 1$. The order $>_S$ is also called a *well-founded order (wfo)* on 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 strictly decreasing 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 1.3 Let P be the following program:

$member(X, [X|T]) \leftarrow$
$member(X, [Y|T]) \leftarrow member(X, T)$

A simple well-founded order on atoms of the form $member(t_1, t_2)$ might be based on comparing the list length of the second argument.
We then define the wfo on atoms by $member(t_1, t_2) > member(s_1, s_2)$ iff $list\_length(t_2) > list\_length(s_2)$.
Based on that wfo, the goal $\leftarrow member(X, [a, b|T])$ can be unfolded into $\leftarrow member(X, [b|T])$ and further into $\leftarrow member(X, T)$ because the list length of the second argument strictly decreases at each step. However, $\leftarrow member(X, T)$ cannot be further unfolded into $\leftarrow member(X, T')$ because the list length does not strictly decrease.

Much more elaborate techniques based upon well-founded orders, which e.g. split the expressions into classes or continuously refine the orders during the unfolding process, exist and we refer the reader to [4, 34, 33, 32] for further details. These works also present a further refinement which, instead of requiring a decrease wrt every ancestor, only requires a decrease wrt the *covering ancestors*, i.e. one only compares with the ancestor atoms from which the current atom descends (via resolution).

However, in an on-line setting, well-founded orders are sometimes too rigid or too complex. Recently, well-quasi orders have therefore gained popularity to ensure on-line termination of program manipulation techniques [3, 41, 43, 26, 27, 13, 18, 1, 20, 46]. Indeed, as we will see below, well-quasi orders are often much more flexible than well-founded orders in an on-line context. We start examining them in the next section.

2 Well-quasi orders and homeomorphic embedding

From now on, we suppose familiarity with basic notions in logic programming [2, 29]. We also define an *expression* to be either a term, an atom, a literal, a conjunction, a disjunction or a program clause.

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

Definition 2.1 (poset) A *(non-strict) partial order* (also called a *quasi order*) on a set S is a reflexive and transitive binary relation on $S \times S$. A couple $S, \geq_S$ consisting of a set S and a partial order $\geq_S$ on S is called a *poset* or partially ordered set.

Henceforth, we will use symbols like $<$, $>$ (possibly annotated by some subscript) to refer to strict and $\leq$, $\geq$ to refer to non-strict partial orders. We will use either "directionality" as is convenient in the context.

Definition 2.2 (wqo) A poset $V, \leq_V$ is called *well-quasi-ordered (wqo)* iff for any infinite sequence of elements $e_1, e_2, \ldots$ in V there are $i < j$ such that $e_i \leq_V e_j$. We also say that $\leq_V$ is a *well-quasi order (wqo)* on V.

Note that e.g. Higman [15] used an alternate definition of well-quasi orders in terms of the "finite basis property" (or "finite generating set" in [19]). However, the above definition is the one most often used in the context of termination. Both definitions are equivalent by Theorem 2.1 in [15]. A different (but equivalent) definition of a wqo is given in [21, 47]: 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 insight can be used to dynamically construct well-founded orders for static termination analysis from well-quasi ones in a flexible way. More on this below.

The following lemmas about well-quasi orders will later enable us to prove Theorem 3.4 (and their proofs can be found in Appendix A), but they are also interesting in their own right.

It follows from Definitions 1.2 and 2.2 that if $\leq_V$ is a wqo then $<_V$ is a wfo, but not vice versa. The following shows how to obtain a wqo from a wfo.

Lemma 2.3 (wqo 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 wqo on V.

Lemma 2.4 Let $\preceq_V$ be a wqo on V and let $\sigma = v_1, v_2, \ldots$ be an infinite sequence of elements of V.

1. There exists an $i > 0$ such that the set $\{v_j \mid i < j \wedge v_i \preceq_V v_j\}$ is infinite.
2. There exists an infinite subsequence $\sigma^* = v_1^*, v_2^*, \ldots$ of σ such that for all $i < j$ we have $v_i^* \preceq_V v_j^*$.

Point 2 thus provides an alternate (and stronger) characterisation of well-quasi orders.

Lemma 2.5 (combination of wqo) Let $\preceq_V^1$ and $\preceq_V^2$ be wqo's on V. Then the quasi order $\preceq_V$ defined by $v_1 \preceq_V v_2$ iff $v_1 \preceq_V^1 v_2$ and $v_1 \preceq_V^2 v_2$, is also a wqo on V.

An interesting wqo is the homeomorphic embedding relation $\trianglelefteq$, which derives from results by Higman [15] and Kruskal [19]. It has been used in the context of term rewriting systems in [6, 7], and adapted for use in supercompilation ([45]) in [43]. Its usefulness as a stop criterion for partial evaluation is also discussed and advocated in [31]. Some complexity results can be found in [44] and [14] (also summarised in [31]).

The following is the definition from [43], which adapts the pure homeomorphic embedding from [7] by adding a rudimentary treatment of variables.

Definition 2.6 ($\trianglelefteq$) The *(pure) homeomorphic embedding* relation $\trianglelefteq$ on expressions is defined inductively as follows:

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 $s \trianglelefteq t$ then we also say that s is *embedded in* t or t is *embedding* s.

Example 2.7 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.
We also have e.g. that: $X \trianglelefteq X$, $p(X) \trianglelefteq p(f(Y))$, $p(X,X) \trianglelefteq p(X,Y)$ and $p(X,Y) \trianglelefteq p(X,X)$. Note that the homeomorphic embedding is also sometimes called a *divisibility ordering*, on account of the fact that when a natural number m divides another one n, then the prime divisors of m can be obtained from the ones of n by "striking out" some divisors. E.g. 10 divides 30 and 2×5 can be obtained from $2 \times 3 \times 5$ by striking out the 3.

Proposition 2.8 The relation $\trianglelefteq$ is a wqo on the set of expressions over a finite alphabet.

For a complete proof, reusing Higman's Lemma [15, 19] in a straightforward manner, see e.g. [24]. (For constructive proofs of Higman's Lemma [15] see [42, 36]. See also [12] and [40]. Another, non-constructive one can be found in [37].)

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 1.3, 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 $\leftarrow member(X,[a,b|T])$ of Example 1.3 can be unfolded into $\leftarrow member(X,[b|T])$ and further into $\leftarrow member(X,T)$ because $member(X,[a,b|T]) \not\trianglelefteq member(X,[b|T])$ as well as $member(X,[a,b|T]) \not\trianglelefteq member(X,T)$ and $member(X,[b|T]) \not\trianglelefteq member(X,T)$. However, $\leftarrow member(X,T)$ cannot be further unfolded into $\leftarrow member(X,T')$ because $member(X,T) \trianglelefteq member(X,T')$. Observe that, in contrast to Example 1.3, we did not have to chose how to measure which arguments.

The homeomorphic embedding relation is very generous. It will for example, without further refinement, permit the full unfolding of most terminating Datalog programs, the quicksort or even the mergesort program (the latter poses problems to some static termination analysis methods [38, 28]) — when the list to be sorted is known (see Appendix B). Also, it will allow a process to go from $p([],[a])$ to $p([a],[])$ but also the other way around. This illustrates the flexibility of using well-quasi orders compared to well-founded ones in an on-line setting, as there exists *no* wfo which will allow both these unfoldings.[1] 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 $p([a],[]) \rightarrow p([],[a]) \rightarrow p([a],[]) \rightarrow \ldots$[2]

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 $p([a],[]) \triangleright p([],a)$ (thus making

[1]To allow to go from $p([],[a])$ to $p([a],[])$ via a wfo $<$ we need to have that $p([],[a]) > p([a],[])$ (cf. Example 1.3). Now, if in another context we want to go from $p([a],[])$ to $p([],[a])$ this is clearly impossible (using $<$) because we cannot have $p([a],[]) > p([],[a])$.

[2]When using a wfo one has to compare only to the closest predecessor [33], because of the transitivity of the order and the strict decrease enforced at each step. However, wfo 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. [32, 33]).

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).

The homeomorphic embedding relation is also useful for handling structures other than expressions. It has e.g. been successfully applied in [26, 24, 27] 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 [31]). To conclude this section, we conjecture that the homeomorphic embedding relation might also be useful within debuggers to spot potential loops and alert the user.

3 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 2.7, the homeomorphic embedding relation $\trianglelefteq$ as defined in Definition 2.6 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, $p(X,Y) \trianglelefteq p(X,X)$ is acceptable, 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 clearly embeds $p(X,Y)$, but the reverse does not hold.

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

Example 3.1 Take for instance the following generalisation algorithm, which appears (in disguise) in a lot of partial deduction algorithms (e.g. [26, 24, 27]). (In that context $\mathcal{A}$ stands for 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 $\mathcal{A}' \supseteq \mathcal{A}$ such that 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. 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, [26, 24, 27] introduced the so called strict homeomorphic embedding as follows:

Definition 3.2 ($\trianglelefteq^+$) 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.

Example 3.3 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 $X \trianglelefteq^+ X$.

Also notice that, if we replace $\trianglelefteq$ of Example 3.1 by $\trianglelefteq^+$ we no longer have a problem with termination (see Section 6.2.4 of [24, 27] for a termination proof of an Algorithm containing the one of Example 3.1).

An alternate approach to Definition 3.2 — 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 [31] 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.[3]

Theorem 3.4 The relation $\trianglelefteq^+$ is a wqo on the set of expressions over a finite alphabet.

Proof In Appendix A. □

4 Extended homeomorphic embedding

Although $\trianglelefteq^+$ from Definition 3.2 has a more refined treatment of variables and has a much better behaviour wrt generalisation than $\trianglelefteq$ of Definition, 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 [19, 7] remedies this aspect. It even allows functors with variable arity[4] (the result of [15] can only be applied when the arities are fixed). We will show below how this definition can be adapted to a logic programming context.

However, there is another unsatisfactory aspect of $\trianglelefteq^+$. 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.

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

Definition 4.1($\trianglelefteq^*$) Given a wqo $\preceq$ on the functors, we define the *extended homeomorphic embedding* on expressions by the following rules:

[3] [31] 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 $\trianglelefteq^+$ of Definition 3.2.

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

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 $f \preceq g$ 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 $\langle s_1, \ldots, s_m \rangle$ is not a strict instance[5] of $\langle t_1, \ldots, t_n \rangle$

Notice that the above definition requires a wqo on the functors. If we have a finite alphabet, then equality is a wqo (and this is actually the way the pure homeomorphic embedding of [7] is obtained from the fully general homeomorphic embedding). In the context of e.g. partial deduction, we know that the functors occurring within the program (text) and goal to be analysed are of finite number. One might call these functors *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 functors for which e.g. a natural well-founded order exists (e.g. the natural numbers) one can use more refined wqo constructed from Lemma 2.3.

The reason why this recursive use of the "not strict instance" test was not incorporated in [26, 24, 27] was that the authors were not sure that $\trianglelefteq^*$ is actually a wqo (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 functors) 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 2.2. But can we construct some sequence for which $\trianglelefteq^*$ does not conform to Definition 2.2? The following Theorem 4.2 shows that such a sequence *cannot* be constructed. However, if we slightly strengthen point 3 of Definition 4.1 by requiring that $\langle s_1, \ldots, s_m \rangle$ is not a strict instance of $\langle t_{i_1}, \ldots, t_{i_m} \rangle$, 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, while using Definition 4.1 unmodified we have e.g. $A_1 \trianglelefteq^* A_2$ (but not $A_0 \trianglelefteq^* A_1$ or $A_0 \trianglelefteq^* A_2$).

Theorem 4.2 $\trianglelefteq^*$ is a wqo on expressions.

To prove Theorem 4.2 we need the following machinery. First we take a well-founded measure function from [11] (also in the extended version of [35]):

Definition 4.3 ($s(.)$, $h(.)$) Let *Expr* denote the sets of expressions. We define the function $s : Expr \to \mathbb{N}$ counting symbols by:

- $s(t) = 1 + s(t_1) + \ldots + s(t_n)$ if $t = f(t_1, \ldots, t_n)$, $n > 0$
- $s(t) = 1$ otherwise

Let the number of distinct variables in an expression t be $v(t)$. We now define the function $h : Expr \to \mathbb{N}$ by $h(t) = s(t) - v(t)$.

The well-founded measure function h has the property that $h(t) \geq 0$ for any expression t and $h(t) > 0$ for any non-variable expression t. The following important lemma is proven for $h(.)$ in [10] (see also [35]).

Lemma 4.4 If A and B are expressions such that B is strictly more general than A, then $h(A) > h(B)$.

[5] I.e. either $\langle s_1\theta, \ldots, s_m\theta \rangle = \langle t_1, \ldots, t_n \rangle$ for some θ or for no γ we have $\langle t_1\gamma, \ldots, t_n\gamma \rangle = \langle s_1, \ldots, s_m \rangle$. The latter will always be satisfied if $n \neq m$.

It follows that, for every expression A, there are no infinite chains of strictly more general expressions. We can now prove Theorem 4.2 as follows.

Proof of Theorem 4.2. We know by Higman-Kruskal's theorem [19] that without the extra condition of point 3 requiring $\langle s_1, \ldots, s_m \rangle$ not to be a strict instance of $\langle t_1, \ldots, t_n \rangle$ we have a wqo. Let us refer to this wqo by $\trianglelefteq$. We now simply define the following mapping $||.||$ from expressions to expressions, where we suppose that the natural numbers have been added to the set of functors (and were not previously in the set of functors). We also suppose that the wqo on functors is extended for the natural numbers by $i \preceq j$ if $i \leq j$.

- $||X|| = X$
- $||f(t_1, \ldots, t_n)|| = f(h, ||t_1||, \ldots, ||t_n||)$ where $h = h(\langle t_1, \ldots, t_n \rangle)$ (i.e. $h \in I\!N$).

Given any two expressions s and t we now have by Lemma 4.4 that $||s|| \trianglelefteq ||t|| \Rightarrow s \trianglelefteq^* t$:

- Diving (rule 2.) into the first argument of $||t||$ will never be successful, as we supposed that the natural numbers are distinct from the initial functors. Hence the same diving must be possible to deduce $s \trianglelefteq^* t$.
- For the coupling rule 3, the not strict instance test of $\trianglelefteq^*$ can only fail (in the sense of preventing $\trianglelefteq^*$ from holding) if f and g have the same arity (i.e. $n = m$). Therefore we must have $i_j = j$ and $h(\langle s_1, \ldots, s_m \rangle) \trianglelefteq h(\langle t_1, \ldots, t_n \rangle)$. The latter is equivalent to $h(\langle s_1, \ldots, s_m \rangle) \not> h(\langle t_1, \ldots, t_n \rangle)$ and hence, by Lemma 4.4, $\langle s_1, \ldots, s_m \rangle$ cannot be a strict instance of $\langle t_1, \ldots, t_n \rangle$).
  Similarly, we can prove inductively for each $||s_j|| \trianglelefteq ||t_{i_j}||$ (independently of whether $n = m$ or not) that $s_j \trianglelefteq^* t_{i_j}$ must hold. Hence, we can conclude that $f(s_1, \ldots, s_m) \trianglelefteq^* g(t_1, \ldots, t_n)$.

Thus, as $\trianglelefteq$ is a wqo so is $\trianglelefteq^*$ (for every infinite sequence of terms $t_1, t_2, \ldots$ we must have some $i < j$ such that $||t_i|| \trianglelefteq ||t_j||$ and hence $t_i \trianglelefteq^* t_j$ as well). □

5 Conclusion

In summary, the new embedding relation $\trianglelefteq^*$ inherits all the good properties of $\trianglelefteq^+$ while providing an even more refined treatment of (logical) variables and getting rid of the unsatisfactory, ad hoc aspects of $\trianglelefteq^+$.

Of course $\trianglelefteq^*$ is not the ultimate relation for ensuring on-line termination. Used on its own, $\trianglelefteq^*$ (as well as $\trianglelefteq^+$ and $\trianglelefteq$) will sometimes allow too much unfolding than desirable for efficiency concerns (i.e. more unfolding does not always imply a better specialised program) and additional considerations, like determinacy, have to be taken into account (see e.g. the experiments in [24, 27, 18]).

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 meta-interpretation tasks) [46]. The use of characteristic trees [24, 27] 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 [22] (in the context of static termination analysis of term rewrite systems) can be useful in an on-line setting.

We believe that $\trianglelefteq^*$ can be of value in other contexts and for other languages (such as in the context of partial evaluation of functional-logic programs [1] or of supercompilation [45] of functional programming languages, where — at specialisation time – variables also appear). We also believe that $\trianglelefteq^*$ provides both a theoretically and practically more satisfactory basis than $\trianglelefteq^+$ or $\trianglelefteq$ and we will incorporate it into the automatic partial deduction system ECCE [23] in the near future.

Acknowledgments

I would like to thank Jesper Jørgensen, Bern Martens, Jacques Riche and Morten Heine Sørensen for all the discussions, comments and joint research which led to this paper. Bern Martens also provided very useful comments on a draft of this paper. Finally, I want to thank anonymous referees for their comments.

References

[1] 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*, 1997. To appear.

[2] 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.

[3] R. Bol. Loop checking in partial deduction. *The Journal of Logic Programming*, 16(1&2):25–46, 1993.

[4] 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.

[5] 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.

[6] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[7] 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.

[8] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

[9] 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.

[10] J. Gallagher and M. Bruynooghe. Some low-level transformations for logic programs. In M. Bruynooghe, editor, *Proceedings of Meta90 Workshop on Meta Programming in Logic*, pages 229–244, Leuven, Belgium, 1990.

[11] J. Gallagher and M. Bruynooghe. The derivation of an algorithm for program specialisation. *New Generation Computing*, 9(3 & 4):305–333, 1991.

[12] J. H. Gallier. What's so special about Kruskal's theorem and the ordinal Γ_0? A survey of some results in proof theory. Technical report, University of Pennsylvania, October 1993.

[13] 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. Extended version as Technical Report CW 226, K.U. Leuven. Accessible via `http://www.cs.kuleuven.ac.be/~lpai`.

[14] J. Gustedt. *Algorithmic Aspects of Ordered Structures.* PhD thesis, Technische Universität Berlin, 1992.

[15] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 2:326–336, 1952.

[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 of the International Workshop on Logic Program Synthesis and Transformation (LOPSTR'96)*, LNCS 1207, pages 59–82, Stockholm, Sweden, August 1996. Springer-Verlag. Also in the Proceedings of BENELOG'96. Extended version as Technical Report CW 242, K.U. Leuven.

[19] J. B. Kruskal. Well-quasi ordering, the tree theorem, and Vazsonyi's conjecture. *Tansactions 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, *Pre-Proceedings of the International Workshop on Logic Program Synthesis and Transformation (LOPSTR'97)*, Leuven, Belgium, July 1997.

[21] 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.

[22] P. Lescanne. Well rewrite orderings and well quasi-orderings. Technical Report N^o 1385, INRIA-Lorraine, France, January 1991.

[23] M. Leuschel. The ECCE partial deduction system and the DPPD library of benchmarks. Obtainable via `http://www.cs.kuleuven.ac.be/~lpai`, 1996.

[24] M. Leuschel. *Advanced Techniques for Logic Program Specialisation.* PhD thesis, K.U. Leuven, May 1997. Accessible via `http://www.cs.kuleuven.ac.be/~michael`.

[25] M. Leuschel. Extending homeomorphic embedding in the context of logic programming. Technical Report CW 252, Departement Computerwetenschappen, K.U. Leuven, Belgium, June (revised August) 1997.

[26] 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. Extended version as Technical Report CW 220, K.U. Leuven. Accessible via `http://www.cs.kuleuven.ac.be/~lpai`.

[27] M. Leuschel, B. Martens, and D. De Schreye. Controlling generalisation and polyvariance in partial deduction of normal logic programs. Technical Report CW 248, Departement Computerwetenschappen, K.U. Leuven, Belgium, February 1996. Submitted to ACM Toplas.

[28] N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. Unfolding the mystery of mergesort. In N. Fuchs, editor, *Pre-Proceedings of the International Workshop on Logic Program Synthesis and Transformation (LOPSTR'97)*, Leuven, Belgium, July 1997.

[29] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[30] J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *The Journal of Logic Programming*, 11(3& 4):217–242, 1991.

[31] R. Marlet. *Vers une Formalisation de l'Évaluation Partielle.* PhD thesis, Université de Nice - Sophia Antipolis, December 1994.

[32] B. Martens. *On the Semantics of Meta-Programming and the Control of Partial Deduction in Logic Programming.* PhD thesis, K.U. Leuven, February 1994.

[33] B. Martens and D. De Schreye. Automatic finite unfolding using well-founded measures. *The Journal of Logic Programming*, 28(2):89–146, August 1996. Abridged and revised version of Technical Report CW180, Departement Computerwetenschappen, K.U.Leuven, October 1993, accessible via `http://www.cs.kuleuven.ac.be/~lpai`.

[34] 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.

[35] 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. Extended version as Technical Report CSTR-94-16, University of Bristol.

[36] C. R. Murthy and J. R. Russel. A constructive proof of Higman's lemma. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 257–267, Alamitos, California, 1990. IEEE Computer Society Press.

[37] C. Nash-Williams. On well-quasi-ordering finite trees. *Proc. Cambridge Phil. Soc*, 59:833–835, 1963.

[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] J. Riche. *Decidability, Complexity and Automated Reasoning in Relevant Logic.* PhD thesis, Australian National University, September 1991.

[41] D. Sahlin. Mixtus: An automatic partial evaluator for full Prolog. *New Generation Computing*, 12(1):7–51, 1993.

[42] S. G. Simpson. Ordinal numbers and the Hilbert basis theorem. *Journal of Symbolic Logic*, 53(3):961–974, 1988.

[43] 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.

[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, *Pre-Proceedings of the International Workshop on Logic Program Synthesis and Transformation (LOPSTR'97)*, Leuven, Belgium, July 1997. Also as Technical Report CW 251, K.U.Leuven.

[47] A. Weiermann. Complexity bounds for some finite forms of Kruskal's theorem. *Journal of Symbolic Computation*, 18(5):463–488, November 1994.

A Assorted proofs

Lemma 2.3 (wqo 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 wqo on V.

Proof Suppose that there is an infinite sequence $v_1, v_2, \ldots$ of elements of V such that, for all $i < j$, $v_i \not\preceq_V v_j$. By definition this means that, for all $i < j$, $v_i >_V v_j$. In particular this means that we have an infinite sequence with $v_i >_V v_{i+1}$, for all $i \geq 1$. We thus have a contradiction with Definition 1.2 of a well-founded order and $\preceq_V$ must be a wqo on V. $\square$

Lemma 2.4 Let $\preceq_V$ be a wqo on V and let $\sigma = v_1, v_2, \ldots$ be an infinite sequence of elements of V.

1. There exists an $i > 0$ such that the set $\{v_j \mid i < j \wedge v_i \preceq_V v_j\}$ is infinite.
2. There exists an infinite subsequence $\sigma^* = v_1^*, v_2^*, \ldots$ of σ such that for all $i < j$ we have $v_i^* \preceq_V v_j^*$.

Proof The proof will make use of the axiom of choice at several places. Given a sequence ρ, we denote by $\rho_{v' \preceq_V \circ}$ the subsequence of ρ consisting of all elements v'' which satisfy $v' \preceq_V v''$. Similarly, we denote by $\rho_{v' \not\preceq_V \circ}$ the subsequence of ρ consisting of all elements v'' which satisfy $v' \not\preceq_V v''$.
Let us now prove point 1. Assume that such an i does not exist. We can then construct the following infinite sequence $\sigma_0, \sigma_1, \ldots$ of sequences inductively as follows:
- $\sigma^0 = \sigma$
- if $\sigma^i = v_i'.\rho^i$ then $\sigma^{i+1} = \rho^i_{v_i' \not\preceq_V \circ}$

All σ_i are indeed properly defined because at each step only a finite number of elements are removed (by going from ρ^i to $\rho^i_{v_i' \not\preceq_V \circ}$; otherwise we would have found an index i satisfying point 1). Now the infinite sequence $v_1', v_2', \ldots$ has by construction the property that, for $i < j$, $v_i' \not\preceq_V v_j'$. Hence $\preceq_V$ cannot be a wqo on V and we have a contradiction.
We can now prove point 2. (As we found out, this point is actually a corollary of point iv of Theorem 2.1 in [15]. But as no formal proof is provided by [15], we include one for completeness.) Let us construct $\sigma^* = v_1^*, v_2^*, \ldots$ inductively as follows:
- $\sigma^0 = \sigma$
- if $\sigma^i = r_1, r_2, \ldots$ then $v_{i+1}^* = r_k$ and $\sigma^{i+1} = \rho^i_{r_k \preceq_V \circ}$ where k is the first index satisfying the requirements of point 1 for the sequence σ^i (i.e. $\{r_j \mid k < j \wedge r_k \preceq_V r_j\}$ is infinite) and where $\rho^i = r_{k+1}, r_{k+2}, \ldots$

By point 1 we know that each $\rho^i_{r_k \preceq_V \circ}$ is infinite and σ^* is thus an infinite sequence which, by construction, satisfies $v_i^* \preceq_V v_j^*$ for all $i < j$. $\square$

Lemma 2.5 (combination of wqo) Let $\preceq_V^1$ and $\preceq_V^2$ be wqo's on V. Then the quasi order $\preceq_V$ defined by $v_1 \preceq_V v_2$ iff $v_1 \preceq_V^1 v_2$ and $v_1 \preceq_V^2 v_2$, is also a wqo on V.
Proof (As we found out, this lemma is actually a corollary of Theorem 2.3 in [15]. But as no formal proof is given in [15], we include one for completeness.) Let σ be any infinite sequence of elements from V. We can apply point 2 of Lemma 2.4 to obtain the infinite subsequence $\sigma^* = v_1^*, v_2^*, \ldots$ of σ such that for all $i < j$ we have $v_i^* \preceq_V^1 v_j^*$. Now, as $\preceq_V^2$ is also a wqo we know that, for some $i < j$, $v_i^* \preceq_V^2 v_j^*$ holds as well. Hence, for these particular indices, $v_i^* \preceq_V v_j^*$ and $\preceq_V$ satisfies the requirements of a wqo on V. $\square$

We now prove Theorem 3.4 in a simpler way than in [26] (the following proof can also be found in [24, 27]).
Theorem 3.4 The relation $\trianglelefteq^+$ is a wqo on the set of expressions over a finite alphabet.
Proof of Theorem 3.4. $\trianglelefteq^+$ can be expressed as a combination of two quasi orders on expressions: $\trianglelefteq$ and $\preceq_{NotStrictInst}$ where $A \preceq_{NotStrictInst} B$ iff $B \not\prec A$ (i.e. B is not strictly more general than A or equivalently A is not a strict instance of B). By Lemma 4.4 we know

that $\prec$ is a well-founded order on expressions. Hence by Lemma 2.3 $\preceq_{NotStrictInst}$ is a wqo on expressions. By Proposition 2.8 we also have that $\trianglelefteq$ is a wqo on expressions (given a finite underlying alphabet). Hence we can apply Lemma 2.5 to deduce that $\trianglelefteq^+$ is also a wqo on expressions over a finite alphabet. □

B Small Experiments with the ECCE system

The objective 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 e.g. the one used by Mixtus [41], can handle the examples below as well). For that we experiment with the ECCE partial deduction system [23] using an unfolding rule based on $\trianglelefteq^+$ which allows the selection of determinate literals or left-most literals within a goal, given that no covering ancestor [4] is embedded (via $\trianglelefteq^+$).

To ease readability, the specialised programs are presented in unrenamed form.

Let us first illustrate the generosity of $\trianglelefteq^+$ on the *quicksort* program (as taken from [38]):

```
qsort([],[]).
qsort([H|L],S) :-
       split(H,L,[],[],A,B),
       qsort(A,A1),qsort(B,B1),
       append(A1,[H|B1],S).

append([],L,L).
append([H|X],Y,[H|Z]) :- append(X,Y,Z).

split(H,[X|L],A,B,A1,B1) :- X =< H, split(H,L,[X|A],B,A1,B1).
split(H,[X|L],A,B,A1,B1) :- X > H, split(H,L,A,[X|B],A1,B1).
split(H,[],A,B,A,B).
```

The partial evaluation query:

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

The resulting specialised program is as follows (and full unfolding has been achieved):

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

Next, let us take the *mergesort* program, which is somewhat problematic for a lot of static termination analysis methods [38, 28].

```
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).

Finally, take this small Datalog program computing the transitive closure of a graph:

```
arc(a,b).
arc(b,c).

trans(X,Y) :- arc(X,Y).
trans(X,Z) :- arc(X,Y),trans(Y,Z).
```

The partial evaluation query:

```
?- trans(a,X).
```

Again full unfolding was accomplished, as illustrated by the following specialised program:

```
trans(a,b).
trans(a,c).
```