Ecological Partial Deduction: Preserving Characteristic Trees Without Constraints

Michael Leuschel

K.U. Leuven, Department of Computer Science
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
e-mail: michael@cs.kuleuven.ac.be

Abstract. A partial deduction strategy for logic programs usually uses an abstraction operation to guarantee the finiteness of the set of atoms for which partial deductions are produced. Finding an abstraction operation which guarantees finiteness and does not loose relevant information is a difficult problem. In earlier work Gallagher and Bruynooghe proposed to base the abstraction operation on characteristic paths and trees. A characteristic tree captures the relevant structure of the generated partial SLDNF-tree for a given goal. Unfortunately the abstraction operations proposed in the earlier work do not always produce more general atoms and do not always preserve the characteristic trees. This problem has been solved for purely determinate unfolding rules and definite programs in [12, 13] by using constraints inside the partial deduction process.
In this paper we propose an alternate solution which achieves the preservation of characteristic trees for *any* unfolding rule, *normal* logic programs (it can even handle some built-in's if so desired) and *without* adding constraints to the partial deduction process (making the re-use of existing unfolding techniques very simple). We thus provide a powerful, generally applicable and elegant abstraction operation for partial deduction providing a fine-grained and terminating control of polyvariance.

1 Introduction

Partial evaluation has received considerable attention in logic programming (e.g. [5, 8, 22, 24]). In the context of pure logic programs, partial evaluation is often referred to as *partial deduction*, a convention we will also adhere to in this paper. An important milestone is [17], where firm theoretical foundations for partial deduction are established. It introduces the notions of *independence* and *closedness*, which are properties of the set of atoms for which the partial deduction is performed. Under these conditions, soundness and completeness of the transformed program are guaranteed. In the light of these conditions, a key problem in partial deduction is: given a set of atoms of interest, $\mathcal{A}$, provide a *terminating* procedure that computes a new set of atoms, $\mathcal{A}'$, and a partial deduction for the atoms in $\mathcal{A}'$, such that:

- every atom in $\mathcal{A}$ is an instance of an atom in $\mathcal{A}'$, and
- the closedness and independence conditions are satisfied.

Moving from the initial set $\mathcal{A}$ to the new set $\mathcal{A}'$ requires the (repeated) application of an abstraction operation. The problem of finding a proper abstraction operation is closely related to the problem of *polyvariance*, i.e. controlling how many different specialised versions of a given predicate should be generated. A good abstraction operation should, while guaranteeing termination, produce enough polyvariance to ensure satisfactory specialisation.

An approach which aims at achieving all these goals in a refined way is that of Gallagher and Bruynooghe ([7, 4]). Its abstraction operation is based on the notions of *characteristic paths* and *characteristic trees*. Intuitively, two atoms of $\mathcal{A}$ are replaced by their *msg*[1] in $\mathcal{A}'$, if their (incomplete) SLDNF-trees have an identical structure (this structure is referred to as the characteristic tree). Unfortunately, although the approach is conceptually appealing, several errors turn up in arguments provided in [7, 4]. These errors invalidate the termination proofs as well as the arguments regarding precision and preservation of specialisation under the abstraction operation.

In [12, 13], Leuschel and De Schreye have significantly adapted the approach to overcome these problems. An alternative abstraction operation has been introduced, which is based on so called *negative binding constraints*. The partial deduction procedure is then formulated in terms of a special purpose constraint logic programming language. The adapted approach allows to solve all problems with the original formulations in [7, 4], without loosing the claimed termination and precision properties. Unfortunately the approach is (exactly like [7]) limited to purely determinate unfolding rules (i.e. without lookahead for detecting determinacy) and is restricted to definite logic programs. These restrictions limit the practical applicability of the method.

In this paper we will present an alternate technique for *normal* logic programs (with built-in's if so desired) which is valid for *any* unfolding rule and which preserves characteristic trees *without* using constraints. As such we use the central ideas of [12, 13, 7, 4] in a novel way to provide a practically useful system which can make use of existing unfolding technology (e.g. [3, 19, 18]) in a straightforward way. The overview of the paper is as follows. In Sect. 2 we introduce the concept of a characteristic tree and give some motivations. In Sect. 3 we present the new technique of partial deduction working on characteristic atoms. In Sect. 4 we put this idea in practice and present an algorithm for partial deduction. Some examples and results are also presented. We provide some concluding remarks in Sect. 5.

2 Preliminaries and Motivations

Throughout this paper, we suppose familiarity with basic notions in logic programming ([16]) and partial deduction ([17]). Notational conventions are standard and self-evident. In particular, in programs, we denote variables through strings starting with (or usually just consisting of) an upper-case symbol, while

[1] The *most specific generalisation*, also known as anti-unification or least general generalisation, see for instance [10].

the notations of constants, functions and predicates begin with a lower-case character. Unless stated explicitly otherwise, the terms "(logic) program" and "goal" will refer to a *normal* logic program and goal, respectively.

Given a program P and a goal G, partial deduction produces a new program P' which is P "specialised" to the goal G. The underlying technique is to construct "incomplete" SLDNF-trees for a set of atoms $\mathcal{A}$ to be specialised and extract the program P' from these incomplete search trees. An *incomplete* SLDNF-tree is an SLDNF-tree which, in addition to success and failure leaves, may also contain leaves where no literal has been selected for a further derivation step. Leaves of the latter kind will be called *dangling*. Under the conditions stated in [17], namely closedness and independence, correctness of the specialised program is guaranteed. In the context of partial deduction, incomplete SLDNF-trees are obtained by applying an unfolding rule, defined as follows:

Definition 1. An *unfolding rule* U is a function which given a program P and a goal G returns a finite, possibly incomplete and non-trivial[2] SLDNF-tree for $P \cup \{G\}$.

Definition 2. Let P be a program and A an atom. Let τ be a finite, incomplete SLDNF-tree for $P \cup \{\leftarrow A\}$ in which A has been selected in the root node.[3] Let $\leftarrow G_1, \ldots, \leftarrow G_n$ be the goals in the (non-root) leaves of the non-failing branches of τ. Let $\theta_1, \ldots, \theta_n$ be the computed answers of the derivations from $\leftarrow A$ to $\leftarrow G_1, \ldots, \leftarrow G_n$ respectively. Then the set of resultants $resultants(\tau)$ is defined to be $\{A\theta_1 \leftarrow G_1, \ldots, A\theta_n \leftarrow G_n\}$.

Partial deduction, as defined e.g. in [17, 2], uses the resultants for a given set of atoms $\mathcal{A}$ to construct the specialised program (and for each atom in $\mathcal{A}$ a different specialised predicate definition is generated). Under the conditions stated in [17], namely closedness and independence, correctness of the specialised program is guaranteed. The problem of *control of polyvariance* of partial deduction consists in coming up with a *terminating* procedure to produce a *finite* set of atoms $\mathcal{A}$ which satisfies the *correctness* conditions of [17] while at the same time providing as much potential for *specialisation* as possible (usually the more instantiated[4] the set $\mathcal{A}$ is, the more specialisation can be performed). Most approaches to the control of polyvariance in the literature so far are based on the syntactic structure of the atoms to be specialised (like e.g. the 1 *msg* per predicate approach in [19] or dynamic renaming of [1], the latter not guaranteeing termination). The following example shows that the syntactic structure alone does not provide enough details for a satisfactory control of polyvariance.

Example 1. Let P be the append program (where clauses have been numbered):

(1) $append([], Z, Z) \leftarrow$
(2) $append([H|X], Y, [H|Z]) \leftarrow append(X, Y, Z)$

[2] A trivial SLDNF-tree is one whose root is a dangling leaf. See also Definition 2.
[3] If this is not the case we will get the problematic resultant $A \leftarrow A$.
[4] $\mathcal{A}$ is more instantiated than $\mathcal{A}'$, iff every atom in $\mathcal{A}$ is an instance of an atom in $\mathcal{A}'$.

Also let $B = append([a], X, Y)$, $C = append(X, [a], Y)$ and $\mathcal{A} = \{B, C\}$. Typically a partial deducer will unfold the atoms of $\mathcal{A}$ as depicted in Fig. 1, yielding the SLDNF-trees τ_B and τ_C. These two SLDNF-trees, as well as their resultants, have a very different structure. For $append([a], X, Y)$ we obtain for $resultants(\tau_B)$ the single fact:

$$append([a], X, [a|X]) \leftarrow$$

while for $append(X, [a], Y)$ we obtain the recursive $resultants(\tau_C)$:

$$append([], [a], [a]) \leftarrow$$
$$append([H|X], [a], [H|Z]) \leftarrow append(X, [a], Z)$$

So in this case it is vital for precision to produce separate specialised versions for the dependent atoms B and C. However it is very easy to come up with another predicate definition of *append* (which then no longer appends two lists but for instance, as in the program below, finds common elements at common positions) such that the incomplete SLDNF-trees τ_B and τ_C for B and C are almost fully identical:

(1') $append^*([X|T_X], [X|T_Y], [X]) \leftarrow$
(2') $append^*([X|T_X], [Y|T_Y], E) \leftarrow append^*(T_X, T_Y, E)$

In that case it is not useful to keep different specialised versions for B and C because one more general version can be used without loss of specialisation:

$$append^*([a|T_1], [a|T_2], [a]) \leftarrow$$

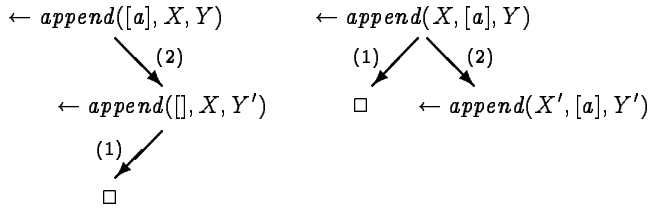

Fig. 1. SLDNF-trees τ_B and τ_C for example 1

This illustrates that the syntactic structures of B and C alone provide insufficient information for a satisfactory control of polyvariance. It is much more important to know how the syntactic structure behaves in the context of the program to be specialised. This information can be obtained by using the following characterisation of the associated incomplete SLDNF-trees (and as such also of the resultants they represent). The definitions are adapted from [4, 12, 13].

Definition 3. Let G_1 be a goal and let P be a program whose clauses are numbered. Let $G_1, \ldots, G_n$ be a finite, incomplete SLDNF-derivation of $P \cup \{G_1\}$. The *characteristic path* of the derivation is the sequence $(l_1, c_1), \ldots, (l_{n-1}, c_{n-1})$, where l_i is the position of the selected literal in G_i, and c_i is defined as:

- if the selected literal is an atom, then c_i is the number of the clause chosen to resolve with G_i.
- if the selected literal is $\neg p(\bar{t})$, then c_i is the predicate p.

The set of all characteristic paths for a given program P and a given goal G will be denoted by $chpaths(G, P)$.

Note that if two non-failing derivations for two goals with the same number of literals (and for the same program P) have the same characteristic path then they are either both successful or both incomplete (for a proof see [13]). This justifies the fact that the leaves of derivations are not registered in characteristic paths.

Definition 4. Let G be a goal, P a program and U an unfolding rule. Then *the characteristic tree τ of G (in P) via U* is the set of characteristic paths of the non-failing derivations of the incomplete SLDNF-tree obtained by applying U to G (in P). We introduce the notation $chtree(G, P, U) = \tau$. We also say that τ is *a characteristic tree of G (in P)* if it is *the* characteristic tree for some unfolding rule U. Also τ is *a characteristic tree* if it is *a* characteristic tree of some G in some P.

Although a characteristic tree only contains a collection of characteristic paths the actual tree structure can be reconstructed without ambiguity. The "glue" is provided by the clause numbers inside the characteristic paths (branching in the tree is indicated by differing clause numbers).

Characteristic trees are a very interesting abstraction because there is a close link between characteristic trees and the specialisation performed locally by partial deduction. If two atomic goals have the same characteristic tree then:

- the same branches have been pruned by partial deduction. The pruning of branches at partial deduction time is one important aspect of the specialisation. For instance in Example 1 and Fig. 1 we can see that two branches have been pruned for B (thereby removing recursion) whereas no pruning could be performed for C.
- the atoms have been unfolded in exactly the same way, e.g. the same clauses have been resolved with literals in the same position in that process. This captures the computation steps that have already been performed at partial deduction time, which is another important aspect of specialisation.
- the resultants have the same structure, differing only by the particular instantiations. For instance this captures that, in Example 1, a single fact will be generated for B whereas a potentially recursive predicate definition with two clauses will be generated for C.

As such a characteristic tree is an almost perfect characterisation of the specialisation that has been performed *locally* on an atom. When two atoms have the same characteristic tree then a single predicate definition can in principle be used without local specialisation loss.[5] For more details about the interest of characteristic trees and local and global precision aspects, we refer to [13, 7, 4].

[5] Sometimes atoms with different characteristic trees have (almost) identical resultants (due to independence of the computation rule) and could therefore also be replaced

The approach to partial deduction in [4] (see also [5]) uses characteristic trees to classify the atoms to be specialised according to their characteristic tree. The basic idea is to have only one specialised version for each characteristic tree. If the number of characteristic trees is finite[6] we get a control of polyvariance which ensures termination and correctness of the specialised program while at the same time (hopefully) ensuring that different specialised version are produced if the associated predicate definitions have a different form. The basic outline of the algorithm in [4, 5] is as follows:

1. For a set of atoms $\mathcal{A}$ to be specialised: apply an unfolding rule to obtain a set of incomplete SLDNF-trees.
2. Add the atoms occurring in the bodies of the resultants of these SLDNF-trees to $\mathcal{A}$.
3. Apply an *abstraction operation* which, based on characteristic trees, will produce a new set $\mathcal{A}'$ such that every atom in $\mathcal{A}$ is an instance of an atom in $\mathcal{A}'$.[7]
4. If $\mathcal{A}'$ is different from $\mathcal{A}$ restart the procedure, otherwise apply a renaming transformation (to ensure independence) and generate the specialised program by taking the resultants of the incomplete SLDNF-trees.

The following example illustrates the above procedure.

Example 2. Let P be the following program:

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

Let $\mathcal{A} = \{member(a, [a, b]), member(a, [a])\}$. As can be seen in Fig. 2 both these atoms have the same characteristic tree $\tau = \{((1, 1))\}$ (given an unfolding rule which unfolds deep enough to detect the failing branches). The method of [4, 5] would thus decide to abstract both atoms producing the generalisation $\mathcal{A}' = \{member(a, [a|T])\}$ (the *msg* has been calculated). Unfortunately (as already pointed out for other examples in [12, 13]) the generalisation has a different characteristic tree τ' (depending on the unfolding rule: $\{((1, 1)), ((1, 2))\}$ or $\{((1, 1)), ((1, 2), (1, 1)), ((1, 2), (1, 2))\}$ or something even deeper).

This loss of precision leads to sub-optimal specialised programs. In this case the atom $member(a, T)$ would be added to $\mathcal{A}$ at the following step of the algorithm. This atom (usually) also has τ' as characteristic tree. Hence the final set $\mathcal{A}'$ is $\{member(a, L)\}$ (the *msg* of $\{member(a, [a|T]), member(a, T)\}$) and we obtain the following sub-optimal specialisation:

---

by a single predicate definition. Normalising characteristic trees (after unfolding) by imposing e.g. a left-to-right ordering of selected literals and delaying the selection of negative literals to the end solves this problem (see also [13]). Thanks to Maurice Bruynooghe for pointing this out.

[6] In [4] this is ensured by using a depth bound on characteristic trees.

[7] Note that without an abstraction operation the procedure is not guaranteed to terminate.

(1') $member(a, [a|T]) \leftarrow$
(2') $member(a, [X|T]) \leftarrow member(a, T)$

So although partial deduction was able to conclude that both $member(a, [a, b])$ and $member(a, [a])$ have only one non-failing resolvent and are determinate, this information has been lost due to an imprecision of the abstraction operator leading to a sub-optimal program in which the determinacy is not explicit (and redundant computation steps will occur at run-time). Note that, for the set of atoms $\mathcal{A} = \{member(a, [a, b]), member(a, [a])\}$, an "optimal" program would just consist of the clause (1').

Unfortunately imprecision is not the only problem. The fact that the generalisation does not preserve the characteristic trees can also lead to non-termination of the partial deduction process. An example can be found in [13]. So it is vital, as well for precision and termination, that the abstraction operation preserves characteristic trees.

To remedy this problem (while still ensuring termination of partial deduction in general) we would have to replace the atoms in $\mathcal{A}$ by a more general atom having the same characteristic tree. Unfortunately in a general context this is impossible, no such generalisation exists. Also notice that the unfoldings in Fig. 2 are not "purely" determinate (because a lookahead has been used to decide about determinacy, see [4, 12, 13]). Hence the method of [12, 13] cannot be applied (unless failing branches are incorporated into the characteristic trees and a post-processing phase is added which removes the unnecessary polyvariance).

In the following sections we will present an elegant solution to this problem in general and this example in particular.

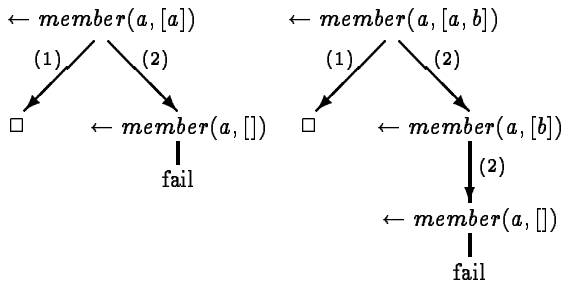

Fig. 2. Incomplete SLDNF-trees for example 2

3 Partial Deduction with Characteristic Atoms

The basic idea of the solution proposed in this paper is to simply *impose* characteristic trees on the generalised atoms. This amounts to associating characteristic

trees with the atoms to be specialised. This will allow the preservation of characteristic trees in a straightforward way without having to construct intricate generalisations. Let us provide the definitions.

Definition 5. A *P-characteristic atom*, for a given program P, is a pair (A, τ) consisting of an atom A and a characteristic tree τ with $\tau \subseteq chpaths(P, \leftarrow A)$.

Often, when the context allows it, we will drop the P annotation and simply refer to *characteristic* atoms. Also note that τ is not necessarily a characteristic tree of $\leftarrow A$ in P. The following definition associates a set of concretisations with each characteristic atom.

Definition 6. An atom A is a *precise concretisation* of a P-characteristic atom (A', τ') iff A is an instance of A' and for some unfolding rule U we have that $chtree(\leftarrow A, P, U) = \tau'$. An atom B is a *concretisation* of (A', τ') iff it is an instance of a precise concretisation of (A', τ').

A P-characteristic atom can thus be seen as standing for a (possibly infinite) set of atoms, namely the concretisations according to the above definition.

E.g. both $member(a, [a])$ and $member(a, [a, b])$ of Example 2 are precise concretisations of $(member(a, [a|T]), \{((1, 1))\})$. This already hints at a possible solution for Example 2. Also note that it is decidable whether or not an atom is a (precise) concretisation of a characteristic atom.

Definition 7. Let (A, τ) be a P-characteristic atom. Then $\delta(P, (A, \tau))$ is the set of all (necessarily non-failing) SLDNF-derivations for $P \cup \{\leftarrow A\}$ such that their characteristic paths are in τ.

A characteristic atom (A, τ) uniquely determines a set of resultants (which in the definite case is a subset of the resultants for A and some properly chosen unfolding rule):

Definition 8. Let (A, τ) be a characteristic atom and P be a normal program. Let $\{\delta_1, \ldots, \delta_n\}$ be the SLDNF-derivations in $\delta(P, (A, \tau))$ and let $\leftarrow G_1, \ldots, \leftarrow G_n$ be the goals in the leaves of these derivations. Let $\theta_1, \ldots, \theta_n$ be the computed answers of the derivations from $\leftarrow A$ to $\leftarrow G_1, \ldots, \leftarrow G_n$ respectively. Then the set of resultants $\{A\theta_1 \leftarrow G_1, \ldots, A\theta_n \leftarrow G_n\}$ is called the *partial deduction of* (A, τ) *in* P. Every atom occurring in some of the G_i will be called a *body atom (in P)* of (A, τ). We will denote the set of such body atoms by $BA_P(A, \tau)$.

For example the partial deduction of $(member(a, [a|T]), \{((1, 1))\})$ in the program P of Example 2 will be $\{member(a, [a|T]) \leftarrow\}$. Note that it is different from any set of resultants that can be obtained for incomplete SLDNF-trees of the normal atom $member(a, [a|T])$. However the partial deduction is valid for any concretisation (as defined in Definition 6) of the characteristic atom $(member(a, [a|T]), \{((1, 1))\})$.

As can already be guessed from the above definition the main idea pursued in this paper will be to generate a partial deduction not for a set of atoms

but for a set of *characteristic* atoms. As such the same atom A might occur in several characteristic atoms with entirely different characteristic trees. For instance $member(a, L)$ might occur as well in $c_1 = (member(a, L), \{((1, 1))\})$, $c_2 = (member(a, L), \{((1, 2))\})$ as in $c_3 = (member(a, L), \{((1, 1)), ((1, 2))\})$. The set of precise concretisations of these characteristic atoms is disjoint for a fixed unfolding rule and the partial deductions will be completely different. In order to guarantee correctness of the specialised program we have to handle some form of renaming to ensure that the correct partial deductions are called. For example (for any unfolding rule) $member(a, [a])$ is a precise concretisation of c_1 and the call $\leftarrow member(a, [a])$ has to be mapped to the resultants of c_1 (and never to c_2). Similarly (for an unfolding rule which unfolds deep enough) $\leftarrow member(a, [b, a])$ has to be mapped to the resultants of c_2 (and never to c_1).

In addition to renaming we will also incorporate argument filtering (which often greatly improves the efficiency of the specialised program, see for instance [6] or also [23] where filtering is obtained automatically when using folding to simulate partial evaluation).

Definition 9. Let A, B be atoms such that $A = B\theta$ and let the list of distinct variables of B (ordered according to the first occurrence in B) be $(X_1, \ldots, X_n)$. Let p be a predicate symbol with arity n. Then $A[B \mapsto p] = p(X_1, \ldots, X_n)\theta$.

For example $q(a, f(b))[q(a, f(Y)) \mapsto p] = p(Y)\{Y/b\} = p(b)$. Let us now define a renaming (and filtering) operation adapted for characteristic atoms.

Definition 10. Let $\mathcal{A} = \{(A_1, \tau_1), \ldots, (A_n, \tau_n)\}$ be a finite set of characteristic atoms and F a set of first-order formulas. A *renaming function for $\mathcal{A}$ wrt F* is a function ρ which assigns a distinct predicate symbol not occuring in F to each element in $\mathcal{A}$ such that the arity of $\rho((A_j, \tau_j))$ is equal to the number of distinct variables in A_j.

Definition 11. Let $\mathcal{A} = \{(A_1, \tau_1), \ldots, (A_n, \tau_n)\}$ be a finite set of characteristic atoms and F a set of first-order formulas. Let ρ be a renaming function for $\mathcal{A}$ wrt F. Let F' be obtained from F by replacing each atom A occuring in F which is a concretisation of some (A_j, τ_j) (and possibly some other elements of $\mathcal{A}$ as well, preference should normally be given to precise concretisations) by $A[A_j \mapsto \rho((A_j, \tau_j))]$. Then F' is a *renaming of F wrt $\mathcal{A}$ and ρ.*

Notice that in order for A to be a precise concretisation of (A_j, τ_j) the characteristic tree of A has to be τ_j for some U (see Definition 6). This ensures that all the unfolding steps in τ_j (also the ones selecting negative literals) are valid (and sufficient) for A and instances of A. We are now in a position to generate a partial deduction for a set of characteristic atoms.

Definition 12. Let $\mathcal{A} = \{(A_1, \tau_1), \ldots, (A_n, \tau_n)\}$ be a finite set of characteristic atoms, P be a program, U an unfolding rule and let ρ be a renaming function for $\mathcal{A}$ wrt P. For each $i \in \{1, \ldots, n\}$ let $R_i = \{A_i\theta_{1,i} \leftarrow G_{1,i}, \ldots, A_i\theta_{n,i} \leftarrow G_{n,i}\}$ be the partial deduction of (A_i, τ_i) in P. We define $R_i^\rho = \{F_{1,i} \leftarrow G_{1,i}, \ldots, F_{n,i} \leftarrow$

$G_{n,i}\}$ where $F_{j,i} = A_i\theta_{j,i}[A_i \mapsto \rho((A_i, \tau_i))]$. We also define the intermediate program $P' = \{R_i^\rho \mid i \in \{1, \ldots, n\}\}$. A *partial deduction of P wrt $\mathcal{A}$ (and ρ)* is obtained by generating a renaming of P' wrt $\mathcal{A}$ and ρ.

Example 3. Let P be the following program

(1) $t \leftarrow member(a, [a]), member(a, [a, b])$
(2) $member(X, [X|T]) \leftarrow$
(3) $member(X, [Y|T]) \leftarrow member(X, T)$

Let U be an unfolding rule such that: $\tau = chtree(\leftarrow member(a, [a]), P, U) = chtree(\leftarrow member(a, [a, b]), P, U) = \{((1, 2))\}$ and $\tau' = chtree(\leftarrow t, P, U) = \{((1, 1))\}$. Let $\mathcal{A} = \{(member(a, [a|T]), \tau), (t, \tau')\}$, $\rho((member(a, [a|T]), \tau)) = m_1$ and $\rho((t, \tau')) = t$. Then a partial deduction of P wrt $\mathcal{A}$ and ρ is:

(1') $t \leftarrow m_1([]), m_1([b])$
(2') $m_1(X) \leftarrow$

We will now adapt the standard correctness result for partial deduction of ordinary atoms for partial deduction of characteristic atoms.

Definition 13. Let P be a program and $\mathcal{A}$ a set of P-characteristic atoms. Then $\mathcal{A}$ is called *covered* iff for every characteristic atom in $\mathcal{A}$ each of its body atoms (in P) is a concretisation of a characteristic atom in $\mathcal{A}$.

Theorem 14. *Let U be an unfolding rule. Let P be a normal program, $\mathcal{A}$ a set of characteristic atoms and P' a partial deduction of P wrt $\mathcal{A}$ and some ρ. Let G be a goal and G' be a renaming of G wrt $\mathcal{A}$ and ρ.*

If $\mathcal{A}$ is covered and if the atoms of G are all concretisations of at least one characteristic atom in $\mathcal{A}$ then the following hold:

1. *$P' \cup \{\leftarrow G'\}$ has an SLDNF-refutation with computed answer θ iff $P \cup \{\leftarrow G\}$ does.*
2. *$P' \cup \{\leftarrow G'\}$ has a finitely failed SLDNF-tree iff $P \cup \{\leftarrow G\}$ does.*

PROOF SKETCH. In the definite case, each set of resultants for $(A, \tau) \in \mathcal{A}$ is a subset of a set of standard resultants (as defined in [17]) that can be obtained for the (standard) atom A. Because the partial deduction P' is covered, the atoms in the bodies of these resultants are concretisations of at least one characteristic atom in $\mathcal{A}$. This means that by treating characteristic atoms (A, τ) just as a normal atoms A, but just unfolding along τ and keeping the same renaming as performed by ρ we almost get a covered and independent standard partial deduction (with renaming) as defined in [1]. The only thing that is missing are the resultants R that were pruned by imposing τ on A, plus maybe some selected negative literals N which are in theory not selectable for A because they are not ground. If we now add these resultants R and incorporate the negative literals N we will obtain a standard partial deduction with renaming as defined in [1], which is however not necessarily covered because new atoms may appear in N and the bodies of R. All we have to do is add renamed versions of the original predicate definitions and map the uncovered atoms to these new predicates by extending the renaming. We have now obtained a covered and independent partial deduction with

renaming and a resulting specialised program P'' such that every derivation of P' can be mapped to a derivation of P'' (but not vice versa). We can also prove (given the conditions of the theorem concerning concretisations and coveredness) that all calls to N will be ground and succeed. Therefore soundness of the calculated answers of P' ($1.\Rightarrow$) and preservation of finite failure ($2.\Leftarrow$) follows from the standard correctness of partial deduction with renaming for P'' (see [1]). Similarly, P' can only be incomplete wrt the calculated answers ($1.\Leftarrow$) or finitely fail whereas the original program did not ($2.\Rightarrow$) if some derivation in P'' can be constructed which uses at least one of the resultants R added above. However this can only happen if some atom in G or in the leaves of the partial deductions of some $(A, \tau) \in \mathcal{A}$ in P is not a concretisation of a characteristic atom in $\mathcal{A}$, which is impossible given the conditions of the theorem. □

4 An Algorithm for Partial Deduction

We now define a possible partial deduction algorithm using concepts introduced in the previous section. As in [4, 5] we first define an abstraction operator. One can notice that, by definition, the abstraction operator preserves the characteristic trees.

Definition 15. Let $\mathcal{A}$ be a set of characteristic atoms. Also let, for every characteristic tree τ, $\mathcal{A}_\tau$ be defined as $\mathcal{A}_\tau = \{A \mid (A, \tau) \in \mathcal{A}\}$. The operation *abstract* is defined as: $abstract(\mathcal{A}) = \cup_\tau \{(msg(\mathcal{A}_\tau), \tau)\}$.

In other words only one characteristic atom per characteristic tree is allowed. For example in the case that $\mathcal{A} = \{(p(f(X)), \{((1,1))\}), (p(b), \{((1,1))\})\}$ we obtain $abstract(\mathcal{A}) = \{(p(X), \{((1,1))\})\}$.

Definition 16. Let A be an ordinary atom, U an unfolding rule and P a program. Then $chatom(A, P, U) = (A, \tau)$ where $chtree(\leftarrow A, P, U) = \tau$. We extend *chatom* to sets of atoms: $chatoms(\mathcal{A}, P, U) = \{chatom(A, P, U) \mid A \in \mathcal{A}\}$.

Note that A is a precise concretisation of $chatom(A, P, U)$.

The following is a formal description of an algorithm for partial deduction with characteristic atoms (more refined algorithms are possible). Please note that it is parameterised by an unfolding rule U, thus leaving the particulars of local control unspecified.

Algorithm 4.1 *Algorithm for Partial Deduction*

Input
a program P and set of atoms $\mathcal{A}$ to be specialised
Output
a specialised program P'
Initialisation
$k := 0$ and $\mathcal{A}_0 := chatoms(\mathcal{A}, P, U)$
Repeat
$\mathcal{A}_{k+1} := abstract(\mathcal{A}_k \cup chatoms(\{BA_P(A, \tau_A) \mid (A, \tau_A) \in \mathcal{A}_k\}, P, U))$
Until $\mathcal{A}_k = \mathcal{A}_{k+1}$ (modulo variable renaming)
$P' :=$ a partial deduction of P wrt $\mathcal{A}_k$ and some ρ

Theorem 17. *If the number of distinct characteristic trees is finite then Algorithm 4.1 terminates and generates a partial deduction satisfying the requirements of Theorem 14 for queries G whose atoms are instances of atoms in $\mathcal{A}$.*

PROOF. Termination is a direct corollary of Proposition 20 in Appendix A. Reaching the fixpoint guarantees that all predicates in the bodies of resultants are precise concretisations of at least one characteristic atom in $\mathcal{A}_k$, i.e. we always obtain a covered partial deduction. Furthermore *abstract* always generates more general characteristic atoms (even in the sense that any precise concretisation of an atom in $\mathcal{A}_i$ is a precise concretisation of an atom in $\mathcal{A}_{i+1}$ — this follows immediately from Definitions 6 and 15). Hence, because any instance of an atom in $\mathcal{A}$ is a precise concretisation of a characteristic atom in $\mathcal{A}_0$, the conditions of Theorem 14 are satisfied. □

Example 4. We are now in a position to treat Example 2. Let $A = member(a, [a])$, $B = member(a, [a, b])$ and $\tau = \{((1, 1))\}$. For $\mathcal{A} = \{A, B\}$ the algorithm yields:

1. $\mathcal{A}_0 = \{(A, \tau), (B, \tau)\}$
2. $BA_P(A, \tau) = BA_P(B, \tau) = \emptyset$, $\mathcal{A}_1 = abstract(\mathcal{A}_0) = \{(member(a, [a|T]), \tau)\}$
3. $BA_P(member(a, [a|T]), \tau) = \emptyset$, $\mathcal{A}_2 = abstract(\mathcal{A}_1) = \mathcal{A}_1$ and we have reached the fixpoint.

A partial deduction P' wrt $\mathcal{A}_1$ and ρ with $\rho((member(a, [a|T]), \tau)) = m_1$ is:

$m_1(X) \leftarrow$

$\mathcal{A}_1$ is covered and all the atoms in $G = \leftarrow member(a, [a]), member(a, [a, b])$ are instances of an atom in $\mathcal{A}$. As predicted by Theorem 17 all atoms in G are also concretisations of characteristic atoms in $\mathcal{A}_2$ and hence Theorem 14 can be applied: we obtain the renamed goal $G' = \leftarrow m_1([]), m_1([b])$ and $P' \cup \{G'\}$ yields the correct result.

A system incorporating Algorithm 4.1 has been implemented. Incorporating existing unfolding rules into the system was very straightforward — something we have already hinted at earlier. The system has been used for some successful applications, like in [15] where integrity checking was pre-compiled. In addition to the examples of this paper the system is precise enough to solve all the problematic examples in [12, 13]. On some occasions a depth-bound on characteristic trees was required.

The system has also been tested for the Lam and Kusalik benchmarks (originally in[9]) by Meulemans in [21] and the results are very satisfactory, as well for code size as for execution speed. For instance the results are better than those of the SP system ([4, 5]). Further testing (and refinement) is ongoing. Table 1 is extracted from [21] and compares the Algorithm 4.1 with standard partial deduction and where termination is guaranteed by using an abstraction operation which allows only one version (i.e. one *msg*) per predicate. The *Total* row contains the normalised total speedup of all the tests (each test was given the same weight). For both approaches the same simple determinate unfolding rule with a lookahead of one level and without selecting negative literals[8] has been

[8] For comparison's sake with other methods which are not able to handle negation, see [21].

used. As the speedups in Table 1 show (higher figures are better), even for this simple unfolding rule the new partial deduction method pays off. We conjecture that for more sophisticated unfolding rules and for more sophisticated programs the difference will be much more dramatic.

Benchmark	Speedup with Ecological PD	Speedup with 1 *msg* per predicate
advisor	1.27	1.28
ancestor	1.01	0.99
transpose	4.37	4.37
match	0.94	0.80
contains	1.13	1.00
depth	1.56	0.97
*gtrain*1	1.36	0.77
*gtrain*2	2.01	2.01
Total	1.37	1.13

Table 1. Speedup Figures

5 Discussion and Conclusion

For the moment the technique of this paper has to impose a depth bound on characteristic trees to ensure termination. In fact the number of characteristic trees is not always bounded in a natural way. This depth bound becomes necessary for example when an accumulating parameter "influences" the unfolding, meaning that the growth of the accumulator also leads to a corresponding growth of the characteristic tree. Some examples can be found in [14]. Fortunately [14] also proposes a solution to this problem by incorporating the partial deduction technique of this paper with ideas from [20]. The basic idea is to detect growing of characteristic trees and, if necessary, generalise characteristic atoms to ensure termination. Another promising approach to solve this problem might be based pre-computing a finite and "sufficient" set of potential characteristic trees in an off-line manner.

Also note that some built-in's (like `=../2`, `is/2`) can be easily incorporated into the characteristic trees. The concretisation definition for characteristic atoms scales up and the technique will ensure correct specialisation. It should also be possible to incorporate the `if-then-else` into characteristic trees (and then use a specialisation technique similar to [11]).

Characteristic trees and the method proposed in this paper can probably also be useful in other areas, such as constraint logic programming or supercompilation. For further remarks see [13, 14].

The simplicity and universality of the new method comes at a (relatively small) price compared to [12, 13]. The characteristic tree τ inside a characteristic

atom (A, τ) can be seen as an implicit representation of constraints on A. However in this paper these constraints are used only locally and are not propagated towards other characteristic atoms. The constraints in [12, 13] are propagated and thus used globally. Whether this has any real influence in practice remains to be seen.

In conclusion, we have presented a new framework and a new algorithm for partial deduction. The framework and the algorithm can handle *normal* logic programs and place *no* restrictions on the unfolding rule. The abstraction operator of the algorithm preserves the characteristic trees of the atoms to be specialised and ensures termination (when the number of distinct characteristic trees is bounded) while providing a fine grained control of polyvariance. All this is achieved without using constraints in the partial deduction process.

Acknowledgements

Michael Leuschel is supported by Esprit BR-project Compulog II. I would like to thank Danny De Schreye and Bern Martens for proof-reading (several versions) of this paper, for sharing their expertise and for all the encouragement and stimulating discussions. I would also like to thank Maurice Bruynooghe and John Gallagher for interesting comments and for pointing out several improvements. Finally I thank Dominique Meulemans for testing my system and anonymous referees for their helpful remarks.

References

1. K. Benkerimi and P. M. Hill. Supporting transformations for the partial evaluation of logic programs. *Journal of Logic and Computation*, 3(5):469–486, October 1993.
2. K. Benkerimi and J. W. Lloyd. A partial evaluation procedure for logic programs. In S. Debray and M. Hermenegildo, editors, *Proceedings of the North American Conference on Logic Programming*, pages 343–358. MIT Press, 1990.
3. 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.
4. J. Gallagher. A system for specialising logic programs. Technical Report TR-91-32, University of Bristol, November 1991.
5. 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.
6. 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.
7. J. Gallagher and M. Bruynooghe. The derivation of an algorithm for program specialisation. *New Generation Computing*, 9(3 & 4):305–333, 1991.
8. J. Komorowski. An introduction to partial deduction. In A. Pettorossi, editor, *Proceedings Meta'92*, pages 49–69. Springer-Verlag, LNCS 649, 1992.

9. J. Lam and A. Kusalik. A comparative analysis of partial deductors for pure Prolog. Technical report, Department of Computational Science, University of Saskatchewan, Canada, May 1990. Revised April 1991.
10. J.-L. Lassez, M. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan-Kaufmann, 1988.
11. M. Leuschel. Partial evaluation of the "real thing". In L. Fribourg and F. Turini, editors, Logic Program Synthesis and Transformation — Meta-Programming in Logic. *Proceedings of LOPSTR'94 and META'94*, Lecture Notes in Computer Science 883, pages 122–137, Pisa, Italy, June 1994. Springer-Verlag.
12. M. Leuschel and D. De Schreye. An almost perfect abstraction operator for partial deduction. Technical Report CW 199, Departement Computerwetenschappen, K.U. Leuven, Belgium, December 1994.
13. M. Leuschel and D. De Schreye. An almost perfect abstraction operation for partial deduction using characteristic trees. Technical Report CW 215, Departement Computerwetenschappen, K.U. Leuven, Belgium, October 1995. Submitted for Publication. Accessible via `http://www.cs.kuleuven.ac.be/lpai`.
14. M. Leuschel and B. Martens. Global control for partial deduction through characteristic atoms and global trees. Technical Report CW 220, Departement Computerwetenschappen, K.U. Leuven, Belgium, December 1995. Submitted for Publication. Accessible via `http://www.cs.kuleuven.ac.be/lpai`.
15. M. Leuschel and B. Martens. Partial deduction of the ground representation and its application to integrity checking. In J. Lloyd, editor, *Proceedings of ILPS'95, the International Logic Programming Symposium*, pages 495–509, Portland, USA, December 1995. MIT Press. Extended version as Technical Report CW 210, K.U. Leuven. Accessible via `http://www.cs.kuleuven.ac.be/lpai`.
16. J. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 1987.
17. J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *The Journal of Logic Programming*, 11:217–242, 1991.
18. B. Martens. *On the Semantics of Meta-Programming and the Control of Partial Deduction in Logic Programming*. PhD thesis, K.U. Leuven, February 1994.
19. 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.
20. 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.
21. D. Meulemans. Partiële deductie: Een substantiële vergelijkende studie. Master's thesis, K.U. Leuven, 1995.
22. A. Pettorossi and M. Proietti. Transformation of logic programs: Foundations and techniques. *The Journal of Logic Programming*, 19 & 20:261–320, May 1994.
23. M. Proietti and A. Pettorossi. The loop absorption and the generalization strategies for the development of logic programs and partial deduction. *The Journal of Logic Programming*, 16(1 & 2):123–162, May 1993.
24. D. Sahlin. Mixtus: An automatic partial evaluator for full Prolog. *New Generation Computing*, 12(1):7–51, 1993.

A Termination Property of *abstract*

First we introduce the following notation: for any finite set of characteristic atoms $\mathcal{A}$ and any characteristic tree τ, let $\mathcal{A}_\tau$ be defined as $\mathcal{A}_\tau = \{A \mid (A, \tau) \in \mathcal{A}\}$. The following well-founded measure function is taken from [6] (also in the extended version of [20]):

Definition 18. Let *Term*, *Atom* denote the sets of terms and atoms, respectively. We define the function $s : Term \cup Atom \rightarrow \mathbb{N}$ 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 a term or atom t be $v(t)$. We now define the function $h : Term \cup Atom \rightarrow \mathbb{N}$ by $h(t) = s(t) - v(t)$.

The well-founded measure function h has the property that $h(t) > 0$ for any non-variable t. Also if A is an atom strictly more general than B we have that $h(A) < h(B)$ (see [20]).

Definition 19. Let $\mathcal{A}$ be a set of characteristic atoms and let $T = \langle \tau_1, \ldots, \tau_n \rangle$ be a finite vector of characteristic trees. We then define the *weight vector* of $\mathcal{A}$ wrt T by $hvec_T(\mathcal{A}) = \langle w_1, \ldots, w_n \rangle$ where

- $w_i = \infty$ if $\mathcal{A}_{\tau_i} = \emptyset$
- $w_i = \sum_{A \in \mathcal{A}_{\tau_i}} h(A)$ if $\mathcal{A}_{\tau_i} \neq \emptyset$

Weight vectors are partially ordered by the usual order relation among vectors: $\langle w_1, \ldots, w_n \rangle \leq \langle v_1, \ldots, v_n \rangle$ iff $w_1 \leq v_1, \ldots, w_n \leq v_n$ and $\vec{w} < \vec{v}$ iff $\vec{w} \leq \vec{v}$ and $\vec{v} \not\leq \vec{w}$. The set of weight vectors is well founded (no infinitely decreasing sequences exist) because the weights of the atoms are well founded.

Proposition 20. *Let P be a normal program, U an unfolding rule and let $T = \langle \tau_1, \ldots, \tau_n \rangle$ be a finite vector of characteristic trees. For every finite set of characteristic atoms $\mathcal{A}$ and $\mathcal{B}$, such that the characteristic trees of their elements are in T, we have that one of the following holds:*

- *$abstract(\mathcal{A} \cup \mathcal{B}) = \mathcal{A}$ (up to variable renaming) or*
- *$hvec_T(abstract(\mathcal{A} \cup \mathcal{B})) < hvec_T(\mathcal{A})$.*

PROOF. Let $hvec_T(\mathcal{A}) = \langle w_1, \ldots, w_n \rangle$ and let $hvec_T(abstract(\mathcal{A} \cup \mathcal{B})) = \langle v_1, \ldots, v_n \rangle$. Then for every $\tau_i \in T$ we have two cases:

- $\{msg(\mathcal{A}_{\tau_i} \cup \mathcal{B}_{\tau_i})\} = \mathcal{A}_{\tau_i}$ (up to variable renaming). In this case the abstraction operation performs no modification for τ_i and $v_i = w_i$.
- $\{msg(\mathcal{A}_{\tau_i} \cup \mathcal{B}_{\tau_i})\} = \{M\} \neq \mathcal{A}_{\tau_i}$ (up to variable renaming). In this case $(M, \tau_i) \in abstract(\mathcal{A} \cup \mathcal{B})$, $v_i = h(M)$ and there are three possibilities:
  - $\mathcal{A}_{\tau_i} = \emptyset$. In this case $v_i < w_i = \infty$.
  - $\mathcal{A}_{\tau_i} = \{A\}$ for some atom A. In this case M is strictly more general than A (by definition of *msg* because $M \neq A$ up to variable renaming) and hence $v_i < w_i$.
  - $\#(\mathcal{A}_{\tau_i}) > 1$. In this case M is more general (but not necessarily strictly more general) than any atom in $\mathcal{A}_{\tau_i}$ and $v_i < w_i$ because at least one atom is removed by the abstraction.

We have that $\forall i \in \{1, \ldots, n\} : v_i \leq w_i$ and either the abstraction operation performs no modification (and $\vec{v} = \vec{w}$) or the well-founded measure $hvec_T$ strictly decreases. □

This article was processed using the LaTeX macro package with LLNCS style