In: O. Danvy, R. Glück, P. Thiemann, editors, *Partial Evaluation, International Seminar, Dagstuhl Castle, Germany, February 12–16, 1996*, Lecture Notes in Computer Science 1110, Springer, 1996.

A Theory of Logic Program Specialization and Generalization for Dealing with Input Data Properties

Alberto Pettorossi 1 and Maurizio Proietti 2

1 Department of Informatics, Systems, and Production,
University of Roma Tor Vergata, 00133 Roma, Italy. adp@iasi.rm.cnr.it
2 IASI-CNR, Viale Manzoni 30, 00185 Roma, Italy. proietti@iasi.rm.cnr.it

Abstract. We address the problem of specializing logic programs w.r.t. the contexts where they are used. We assume that these contexts are specified by means of computable properties of the input data. We describe a general method by which, given a program P, we can derive a specialized program P^1 such that P and P^1 are equivalent w.r.t. every input data satisfying a given property. Our method extends the techniques for partial evaluation of logic programs based on Lloyd and Shepherdson's approach, where a context can only be specified by means of a finite set of bindings for the variables of the input goal. In contrast to most program specialization techniques based on partial evaluation, our method may achieve superlinear speedups, and it does so by using a novel generalization technique.

1 Introduction

Program specialization is a technique which can be used for deriving from a given program, a new and hopefully more efficient program, by exploiting the knowledge of the context in which the given program is used. This knowledge can be expressed as a property which is satisfied by the input data. Thus the specialization of a program P w.r.t. a set of input data satisfying a property φ, called *static property*, consists in the derivation of a new program P^1 such that

$$\begin{array}{l}\text{for each value of } x \text{ in the domain of the input data,}\\ \text{if} \quad \varphi(x) \quad \text{then} \quad P(x) \equiv P^1(x)\end{array} \qquad (1)$$

where $\equiv$ is a chosen equivalence relation w.r.t. a given semantics.

This concept of program specialization includes that of *partial evaluation* where one assumes that x is a pair (s, d) of a *static* variable s and a *dynamic* variable d, and $\varphi(x)$ is of the form: $s = v$, where v is a value in the domain of the input data. However, since most of the work on program specialization deals with the special case of partial evaluation, often these two concepts are identified (see, for instance, [7]).

In this paper we consider the problem of specializing logic programs without negation, that is, *definite* logic programs [9]. We assume that the semantics of a program P is its least Herbrand model $M(P)$, that is, $M(P) = \{A \mid A$ is a ground atom and $P \models A\}$. We also assume that the static property $\varphi(X)$ is specified by a predicate $q(X)$ defined by the logic program to be specialized.

Thus, in our case, program specialization can be reformulated as follows: given a logic program P, a goal $q(X)$, called the *static* goal, and an input goal $p(X)$, called the *dynamic* goal, we want to derive a new program P^1, called *specialized* (or *residual*) program, and a new input goal $p^1(X)$ such that

$$\begin{array}{l}\text{for every ground substitution } \theta \text{ for } X,\\ \text{if} \quad M(P) \models q(X)\theta \ \text{ then } \ \big(M(P) \models p(X)\theta \ \text{ iff } \ M(P^1) \models p^1(X)\theta\big)\end{array} \qquad (2)$$

If this condition holds we say that *program P^1 with input goal $p^1(X)$ is a specialization of program P with input goal $p(X)$ w.r.t. the static goal $q(X)$*, and we also write $\langle P^1, p^1(X)\rangle$ *is a specialization of* $\langle P, p(X)\rangle$ *w.r.t.* $q(X)$.

The partial evaluation technique for logic programs as presented by Lloyd and Shepherdson [10], may be viewed as performing program specialization w.r.t. a static goal of the form $X = t$ where t is a possibly non-ground term.

Some extensions of partial evaluation, both of functional and logic programs, have been proposed to deal with static properties of input data, instead of static values. These extensions incorporate complex forms of reasoning based on abstract interpretation [3, 4] and theorem proving [1, 5].

By contrast, we propose a new technique which is based on the unfolding rule and a very limited form of goal replacement. On the one hand, this technique allows us to specify static properties in a precise way (unlike abstract interpretation) and, on the other hand, we avoid the use of very general theorem provers which may require complex heuristics for their effective use.

We now briefly describe our extension of Lloyd and Shepherdson's technique. Assume that the programs P and P^1 do not share any predicate symbol (this hypothesis will be relaxed in the sequel) and let Q be the subset of clauses in P on which the evaluation of $q(X)$ may depend. We observe that Condition (2) is equivalent to the following condition:

$$\begin{array}{l}\text{for every ground substitution } \theta \text{ for } X,\\ M(P) \models \big(\{q(X)\}\ p(X)\big)\theta \quad \text{iff} \quad M(Q \cup P^1) \models \big(\{q(X)\}\ p^1(X)\big)\theta\end{array} \qquad (3)$$

where $\{S\}\,D$ is an alternative notation (borrowed from [1]) for the conjunction (S, D) which explicitly indicates that S is a property that holds when the evaluation of D is performed. If Condition (3) holds we say that $\langle P^1, p^1(X)\rangle$ *is a specialization of P w.r.t. the static part of the goal* $\{q(X)\}\ p(X)$. Condition (3) is equivalent to

$$M(P \cup P^1) \models \ \ \forall X \left(\{q(X)\}\ p(X) \ \leftrightarrow \ \{q(X)\}\ p^1(X)\right) \qquad (4)$$

which makes it clear that the correctness of program specialization is proved by showing the equivalence of two goals.

Notice that after program specialization, the specialized predicate $p^1(X)$ defined by the specialized program P^1, is equivalent to $p(X)$ when $q(X)$ holds, but it need not be equivalent to the conjunction $\{q(X)\}\,p(X)$.

We will provide a proof method which allows us to conclude that Condition (4) holds by constructing, via unfolding and goal replacement steps, two finite trees T and T^1 such that: i) goal $\{q(X)\}\ p(X)$ is at the root of T, ii) goal $\{q(X)\}\ p^1(X)$ is at the root of T^1, iii) T and T^1 are *closed* trees (see Section 4), and iv) T and T^1 are *isomorphic* (see Section 5).

This proof method justifies our specialization technique described in Section 4, which, in fact, from a closed tree T with goal $\{q(X)\}\ p(X)$ at its root, derives the clauses for a new atom $p^1(X)$ such that there exists a closed tree T^1 with goal $\{q(X)\}\ p^1(X)$ at its root, such that T and T^1 are isomorphic.

It is straightforward to extend our techniques to the general case where: i) the predicates p, p^1, and q have any arity, ii) the static goal is any conjunction of atoms, instead of $q(X)$ only, and iii) the static and dynamic goals share any number of variables. Indeed, the formal presentation of our technique in Section 4 and its correctness proof in Section 5, are done for this general case.

In Section 6 we present a generalization technique which allows us to derive very efficient specialized programs. In particular, we give some examples where program specialization achieves superlinear speedups. Finally, in Section 7 we briefly compare our techniques with related specialization techniques both for logic and functional programs.

2 Preliminary Notions

In this section we introduce some definitions which are used in the sequel. By $hd(C)$ and $bd(C)$ we denote the head and the body, respectively, of a clause C. Goals are conjunctions of zero or more atoms. The empty goal is denoted by $\Box$ and a clause with head A and empty body is written as $A \leftarrow$. The conjunction operator denoted by "," is commutative, and thus the order of the atoms in a goal is not significant. By contrast, when performing specializations we use a specific rule to eliminate duplicate occurrences of an atom in a goal. This is motivated by the fact that two distinct occurrences of the same atom may have different static/dynamic classifications (see Section 4). By $vars(G)$ we denote the set of variables occurring in a goal G.

Given a predicate p occurring in a definite program P, the *definition* of p in P is the set of all clauses in P whose head predicate is p. We say that a predicate p *depends on* a predicate q in a program P iff *either* there exists in P a clause of the form $p(\ldots) \leftarrow B$ such that q occurs in the goal B *or* there exists in P a predicate r such that p depends on r in P and r depends on q in P. The *relevant part* of program P for predicate p, denoted by $Rel(P,p)$, is the union of the following definitions in P: i) the definition of p and (ii) the definitions of all the predicates on which p depends in P.

We now give two rules which are used to derive new goals from old goals during program specialization.

Definition 1 (Unfolding Rule). Let C be a clause and G be the goal (H, A, K), where H and K are (possibly empty) goals and A is an atom. Suppose that A and $hd(C)$ are unifiable via the most general unifier θ. By *unfolding* G w.r.t. A using C we derive the goal $(H, bd(C), K)\theta$.

Thus, an application of the unfolding rule essentially is an application of the SLD-resolution rule.

Definition 2 (Goal Replacement Rule). Let P be a program and G be a goal of the form (H, U, K), where H, U, and K are (possibly empty) goals. Let V be a goal such that the following equivalence holds:

$$M(P) \models \forall X_1, \ldots, X_n \, (U \leftrightarrow V) \tag{5}$$

where $X_1, \ldots, X_n$ are the variables occurring in $U \leftrightarrow V$. By *goal replacement* (or simply, *replacement*) of U by V in G we derive the goal (H, V, K).

The process of deriving new goals from old ones by applying the unfolding and the goal replacement rules, is represented as a tree of goals as follows.

Definition 3 (Unfolding-Replacement Tree and Forest). Given a program P and a goal G, an *unfolding-replacement tree* (or *UR-tree*) for $\langle P, G \rangle$ is a non-empty, finite, directed tree with nodes labeled by goals and arcs labeled by substitutions such that:

- the goal labeling the root, also called root-goal, is G,
- if M is a node labeled by a goal G_M then exactly one of the following three cases occurs:
  1. M has no sons, and G_M is called a leaf-goal,
  2. G_M is of the form (H, A, K) and A is unifiable with the heads of the (standardized apart) clauses $C_1, \ldots, C_n$, with $n \geq 1$, in P via the most general unifiers $\theta_1, \ldots, \theta_n$, respectively. In this case M has n sons, say $N_1, \ldots, N_n$, and for $i = 1, \ldots, n$, the node N_i is labeled by the goal derived by unfolding G_M w.r.t. A using C_i and the arc from M to N_i is labeled by θ_i,
  3. G_M is of the form (H, U, K) and Equivalence (5) holds. In this case M has exactly one son N labeled by the goal (H, V, K) derived by the replacement of U by V in G_M, and the arc from M to N is labeled by the identity substitution which we denote by ε.

A node N is said to be a *failing node* iff it is labeled by a goal G_N, called a *failing goal*, such that $M(P) \models \forall X_1, \ldots, X_n \, (\neg G_N)$, where $X_1, \ldots, X_n$ are the variables occurring in G_N. In particular, if G_N is of the form (H, A, K) and A is an atom which is not unifiable with the head of any clause in P, then N is a failing node. A path from a node to a leaf is called a *branch*, and a branch ending with a failing leaf-node is called a *failing branch*.

A finite, non-empty set of UR-trees is called a *UR-forest*.

The reader will notice that the unfolding-replacement trees introduced here are similar to the SLD-trees considered in [10], where, however, no goal replacements are allowed.

3 An Introductory Example

This example was given in [8] to show that the usual partial evaluation techniques as described by Lloyd and Shepherdson [10], are not adequate. Let us consider the following logic program, call it *Append-Last*:

1. $append([\,], Ys, Ys) \leftarrow$
2. $append([X|Xs], Ys, [X|Zs]) \leftarrow append(Xs, Ys, Zs)$
3. $last([X], X) \leftarrow$
4. $last([X|Xs], Y) \leftarrow last(Xs, Y)$.

Let us suppose that we want to specialize *Append-Last* w.r.t. the static part of the goal:

$$\{append(Xs, [a], Ys)\}\ last(Ys, Z) \tag{6}$$

that is, we look for the specialization of *Append-Last* with input goal $last(Ys, Z)$ w.r.t. the static goal $append(Xs, [a], Ys)$, which indeed expresses the fact that the last element of the list Ys is a. Partial evaluation techniques like those in [10] are not applicable in this case, because there is no input goal of the form $last(t, Z)$ such that for some choice of the term t the set of all instances of t is the set of all lists whose last element is a.

Let us now consider the UR-tree T depicted in Figure 1 with root-goal $\{append(Xs, [a], Ys)\}\ last(Ys, Z)$. In this figure and in the subsequent ones the static goals are written between curly brackets, unfolded atoms are underlined, and an upgoing arrow from goal H to goal K indicates that H is an instance of K, but that arrow itself is *not* an arc of the UR-tree.

Notice that in the non-failing leaves of T the only occurrence of the predicate *last* is in the goal $\{append(Xs1, [a], Ys1)\}\ last(Ys1, Z)$ which is an instance (actually, a variant) of the root-goal. In Section 4 this property of a UR-tree is generalized to the notion of *closed specialization tree.*

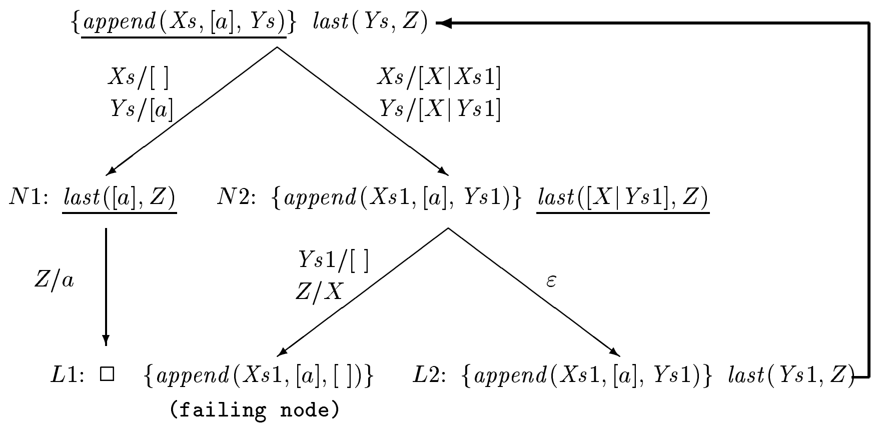

$\{\underline{append(Xs, [a], Ys)}\}\ last(Ys, Z)$
$Xs/[\,]$
$Ys/[a]$
$Xs/[X|Xs1]$
$Ys/[X|Ys1]$
$N1$: $\underline{last([a], Z)}$
$N2$: $\{append(Xs1, [a], Ys1)\}\ \underline{last([X|Ys1], Z)}$
Z/a
$Ys1/[\,]$
Z/X
ε
$L1$: $\square$
$\{append(Xs1, [a], [\,])\}$
(failing node)
$L2$: $\{append(Xs1, [a], Ys1)\}\ last(Ys1, Z)$

Fig. 1. The UR-tree T with root-goal $\{append(Xs, [a], Ys)\}\ last(Ys, Z)$.

Now we look for a new definition of the predicate *last*, by which we can construct, starting from a root-goal equal to the one of T, a UR-tree T^1 which has the same set of $\langle G, \theta \rangle$ pairs, where G is a non failing leaf-goal and θ is the composition of the substitutions along the branch from the root to G, restricted to the variables occurring in the root-goal.

In order to find such new clauses for *last* we apply the following technique which is a generalization of that considered in [10] for computing the *resultants* of a given SLD-tree.

Let us first consider in the tree T the non-failing branches together with their substitutions, which have been produced by performing an unfolding step w.r.t. an atom with predicate *last*. They are: i) branch $b1$ from node $N1$ to leaf $L1$, and ii) branch $b2$ from node $N2$ to leaf $L2$, that is:

$b1.$ $\mathit{last}([a], Z) \xrightarrow{\{Z/a\}} \square$

$b2.$ $\{\mathit{append}(Xs1, [a], Ys1)\}\ \mathit{last}([X \mid Ys1], Z) \xrightarrow{\varepsilon} \{\mathit{append}(Xs1, [a], Ys1)\}$
$\mathit{last}(Ys1, Z).$

Given these branches we consider the corresponding resultants

$B1.$ $\mathit{last}([a], a) \leftarrow$
$B2.$ $\mathit{last}([X \mid Ys1], Z) \leftarrow \mathit{last}(Ys1, Z).$

(In Definition 7 we introduce the formal notion of resultant we use in this paper.)

We have that the leaf-goals and the substitutions associated with the two branches $b1$ and $b2$, can be derived by using clauses $B1$ and $B2$, instead of clauses 3 and 4. In this situation we say that clauses $B1$ and $B2$ *validate* branches $b1$ and $b2$ (see Definition 14 for this concept).

Notice that branch $b1$ is also validated by either one of the following two clauses, which are *generalizations* of clause $B1$:

5. $\mathit{last}([Z], Z) \leftarrow$
6. $\mathit{last}(Ys, a) \leftarrow$

because, by unfolding $\mathit{last}([a], Z)$ using either clause 5 or clause 6 we get the empty leaf-goal and the substitution $\{Z/a\}$ associated with branch $b1$.

Furthermore, by using clause 6 one may validate not only branch $b1$ but also branch $b2$. In fact, the leaf-goal of $b2$ can be derived by a goal replacement step, if we assume that clause 6 is the only clause defining the predicate *last*. Condition (5) for goal replacement is fulfilled because by unfolding $\{\mathit{append}(Xs1, [a], Ys1)\}\ \mathit{last}([X \mid Ys1], Z)$ w.r.t. $\mathit{last}([X \mid Ys1], Z)$ using clause 6 we get the result we also get by unfolding $\{\mathit{append}(Xs1, [a], Ys1)\}\ \mathit{last}(Ys1, Z)$ w.r.t. $\mathit{last}(Ys1, Z)$ using again clause 6, and the unfolding rule preserves equivalence w.r.t. the least Herbrand model semantics.

As a consequence of what we have said so far, by using the program made out of clauses 1, 2, and 6, we may construct a UR-tree T^1 (see Figure 2) which has the same non-failing branches of T and the same associated substitutions. Branch $b1$ from node $N1$ to node $L1$ of T^1 is generated by an unfolding step, whereas branch $b2$ from node $N2$ to node $L2$ is generated by a goal replacement step justified by the fact that $\{append(Xs1,[a],Ys1)\}\ last([X|Ys1],Z)$ and $\{append(Xs1,[a],Ys1)\}\ last(Ys1,Z)$ are equivalent in the least Herbrand model of clauses 1, 2, and 6.

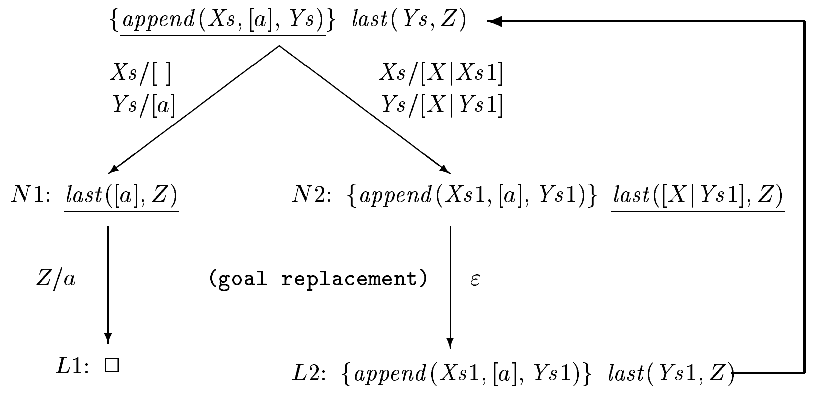

$\{append(Xs,[a],Ys)\}\ last(Ys,Z)$
$Xs/[\,]$
$Ys/[a]$
$Xs/[X|Xs1]$
$Ys/[X|Ys1]$
$N1$: $last([a],Z)$
$N2$: $\{append(Xs1,[a],Ys1)\}\ last([X|Ys1],Z)$
Z/a
(goal replacement)
ε
$L1$: $\square$
$L2$: $\{append(Xs1,[a],Ys1)\}\ last(Ys1,Z)$

Fig. 2. The UR-tree T^1 with root-goal $\{append(Xs,[a],Ys)\}\ last(Ys,Z)$.

As indicated later in Sections 5 and 6 (see Theorems 12 and 15), the existence of tree T^1 ensures that the following property holds for all ground terms x, y, and z:

$$\begin{array}{ll} \text{if} & M(\textit{Append-Last}) \models append(x,[a],y) \\ \text{then} & M(\textit{Append-Last}) \models last(y,z) \ \text{ iff } \ M(\{\text{clause } 6\}) \models last(y,z). \end{array} \quad (7)$$

Thus, according to Condition (2), $\langle\{\text{clause } 6\},\ last(Ys,Z)\rangle$ is a specialization of *Append-Last* w.r.t. the static part of $\{append(Xs,[a],Ys)\}\ last(Ys,Z)$.

The reader should notice that, as already mentioned in the Introduction, the goal $\{append(Xs,[a],Ys)\}\ last(Ys,Z)$, where $last(Ys,Z)$ is defined by clauses 3 and 4, is *not* equivalent to the specialized predicate $last(Ys,Z)$ defined by clause 6. Indeed, there are ground terms x, y, and z such that

$$M(\textit{Append-Last}) \models \{append(x,[a],y)\}\ last(y,z) \ \text{ iff } \\ M(\{\text{clause } 6\}) \models last(y,z)$$

does *not* hold. However, Condition (7) shows that if $append(Xs,[a],Ys)$ holds then the two definitions of $last(Ys,Z)$ are equivalent.

In the next sections we present a general theory of program specialization which justifies the various steps performed during the development of this *Append-Last* example. In the general case where the specialized program is derived from more than one tree, we will see that one also needs to perform some predicate renaming steps for avoiding undesired interactions among specialized clauses.

4 Deriving Specialized Programs from Unfolding Trees

In this section we describe our technique for deriving specialized programs from UR-trees. Actually, we need to construct UR-trees of a particular kind, called *specialization trees*, which we introduce in the sequel.

We first need some preliminary notions. We assume that the predicates occurring in a program P are partitioned into two disjoint sets: the *basic predicates* and the *defined predicates*. We require that basic predicates do not depend on defined predicates. Also the atoms that we consider during program specialization are assumed to be either *basic atoms* or *defined atoms*, depending on whether their predicates are basic or defined.

The distinction between basic and defined predicates is introduced for ensuring the correctness of our specialization technique. Indeed, it is well known (see, for instance, [13]), that the unrestricted use of the goal replacement rule may lead to incorrect programs. However, goal replacement steps which replace basic atoms by new basic atoms are always correct (see Section 5). A more general way of stratifying program predicates to ensure the correctness of program transformations based on unfolding, folding, and goal replacement, is proposed in [15].

We have seen in our introductory example, that the atoms of a given goal may be classified as either static or dynamic. This classification of the atoms is taken into account for the extraction of the specialized program from a set of specialization trees (see Definition 8). Indeed, each clause of the specialized program is obtained from the dynamic atoms of a non-failing branch of a specialization tree. We assume that *all* atoms occurring in a UR-tree are classified as either static or dynamic (thereby determining a partition of those atoms), and in order to derive that classification for every atom, we will give rules which are based on the classification of the atoms in the root-goal and on the way in which the UR-tree is constructed.

The goals made out of static atoms only are called *static goals* and the goals made out of dynamic atoms only are called *dynamic goals*. A goal which is written as $\{S\}\,D$, is the conjunction of a static goal S and a dynamic goal D.

Based on the static/dynamic classification of goals, we can also partition the non-leaf nodes of a UR-tree as indicated in the following definition.

Definition 4 (s-Nodes, d-Nodes, Branching Nodes). A non-leaf node N in a UR-tree is said to be an *s-node* iff either its k (≥ 1) sons are derived by unfolding w.r.t. a static atom, or its son is derived by a goal replacement step which replaces an old static goal by a new static goal. The definition of a *d-node* is obtained from that of an s-node by replacing every occurrence of 'static' by 'dynamic'.
A node (either an s-node or a d-node) is said to be a *branching* node iff its sons are more than one, that is, they are derived by unfolding a goal w.r.t. an atom which unifies with the head of more than one program clause.

Definition 5 (Specialization Tree). A *specialization tree* for a program P, a static goal S, and a dynamic defined atom D, is a UR-tree for $\langle P, \{S\}\,D\rangle$, where each atom is classified as either static or dynamic. We now list the restrictions on the use of the unfolding and replacement rules during the construction of a specialization tree for $\langle P, \{S\}\, D\rangle$, and we also indicate how to classify as either static or dynamic every atom in that tree.

1. If $(H\theta, bd(C)\theta, K\theta)$ is the goal derived by unfolding a goal of the form (H, A, K) w.r.t. A using clause C, then i) the atoms occurring in the goals $H\theta$ and $K\theta$ have the same static/dynamic classification of the corresponding atoms in H and K, and ii) the atoms occurring in $bd(C)\theta$, if any, have the same static/dynamic classification of A.
2. If a goal replacement is performed, whereby an old goal U is replaced by a new goal V, then at least one of the following four cases applies (where S_1 and D_1 denote goals and, as usual, the order of the atoms is not significant):
   i) U is of the form $\{X, S_1\}\, D_1$ and V is of the form $\{Y, S_1\}\, D_1$, where X and Y are static goals made out of basic atoms only;
   ii) U is of the form $\{S_1\}\, X, D_1$ and V is of the form $\{S_1\}\, Y, D_1$, where X and Y are dynamic goals made out of basic atoms only;
   iii) U is of the form $\{G\}\, G$ and V is of the form $\{G\}$ for some goal G (deletion of a duplicate dynamic goal);
   iv) U is of the form $\{G\}\, A$ and V is of the form $\{G\}\, A^1$, where A^1 is an atom derived in a previous process of program specialization w.r.t. the static goal G. More precisely, P is of the form $Q \cup Q^1$ and $\langle Q^1, A^1\rangle$ is a specialization of $\langle Q, A\rangle$ w.r.t. G (see Definition 8).
3. Every branch from the root to a non-failing leaf contains at least one goal derived by performing an unfolding step w.r.t. a dynamic defined atom.
4. Every ancestor of a branching s-node is an s-node.

Notice that, in all goal replacements which can take place during the construction of a specialization tree, we replace a goal U by a new goal which differs from U either in the static part only or in the dynamic part only. Thus, the static/dynamic classification of the atoms occurring in son-goals can be inferred in the obviuos way from the static/dynamic classifications of the father-nodes.

Notice also that case 2.iv) may indeed be viewed as a goal replacement step, because if $\langle Q^1, A^1\rangle$ is a specialization of $\langle Q, A\rangle$ w.r.t. the static goal G, then (by Property (4) or, more formally, by Property (10) of the following Theorem 12) we have that Condition (5) of the goal replacement rule is satisfied, that is, the following property holds:

$$M(Q \cup Q^1) \models \forall X_1, \ldots, X_n\, (\{G\}\, A \leftrightarrow \{G\}\, A^1) \tag{8}$$

where $X_1, \ldots, X_n$ are the variables occurring in (G, A, A^1).

Definition 6 (Closed Specialization Forest). A *specialization forest* for a program P, a static goal S, and a dynamic defined atom D, is a finite, non-empty set of specialization trees such that every tree is constructed by using program P and $\{S\}\, D$ is one of the root-goals.
A specialization forest is said to be *closed* iff for each leaf-goal L one of the following three cases applies: 1) L is a failing goal, or 2) L contains no occurrences

of defined dynamic atoms, or 3) for every dynamic defined atom A occurring in L, there exist a static, possibly empty (sub)goal G of L, a root-goal $\{H\}\, B$ of a tree in the forest, and a substitution σ such that $G = H\sigma$ and $A = B\sigma$.
A specialization tree T is said to be closed iff the specialization forest $\{T\}$ is closed.

Examples of closed specialization trees and forests may be found in Figures 1, 4, and 6. As already mentioned, in those figures an upgoing arrow from a leaf-goal L to a root-goal R, indicates that L contains a subgoal (possibly L itself) which is an instance of R.

Our notion of closed specialization tree is strongly related to the notion of closedness considered by Lloyd and Shepherdson [10], which, however, refers to resultants, and not to trees. Indeed, the reader may verify that a set of resultants of a set F of SLDNF-trees enjoys the closedness property defined in [10] iff for each leaf-goal L of a tree in F, one of the following three cases applies: 1) L is a failing goal, or 2) L is the empty goal, or 3) each atom occurring in L is an instance of an atom occurring in a root (not necessarily of the same tree) of F.

Notice that in Lloyd and Shepherdson's technique the root-goal of an SLDNF-tree can only be an atom, there is no distinction between static and dynamic goals, and there is no distinction between basic and defined atoms. Notice also that Definition 6 does not require that *all* static atoms occurring in a leaf-goal of a closed specialization forest are instances of static atoms occurring in the roots. Indeed, we do not want to derive new clauses to evaluate static goals. Likewise, we do not want to derive new clauses for the dynamic basic atoms, and this is why in Definition 6 (Case 3), in the leaves of a closed specialization forest we consider dynamic defined atoms only.

In the following definition we indicate how to derive from a specialization forest new, specialized clauses, called *resultants*, to evaluate dynamic defined atoms.

Definition 7 (Resultants). Given a specialization tree T, a d-node N of T is said to be *topmost* in T iff no ancestor of N is a d-node.
Let $\{S\}\, D$ be the label of a topmost d-node N in T (thus, D is a dynamic defined atom). Let us consider a branch from N to a *non-failing* leaf L, whose goal is $\{Ss\}\, Ds$, where Ds is a possibly empty, possibly non-atomic goal. Let θ be the composition of the substitutions along that branch. Then the clause $D\theta \leftarrow Ds$ is a *resultant* of that branch from N to L.
The set of the resultants of a specialization tree T is $\{r \mid r$ is a resultant of a branch from a topmost d-node of T to a non-failing leaf of $T\}$.

The following Definition 8 introduces the notion of program specialization based on specialization trees. Thus Definition 5 and Definition 8 depend on each other, because during the construction of a specialization tree one may consider the result of a previous program specialization (see case 2.iv in Definition 5).

A program specialization is derived from a specialization forest by first collecting all resultants of that forest, then renaming the predicates so that two resultants have the same head predicate iff they come from the same specialization tree, and finally adding the clauses defining the basic predicates occurring in the resultants.

Definition 8 (Program Specialization). Let P be a program, S_1 be a static goal, and D_1 be a dynamic defined atom. Let $\{T_i \mid i = 1, \ldots, n\}$ be a closed specialization forest for $\langle P, \{S_1\}\, D_1\rangle$ such that $\{S_1\}\, D_1$ is the root-goal of T_1. A *specialization* of $\langle P, D_1\rangle$ w.r.t. the static goal S_1 *via* the closed specialization forest $\{T_i \mid i = 1, \ldots, n\}$, is the pair $\langle P^1, D^1\rangle$ where P^1 is a program and D^1 is an atom which are obtained as follows:

1. Let R be the union of the sets R_i of the resultants of T_i, for $i = 1, \ldots, n$; rename *all* occurrences of the defined predicates in R, whereby deriving a new set $\overline{R}$ of clauses, as follows:
   1.1 for $i = 1, \ldots, n$, the predicate symbol in the head of every resultant of T_i is renamed by decorating it with the superscript i, and
   1.2 an atom A of the form $p(\ldots)$ in the body of the resultant of a branch with leaf-goal L is renamed to $p^i(\ldots)$ if there exist a static, possibly empty (sub)goal G of L and a substitution σ such that $G = S_i\sigma$ and $A = D_i\sigma$, where $\{S_i\}\, D_i$ is the root-goal of T_i.
2. P^1 is obtained as the set $\overline{R} \cup \mathit{Rel}(P, q_1) \cup \ldots \cup \mathit{Rel}(P, q_k)$ of clauses, where $q_1, \ldots, q_k$ are the basic predicates occurring in $\overline{R}$.
3. If D_1 is the atom $p(\ldots)$ then D^1 is the atom $p^1(\ldots)$ (thus, as a consequence of the predicate renaming performed at Point 1, D^1 is unifiable only with the heads of the renamed resultants derived from T_1).

Notice that, in general, for any given specialization forest more than one program specialization can be constructed (because Point 1.2 of Definition 8 allows for nondeterminism).

Our introductory example in Section 3 includes an example of program specialization according to Definition 8. In particular, the reader may notice that nodes $N1$ and $N2$ are the topmost d-nodes of the closed specialization tree for $\langle \mathit{Append\text{-}Last}, \{\mathit{append}(\mathit{Xs}, [a], \mathit{Ys})\}\, \mathit{last}(\mathit{Ys}, Z)\rangle$ depicted in Figure 1. In our example we have already shown how to compute the resultants of the branches from $N1$ and $N2$ to the non-failing leaves $L1$ and $L2$, respectively. The resultants of those branches are clauses $B1$ and $B2$. The predicate renaming indicated at Point 1 of Definition 8, in this case consists of replacing in clauses $B1$ and $B2$ the predicate symbol *last* by the new symbol last^1 because the specialization forest is made out of one tree only.

5 Correctness of Program Specialization

We first notice that the specialization process described in the previous section may sometimes give incorrect programs. We present in this section a result which ensures correctness if the specialization forest satisfies an extra condition which generalizes that of *independence of a set of atoms* considered in [10].

We now show an example where the program specialization defined in Section 4 gives an incorrect result. Let us consider the program P:

1. $p(X) \leftarrow X = 0$
2. $q(X) \leftarrow X = 0$
3. $q(X) \leftarrow X = 1$
4. $X = X \leftarrow$

In order to derive a specialization of $\langle P, p(X) \rangle$ w.r.t. the static goal $q(X)$ we first construct the closed specialization tree shown in Figure 3.

$$\begin{tikzcd}
 & \{\underline{q(X)}\}\; p(X) \arrow[dl, "\varepsilon"'] \arrow[dr, "\varepsilon"] & \\
N1\colon \{X=0\}\; \underline{p(X)} \arrow[d, "\varepsilon"] & & N2\colon \{\underline{X=1}\}\; p(X) \arrow[d, "X/1"] \\
\{X=0\}\; X=0 \arrow[d, "\varepsilon", "\texttt{(deletion of }X=0\texttt{)}"'] & & \underline{p(1)} \arrow[d, "\varepsilon"] \\
L1\colon \{X=0\} & & L2\colon 1=0 \\
 & & \texttt{(failing node)}
\end{tikzcd}$$

Fig. 3. A closed specialization tree with renamed resultant $p^1(X) \leftarrow$.

Then, according to Definition 8, from that tree we get the following program P^1 (due to the resultant of the non-failing branch from $N1$ to $L1$):

5. $p^1(X) \leftarrow$

We have that $M(P) \not\models \{q(1)\}\, p(1)$ whereas $M(\{\text{clause } 2, \text{clause } 3\} \cup P^1) \models \{q(1)\}\, p^1(1)$, and therefore, by Condition (3), $\langle P^1, p^1(X) \rangle$ is *not* a specialization of $\langle P, p(X) \rangle$ w.r.t. $q(X)$. We get this undesired result because the resultant '$p(X) \leftarrow$' of the branch from $N1$ to $L1$ where $X = 0$ holds, can also be used in the case where $X = 1$ holds. Indeed, $p(X)$ also occurs in the branch from $N2$ to $L2$ where $X = 1$ holds. Thus, unlike the case of partial evaluation based on Lloyd and Shepherdson's approach, the renaming of the predicates which occur in the resultants of a closed specialization forest, does not ensure the correctness of the specialization.

For avoiding problems of the kind we have now mentioned, it suffices that the dynamic atoms labeling the roots of the branches from which we compute the resultants (that is, the topmost d-nodes) do not have common instances. This remark motivates the introduction of the following notion of independent nodes.

Definition 9 (Independent Nodes). Let $N1$ and $N2$ be two nodes in a specialization tree labeled by two goals of the form $\{G\}\, A$ and $\{H\}\, B$, respectively. $N1$ and $N2$ are *independent* iff A and B have no common instances.

Our correctness result stated in the following Theorem 12, holds in the case where the topmost d-nodes in each tree of the specialization forest for $\langle P, \{S\}\, D\rangle$ are pairwise independent.

For the proof of Theorem 12 we also need to show that, if we construct, according to Definition 8, a specialized program P^1 with input goal D^1 from a program P with input goal D w.r.t. a static goal S, then (the universal closure of) the goal equivalence $\{S\}\, D \leftrightarrow \{S\}\, D^1$ holds in the least Herbrand model of $P \cup P^1$. This goal equivalence is proved by showing that $\langle P \cup P^1, \{S\}\, D\rangle$ and $\langle P \cup P^1, \{S\}\, D^1\rangle$ generate *isomorphic* closed specialization forests as indicated in the following Lemma 11. We now introduce the notion of isomorphism between forests which will also apply to specialization forests.

Definition 10 (Forest Isomorphism). Let us consider the two forests (with the same number of trees) $T = \{T_i \mid i = 1, \ldots, m\}$ and $U = \{U_i \mid i = 1, \ldots, m\}$ whose nodes are labeled by goals and whose arcs are labeled by substitutions. For $i = 1, \ldots, m$, let TR_i and UR_i be the root-goals of the trees T_i and U_i, respectively. We assume that for $i = 1, \ldots, m$, $vars(TR_i) = vars(UR_i)$. Let $\{TL_{ij} \mid j = 1, \ldots, n_i\}$ and $\{UL_{ij} \mid j = 1, \ldots, n_i\}$ be the multiset of non-failing leaf-goals of T_i and U_i, respectively. For $i = 1, \ldots, m$ and $j = 1, \ldots, n_i$, let θ_{ij} be the composition of all substitutions along the branch from the root of T_i to TL_{ij}, and similarly, let μ_{ij} be the composition of all substitutions along the branch from the root of U_i to the leaf labeled by UL_{ij}. We say that T is *isomorphic* to U iff for $i = 1, \ldots, m$ and $j = 1, \ldots, n_i$, we have that:

1. the leaf-goal UL_{ij} can be obtained from the leaf-goal TL_{ij} by simultaneously replacing in TL_{ij} zero or more occurrences of instances of root-goals of T by the corresponding instances of root-goals of U, that is, by simultaneously replacing in TL_{ij} the occurrences $TR_{k1}\sigma_{k1}, \ldots, TR_{ks}\sigma_{ks}$, for some substitutions $\sigma_{k1}, \ldots, \sigma_{ks}$ with $\{k1, \ldots, ks\} \subseteq \{1, \ldots, m\}$, by $UR_{k1}\sigma_{k1}, \ldots, UR_{ks}\sigma_{ks}$, respectively, and
2. $X\theta_{ij} = X\mu_{ij}$ for every X in $vars(TR_i)$.

Lemma 11. *Let us consider two specialization forests, say $T = \{T_i \mid i = 1, \ldots, m\}$ and $U = \{U_i \mid i = 1, \ldots, m\}$, constructed for the same program P and possibly distinct root-goals. For $i = 1, \ldots, m$, let TR_i and UR_i be the root-goals of T_i and U_i, respectively, with $vars(TR_i) = vars(UR_i)$. If T and U are isomorphic, then for $i = 1, \ldots, m$,*

$$M(P) \models \forall X_1, \ldots, X_k\, (TR_i \leftrightarrow UR_i)$$

where $\{X_1, \ldots, X_k\} = vars(TR_i)$.

Theorem 12 (Correctness of Specialization). *Let P be a program, S_1 be a static goal, and D_1 be a dynamic defined atom. Let $\langle P^1, D^1\rangle$ be a specialization* *of* $\langle P, D_1 \rangle$ *w.r.t.* S_1 *via the* closed *specialization forest* $\{T_i \mid i = 1, \ldots, n\}$ *such that the root of* T_1 *is* $\{S_1\}\, D_1$*. Suppose that vars*$(\{S_1\}\, D_1) = \{X_1, \ldots, X_k\}$*, and for* $i = 1, \ldots, n$*, any two distinct topmost d-nodes of* T_i *are* independent*. Then, we have that:*

$$\begin{array}{l} \textit{for each ground substitution } \theta \textit{ for } X_1, \ldots, X_k, \\ \textit{if} \quad M(P) \models S_1\theta \quad \textit{then} \quad M(P) \models D_1\theta \;\; \textit{iff} \;\; M(P^1) \models D^1\theta \end{array} \tag{9}$$

or, equivalently,

$$M(P \cup P^1) \models \forall X_1, \ldots, X_k\, (\{S_1\}\, D_1 \leftrightarrow \{S_1\}\, D^1). \tag{10}$$

Proof. (Sketch) Let us consider the program $P \cup P^1$. Due to the renaming process during program specialization, P and P^1 may share occurrences of basic predicates only. The basic predicates which are common to P and P^1, have identical definitions (modulo renaming of variables) in these two programs, and thus after performing the union of the two programs, we can delete the duplicate definitions of these predicates without affecting the least Herbrand model semantics.

Our notion of resultant together with the closedness and independence conditions, ensures that, by using the clauses in $P \cup P^1$, we can construct a specialization forest $\{U_i \mid i = 1, \ldots, n\}$ which is isomorphic to $\{T_i \mid i = 1, \ldots, n\}$, being $\{S_1\}\, D^1$ and the root-goal of U_1. Thus, Property (10) follows from Lemma 11.

We now show that (9) and (10) are equivalent. By definition of Herbrand model, Property (10) is equivalent to

$$\begin{array}{l} \text{for each ground substitution } \theta \text{ for } X_1, \ldots, X_k, \\ M(P \cup P^1) \models \{S_1\}\theta \;\text{ and }\; M(P \cup P^1) \models D_1\theta \quad \text{iff} \\ M(P \cup P^1) \models \{S_1\}\theta \;\text{ and }\; M(P \cup P^1) \models D^1\theta \end{array} \tag{11}$$

Since the basic predicates which are common to P and P^1, have the same definitions in P, P^1, and $P \cup P^1$, and no other predicate is common to P and P^1, we have that (11) is equivalent to

$$\begin{array}{l} \text{for each ground substitution } \theta \text{ for } X_1, \ldots, X_k, \\ M(P) \models \{S_1\}\theta \;\text{ and }\; M(P) \models D_1\theta \quad \text{iff} \\ M(P) \models \{S_1\}\theta \;\text{ and }\; M(P^1) \models D^1\theta \end{array} \tag{12}$$

which is logically equivalent to (9). □

6 Program Specialization via Generalized Resultants

In this section we introduce a program generalization technique which, when combined with the specialization techniques described in Section 4, allows us to derive very efficient specialized programs.

Our generalization technique consists in replacing the set $\{R_1, \ldots, R_n\}$ of resultants of a specialization tree T by a set $\{G_1, \ldots, G_m\}$ of clauses such that, for $i = 1, \ldots, m$, G_i is a generalization of a clause in $\{R_1, \ldots, R_n\}$. In the case of the *Append-Last* program presented in Section 3, by applying our technique we replace the clauses

$B1.$ $\mathit{last}([a], a) \leftarrow$
$B2.$ $\mathit{last}([X\,|\,Ys1], Z) \leftarrow \mathit{last}(Ys1, Z)$

which are the resultants of the specialization tree of Figure 1, by the clause:

6. $\mathit{last}(Ys, a) \leftarrow$

which is a generalization of clause $B1$.

Later in this section we present a result (see Theorem 15) which ensures that the specialized program constructed via the generalized clauses $G_1, \ldots, G_m$, instead of the resultants $R_1, \ldots, R_n$, is correct if by using $G_1, \ldots, G_m$ we can construct a suitable UR-tree which is isomorphic to T.

In the following definition we present the concept of *generalized resultants* of a specialization tree T, and indeed, by using a set of generalized resultants, we can construct the UR-tree isomorphic to T which is needed for ensuring the correctness of the program specialization.

Definition 13 (Generalized Resultants). Let T be a specialization tree for $\langle P, \{S\}\, D\rangle$. Let $\{R_1, \ldots, R_n\}$ be the set of resultants of all branches of T from topmost d-nodes to non-failing leaves. A set $\{G_1, \ldots, G_m\}$ of clauses, with $m \leq n$, is called a set of *generalized resultants* of T iff

α. for $i = 1, \ldots, m$, there exists a substitution θ_i such that $G_i\theta_i = R_i$ (we assume that the order of the clauses is not significant), and

β. the set $\{G_1, \ldots, G_m\}$ *validates* each subtree rooted in a topmost d-node of T in the sense specified by the next definition.

Definition 14 (Tree Validation). Let P be a program, S be a static goal, D be a dynamic defined atom, and $\{G_1, \ldots, G_m\}$ be a set of clauses, possibly not in P. Let T be a specialization tree for $\langle P, \{S\}\, D\rangle$. Let U be the specialization tree with root-goal $\{S\}\, D$ which is obtained by unfolding once $\{S\}\, D$ w.r.t. D using $\{G_1, \ldots, G_m\}$.
We say that $\{G_1, \ldots, G_m\}$ *validates* the tree T iff there exists a tree V which is isomorphic to U and it is obtained from T by first choosing in the leaves of T *zero or more* occurrences of dynamic defined atoms, and then performing *one* unfolding step w.r.t. each one of these occurrences using $\{G_1, \ldots, G_m\}$. (Thus, in particular, $\{G_1, \ldots, G_m\}$ *validates* T if T is isomorphic to U.)

The reader may check that in our introductory example, $\{$clause 6$\}$ validates both branches $b1$ and $b2$ of the specialization tree of Figure 1 and, therefore, it is a set of generalized resultants of that tree.

Let us now briefly explain why program specialization based on generalized resultants is correct.

Let T be a specialization tree for a program $\langle P, \{S\}\, D\rangle$. Let us assume that the predicates occurring in the static goals of T do not depend (in program P) on the predicate of D (this assumption is not actually a restriction because we may use predicate renaming to make it always true). Let Q be the set of definitions in P of the predicates different from the predicate of D.

We now show that if $\{G_1, \ldots, G_m\}$ is a set of generalized resultants of T, then we can construct a UR-tree isomorphic to T by using $Q \cup \{G_1, \ldots, G_m\}$ and also some extra clauses, say $New_1, \ldots, New_k$, which however, will not be part of the specialized program.

Since the upper portion of T made out of s-nodes can be constructed by using clauses in Q, we have only to show that for each subtree $\overline{T}$ rooted in a topmost d-node of T, we can construct, by using $Q \cup \{G_1, \ldots, G_m\}$, a tree $\overline{T}^1$ isomorphic to $\overline{T}$. The root-goal of $\overline{T}^1$ is equal to the root-goal, say $\{\overline{S}\}\, \overline{D}$, of $\overline{T}$. The construction of $\overline{T}^1$ is done in two steps: (Step 1) we first show that Condition (β) of Definition 13 allows for the replacement of the root-goal $\{\overline{S}\}\, \overline{D}$ of $\overline{T}^1$ by an equivalent goal $\{\overline{S}\}\, \mathit{NewD}$, where NewD is an atom with a new predicate suitably defined, and (Step 2) we then get the desired tree $\overline{T}^1$ isomorphic to $\overline{T}$ by unfolding $\{\overline{S}\}\, \mathit{NewD}$ w.r.t. NewD.

Let us now look at these steps in more detail.

(Step 1) Let NewD be an atom obtained from $\overline{D}$ by renaming the predicate symbol occurring in $\overline{D}$ to a fresh new symbol. Let $\{S_1\}\, D_1, \ldots, \{S_k\}\, D_k$ be the non-failing leaf-goals of $\overline{T}$. For $i = 1, \ldots, k$, let θ_i be the composition of the substitutions computed along the branch of $\overline{T}$ from the root to the leaf containing $\{S_i\}\, D_i$, and let New_i be the clause $\mathit{NewD}\, \theta_i \leftarrow D_i$. We have that:

$$M(Q \cup \{G_1,\ldots,G_m,New_1,\ldots,New_k\}) \models \forall X_1,\ldots,X_r\, (\{\overline{S}\}\, \overline{D} \leftrightarrow \{\overline{S}\}\, \mathit{NewD}) \quad (13)$$

where $X_1, \ldots, X_r$ are the variables occurring in $\{\overline{S}\}\, \overline{D}$ (or $\{\overline{S}\}\, \overline{\mathit{NewD}}$), because by hypothesis, $\{G_1, \ldots, G_m\}$ validates $\overline{T}$ and therefore, by unfolding the two goals $\{\overline{S}\}\, \overline{D}$ and $\{\overline{S}\}\, \mathit{NewD}$ using clauses in $\{G_1, \ldots, G_m, New_1, \ldots, New_k\}$ we get two isomorphic trees. Thus Equivalence (13) is a consequence of Lemma 11.

Now, by making use of (13), we may apply the goal replacement rule to the root-goal $\{\overline{S}\}\, \overline{D}$ of $\overline{T}$ and we derive the new goal $\{\overline{S}\}\, \mathit{NewD}$.

(Step 2) We unfold $\{\overline{S}\}\, \mathit{NewD}$ w.r.t. NewD using clauses $New_1, \ldots, New_k$, and the construction of these clauses ensures that we get exactly the same substitutions and the same non-failing leaf-goals of $\overline{T}$. Thus, the UR-tree $\overline{T}^1$ we have obtained is isomorphic to $\overline{T}$.

Notice that the UR-tree $\overline{T}^1$ constructed using $\{G_1, \ldots, G_m\}$ together with the set $\{New_1, \ldots, New_k\}$ of clauses, is not a specialization tree, because the goal replacement performed at Step 1 does not respect the restrictions of Definition 5, Point 2.

Notice also that the clauses in $\{New_1, \ldots, New_k\}$ are needed only for proving Equivalence (13), and not for the run-time evaluation of (instances of) D because the predicate of D does not depend on the predicate of NewD. Thus, $New_1, \ldots, New_k$ need not be included in the specialized program.

Now, the existence of an isomorphism between the specialization tree T and the UR-tree T^1 constructed by using the generalized resultants $\{G_1, \ldots, G_m\}$ of

T, allows us to prove a fact analogous to Lemma 11. The rest of the proof of correctness of our specialization technique is similar to the proof of Theorem 12. In particular, if we can construct a *closed* specialization forest, we have that, except for the basic predicates, no predicate occurring in the initial program is needed in the specialized program. Thus, the specialized program is made out of all (possibly renamed) generalized resultants together with the clauses defining the basic predicates.

The procedure for extracting specialized programs from closed specialization forests by using generalized resultants, is a modification of the construction presented in Definition 8 obtained by using 'generalized resultants', instead of 'resultants'. In the following theorem we refer to program specializations performed according to this modified construction.

Theorem 15 (Correctness of Specialization via Generalized Resultants). *Let P be a program, S_1 be a static goal, and D_1 be a dynamic atom. Let $\langle P^1, D^1\rangle$ be a specialization of $\langle P, D_1\rangle$ w.r.t. S_1 constructed from a* closed *specialization forest for $\langle P, \{S_1\}\, D_1\rangle$ by using generalized resultants. Assuming that $\{X_1,\ldots,X_k\} = vars(\{S_1\}\, D_1)$, we have that:*

$$\textit{for each ground substitution } \theta \textit{ for } X_1,\ldots,X_k, \quad \textit{if} \quad M(P) \models S_1\theta \quad \textit{then} \quad M(P) \models D_1\theta \;\; \textit{iff} \;\; M(P^1) \models D^1\theta \qquad (14)$$

or, equivalently,

$$M(P \cup P^1) \models \forall X_1,\ldots,X_k\, (\{S_1\}\, D_1 \leftrightarrow \{S_1\}\, D^1). \qquad (15)$$

The reader may notice that when we use generalized resultants, instead of resultants, the independence condition is no longer necessary to ensure that we can construct suitable isomorphic trees (or, more generally, forests) which are needed to prove the correctness of program specialization. This explains why the independence condition does not appear in the hypotheses of Theorem 15. One can also show that, if the independence condition holds for a given specialization tree, then the resultants of that tree are also generalized resultants.

It may be the case that no set of generalized resultants exists for a given specialization forest. When this happens, to apply our technique one has to construct a different specialization forest. The study of the strategies for obtaining specialization forests for which generalized resultants exist, is beyond the scope of this paper.

The following example, taken from [1], shows a complete derivation by program specialization with generalized resultants. It is important to notice that the derivation in [1] is supported by the discovery and the proof of crucial lemmas, whereas we perform unfolding and generalization steps only.

Example 1 (Quicksort of Decreasing Lists). Let us consider the following program *Quicksort*, where $\geq$ and $<$ may be regarded as basic predicates:

1. $quicksort([\,], Ls, Ls) \leftarrow$

2. $quicksort([X|Xs], Ts, Ys) \leftarrow partition(Xs, X, Ss, Bs),$
$\qquad quicksort(Bs, Ts, Ys1),$
$\qquad quicksort(Ss, [X|Ys1], Ys)$
3. $partition([\,], X, [\,], [\,]) \leftarrow$
4. $partition([B|Vs], A, [B|Ss], Bs) \leftarrow A \geq B,\ partition(Vs, A, Ss, Bs)$
5. $partition([B|Vs], A, Ss, [B|Bs]) \leftarrow A < B,\ partition(Vs, A, Ss, Bs)$
6. $decr([\,]) \leftarrow$
7. $decr([X]) \leftarrow$
8. $decr([X1, X2|Xs]) \leftarrow X1 \geq X2, decr([X2|Xs])$.

Suppose that we want to specialize *Quicksort* w.r.t. the static part of the goal $\{decr(Xs)\}\, quicksort(Xs, Ys, Zs)$.

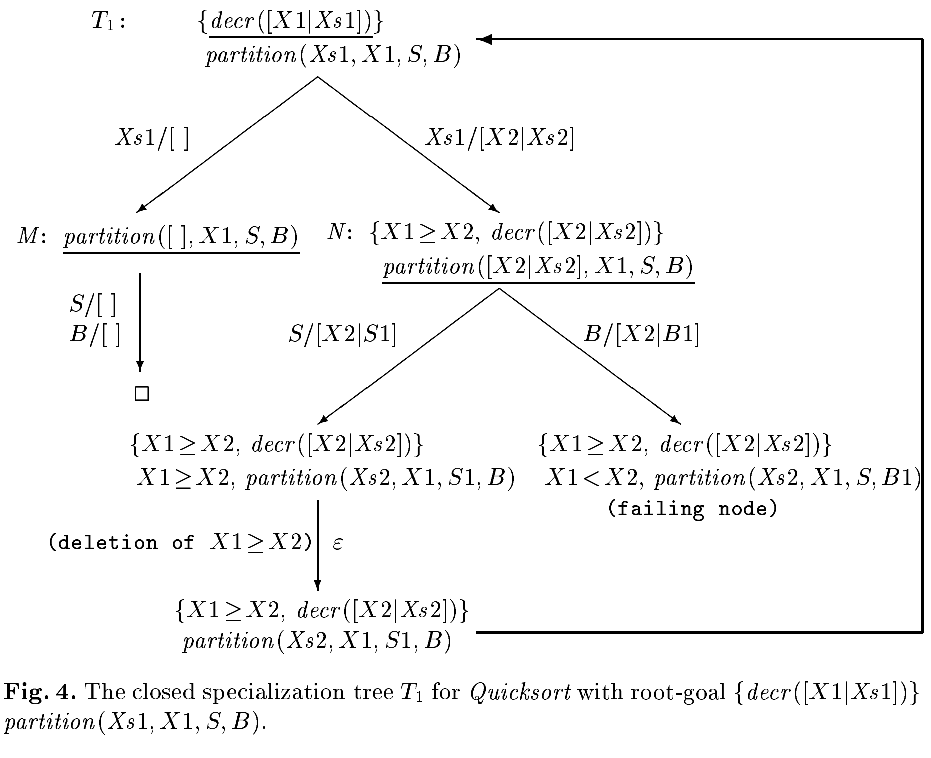
We will see that, for this program specialization we first need to specialize *Quicksort* w.r.t. the static part of $\{decr([X1|Xs1])\}\, partition(Xs1, X1, S, B)$.

To perform this auxiliary specialization, we construct the closed specialization tree, say T_1, for the pair:

$$\langle\ Quicksort,\quad \{decr([X1|Xs1])\}\, partition(Xs1, X1, S, B)\ \rangle$$

which is depicted in Figure 4. The resultants of that tree are:

9. $partition([\,], X, [\,], [\,]) \leftarrow$
10. $partition([X2|Xs2], X1, [X2|S1], B) \leftarrow partition(Xs2, X1, S1, B)$.



T_1:
$\{decr([X1|Xs1])\}$
$partition(Xs1, X1, S, B)$
$Xs1/[\,]$
$Xs1/[X2|Xs2]$
M: $partition([\,], X1, S, B)$
N: $\{X1 \geq X2,\ decr([X2|Xs2])\}$
$partition([X2|Xs2], X1, S, B)$
$S/[\,]$
$B/[\,]$
$\square$
$S/[X2|S1]$
$B/[X2|B1]$
$\{X1 \geq X2,\ decr([X2|Xs2])\}$
$X1 \geq X2,\ partition(Xs2, X1, S1, B)$
$\{X1 \geq X2,\ decr([X2|Xs2])\}$
$X1 < X2,\ partition(Xs2, X1, S, B1)$
(failing node)
(deletion of $X1 \geq X2$)
ε
$\{X1 \geq X2,\ decr([X2|Xs2])\}$
$partition(Xs2, X1, S1, B)$

Fig. 4. The closed specialization tree T_1 for *Quicksort* with root-goal $\{decr([X1|Xs1])\}$ $partition(Xs1, X1, S, B)$.

Now we consider the following clause:

11. $partition(Xs, X, Xs, [\,]) \leftarrow$

which is a generalization of clause 9. We have that {clause 11} is a set of generalized resultants of T_1. To show this, we have to verify that {clause 11} validates all subtrees of T_1 rooted in the topmost d-nodes of T_1, which are node M labeled by the goal

$partition([\,], X1, S, B)$

and node N labeled by the goal

$\{X1 \geq X2, decr([X2|Xs2])\}\ partition([X2|Xs2], X1, S, B)$.

The following facts i) and ii) hold.

i) By unfolding the goal in M using clause 11 we get a specialization tree, say U_M, with root-goal $partition([\,], X1, S, B)$, a unique arc labeled by the substitution $\{S/[\,], B/[\,]\}$, and leaf-goal $\square$. U_M is isomorphic to the subtree of T_1 rooted in M. Thus, {clause 11} validates this subtree.

ii) By unfolding the goal in N using clause 11 we get the specialization tree U_N depicted in Figure 5. By unfolding the non-failing leaf-goal $\{X1 \geq X2, decr([X2|Xs2])\}\ partition(Xs2, X1, S1, B)$ of the subtree of T_1 rooted in N, we get the tree V_N depicted in Figure 5. U_N and V_N are isomorphic and thus {clause 11} also validates the subtree of T_1 rooted in N.

U_N :

$$\begin{tikzcd}
N\!: \begin{array}{c} \{X1 \geq X2,\ decr([X2|Xs2])\} \\ \underline{partition([X2|Xs2], X1, S, B)} \end{array} \arrow[d, "\begin{array}{l} S/[X2|Xs2] \\ B/[\,] \end{array}"'] \\
\{X1 \geq X2,\ decr([X2|Xs2])\}
\end{tikzcd}$$

V_N :

$$\begin{tikzcd}
 & N\!: \begin{array}{c} \{X1 \geq X2,\ decr([X2|Xs2])\} \\ \underline{partition([X2|Xs2], X1, S, B)} \end{array} \arrow[dl, "S/[X2|S1]"'] \arrow[dr, "B/[X2|B1]"] & \\
\begin{array}{c} \{X1 \geq X2,\ decr([X2|Xs2])\} \\ X1 \geq X2,\ partition(Xs2, X1, S1, B) \end{array} \arrow[d, "\varepsilon"'] & & \texttt{(failing node)} \\
\begin{array}{c} \{X1 \geq X2,\ decr([X2|Xs2])\} \\ \underline{partition(Xs2, X1, S1, B)} \end{array} \arrow[d, "\begin{array}{l} S1/Xs2 \\ B/[\,] \end{array}"'] & & \\
\{X1 \geq X2,\ decr([X2|Xs2])\} & &
\end{tikzcd}$$

Fig. 5. The isomorphic trees U_N and V_N for the validation of the subtree rooted of T_1 in N.

Thus, the program made out of the renamed clause:

11.r $\mathit{partition}^1(Xs, X, Xs, [\,]) \leftarrow$

with input goal $\mathit{partition}^1(Xs1, X1, S, B)$ is a specialization of *Quicksort* with input goal $\mathit{partition}(Xs1, X1, S, B)$ w.r.t. the static goal $\mathit{decr}([X1|Xs1])$. The correctness of the derivation is ensured by Theorem 15 because T_1 is closed (recall that the independence property is not needed when we use generalized resultants).

Now, we are ready to address the main problem of specializing *Quicksort* w.r.t. the static part of the goal $\{\mathit{decr}(Xs)\}\, \mathit{quicksort}(Xs, Ys, Zs)$. This problem can be solved by adding clause 11.r to *Quicksort* and by looking for the specialization of the program $\mathit{Quicksort} \cup \{\text{clause}11.r\}$ w.r.t. the static part of the goal $\{\mathit{decr}(Xs)\}\, \mathit{quicksort}(Xs, Ys, Zs)$.

Therefore, we construct the closed specialization forest F for $\langle \mathit{Quicksort} \cup \{\text{clause}11.r\},\ \{\mathit{decr}(Xs)\}\, \mathit{quicksort}(Xs, Ys, Zs)\rangle$ made out of the trees R_1 and R_2 depicted in Figure 6. During the construction of F we use the result of the auxiliary program specialization we have performed, that is, we use the fact that $\langle\{\text{clause } 11.r\},\ \mathit{partition}^1(Xs1, X1, S, B)\rangle$ is a specialization of $\langle \mathit{Quicksort}, \mathit{partition}(Xs1, X1, S, B)\rangle$ w.r.t. $\mathit{decr}([X1|Xs1])$ (see Point 2.iv of Definition 5 where it is indicated how to use previous program specializations during the construction of specialization trees).

The resultants of R_1 are:

12. $\mathit{quicksort}([\,], Ys, Ys) \leftarrow$
13. $\mathit{quicksort}([X1|S], Ys1, Zs) \leftarrow \mathit{quicksort}(S, [X1|Ys1], Zs)$

and the resultants of R_2 are:

14. $\mathit{quicksort}([\,], [X1|Ys1], [X1|Ys1]) \leftarrow$
15. $\mathit{quicksort}([Y1|S1], [X1|Ys1], Zs) \leftarrow \mathit{quicksort}(S1, [Y1, X1|Ys1], Zs)$.

After predicate renaming we get our final specialized program which reverses a list in linear time:

12.r $\mathit{quicksort}^1([\,], Ys, Ys) \leftarrow$
13.r $\mathit{quicksort}^1([X1|S], Ys1, Zs) \leftarrow \mathit{quicksort}^2(S, [X1|Ys1], Zs)$
14.r $\mathit{quicksort}^2([\,], [X1|Ys1], [X1|Ys1]) \leftarrow$
15.r $\mathit{quicksort}^2([Y1|S1], [X1|Ys1], Zs) \leftarrow \mathit{quicksort}^2(S1, [Y1, X1|Ys1], Zs)$.

The correctness of the derivation of our final program is a consequence of Theorem 12 because the specialization forest F is closed and the independence property holds. To see this, notice that in R_1 there is one topmost d-node only, that is, the root, and thus the independence property trivially holds. In R_2 we have precisely two topmost d-nodes, labeled by the goals

$\mathit{quicksort}([\,], [X1|Ys1], Zs)$ and

$\{X1 \geq Y1, \mathit{decr}([Y1|S1])\}\, \mathit{quicksort}([Y1|S1], [X1|Ys1], Zs)$,

respectively, and the atoms with predicate *quicksort* have no common instances. Therefore the two nodes are independent.

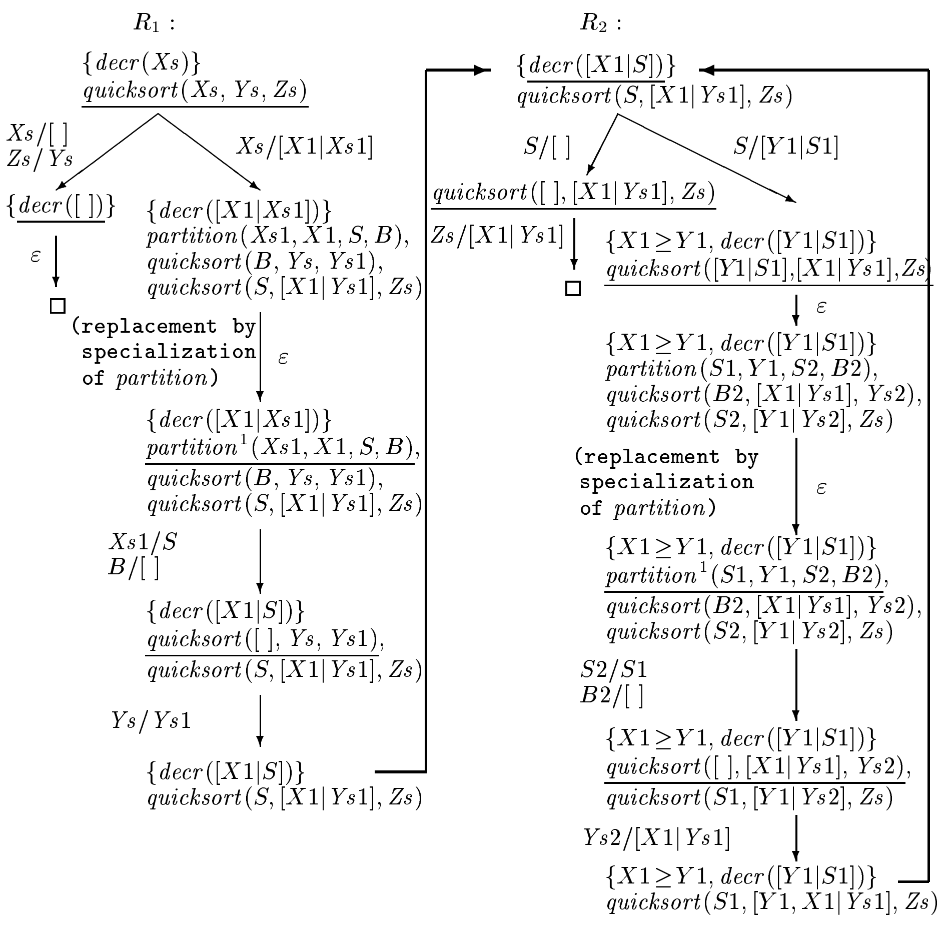

R_1 :
{decr(Xs)}
quicksort(Xs, Ys, Zs)
Xs/[ ]
Zs/Ys
Xs/[X1|Xs1]
{decr([ ])}
ε
{decr([X1|Xs1])}
partition(Xs1, X1, S, B),
quicksort(B, Ys, Ys1),
quicksort(S, [X1|Ys1], Zs)
(replacement by specialization of partition)
ε
{decr([X1|Xs1])}
partition^1(Xs1, X1, S, B),
quicksort(B, Ys, Ys1),
quicksort(S, [X1|Ys1], Zs)
Xs1/S
B/[ ]
{decr([X1|S])}
quicksort([ ], Ys, Ys1),
quicksort(S, [X1|Ys1], Zs)
Ys/Ys1
{decr([X1|S])}
quicksort(S, [X1|Ys1], Zs)
R_2 :
{decr([X1|S])}
quicksort(S, [X1|Ys1], Zs)
S/[ ]
S/[Y1|S1]
quicksort([ ], [X1|Ys1], Zs)
Zs/[X1|Ys1]
{X1≥Y1, decr([Y1|S1])}
quicksort([Y1|S1],[X1|Ys1],Zs)
ε
{X1≥Y1, decr([Y1|S1])}
partition(S1, Y1, S2, B2),
quicksort(B2, [X1|Ys1], Ys2),
quicksort(S2, [Y1|Ys2], Zs)
(replacement by specialization of partition)
ε
{X1≥Y1, decr([Y1|S1])}
partition^1(S1, Y1, S2, B2),
quicksort(B2, [X1|Ys1], Ys2),
quicksort(S2, [Y1|Ys2], Zs)
S2/S1
B2/[ ]
{X1≥Y1, decr([Y1|S1])}
quicksort([ ], [X1|Ys1], Ys2),
quicksort(S1, [Y1|Ys2], Zs)
Ys2/[X1|Ys1]
{X1≥Y1, decr([Y1|S1])}
quicksort(S1, [Y1, X1|Ys1], Zs)

Fig. 6. A closed specialization forest with the two UR-trees R_1 and R_2.

7 Related Work and Concluding Remarks

We have proposed some program specialization techniques which extend the techniques for the partial evaluation of logic programs based on [10]. As in [10] we specialize a program w.r.t. the context where it is used. However, in [10] this context can be defined in terms of bindings of the form $X = t$, where t is a possibly non-ground term, whereas we allow for the definition of more complex contexts, such as those definable by means of any predicate computed by a logic program.

We believe that most of the automated tools and systems developed for partial evaluation methods based on [10], can easily be adapted to take into account our extended techniques, because they are based on similar concepts. Our techniques can be implemented in an efficient way and, in particular, the generalization step proposed in Section 6 requires in practice only a very limited search.

The idea of specializing logic programs w.r.t. complex contexts is strongly related to the techniques of *internal specialization* [14] and *finite differencing* [12], in the case of functional and set-based imperative languages, respectively. However, it is hard to establish a formal relationship among these techniques, because of the different formalisms used.

In the field of functional programming various extensions of the partial evaluation technique have been proposed to deal with properties of the input data. *Parametrized Partial Evaluation* [3] allows for the specialization of programs w.r.t. properties of the input data which are described by means of approximated representations based on abstract interpretations. *Generalized Partial Computation* [5] allows for the use of theorem provers for dealing with properties which restrict the domains of the functional programs to be specialized.

In our approach, we avoid the use of abstract interpretations and we specify properties of the input data in a precise way. We also avoid the need for very sophisticated theorem provers by using only the unfolding rule together a limited form of the goal replacement rule.

Also in the area of logic programs there are various papers which extend partial evaluation with objectives similar to ours. Bossi et al. [1] introduce a method based on unfolding and folding rules à la Burstall and Darlington [2], with some extra rules to deal with contexts (or *constraining* predicates). However, as in [5], most of the power of this method relies on the proof of properties of the program to be specialized, which makes its automation very hard. Moreover, we believe that by using the unfolding and folding rules according to the method of [1], without proving extra program properties, it is impossible to work out the examples we have presented in this paper.

The method presented in [11] is an adaptation of [1] to the case of Prolog programs with the left-to-right, depth-first control strategy. No theorem proving capabilities are allowed in this method, which as a result, is strictly less powerful than the one described in [1].

Finally, de Waal and Gallagher [4, 6] have proposed various techniques to extend partial evaluation. In particular, the notion of *partial evaluation with conditions* is strongly related to our specialization tree. However, similarly to [3], de Waal and Gallagher's approach makes use of a number of abstract interpretation techniques, whereas as already mentioned, our techniques are based on syntactic transformation rules only.

Acknowledgements

We would like to thank the organizers of the Dagstuhl Seminar on "Partial Evaluation" for their kind invitation and the anonymous referees for comments and suggestions. We acknowledge some very fruitful conversations with J. Gallagher and M. H. Sørensen. Our thanks also go to M. Leuschel and B. Martens for pointing out to us the example shown in Section 3.

References

1. A. Bossi, N. Cocco, and S. Dulli. A method for specializing logic programs. *ACM Transactions on Programming Languages and Systems*, 12(2):253–302, April 1990.
2. R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, January 1977.
3. C. Consel and S.C. Khoo. Parameterized partial evaluation. *ACM Transactions on Programming Languages and Systems*, 15(3):463–493, 1993.
4. D. A. de Waal and J. P. Gallagher. Specialization of a unification algorithm. In T. Clement and K.-K. Lau, editors, *Logic Program Synthesis and Transformation, Proceedings LOPSTR '91, Manchester, U.K.*, Workshops in Computing, pages 205–221. Springer-Verlag, 1992.
5. Y. Futamura and K. Nogi. Generalized partial computation. In D. Bjørner, A. P. Ershov, and N. D. Jones, editors, *Partial Evaluation and Mixed Computation*, pages 133–151. North-Holland, 1988.
6. J. P. Gallagher and D.A. de Waal. Deletion of redundant unary type predicates from logic programs. In *Proceedings of LoPSTr'92, Manchester, U.K.*, pages 151–167. Springer–Verlag, 1993.
7. N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice Hall, 1993.
8. M. Leuschel and B. Martens. Partial deduction of the ground representation and its application to integrity checking. In J. W. Lloyd, editor, *Proceedings of the 1995 International Logic Programming Symposium (ILPS'95)*, pages 495–509. MIT Press, 1995.
9. J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1987. Second Edition.
10. J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *Journal of Logic Programming*, 11:217–242, 1991.
11. U. W. Neumerkel. *Specialization of Prolog Programs with Partially Static Goals and Binarization.* PhD thesis, Technical University Wien, Austria, 1992.
12. R. Paige and S. Koenig. Finite differencing of computable expressions. *ACM Transactions on Programming Languages and Systems*, 4(3):402–454, 1982.
13. A. Pettorossi and M. Proietti. Transformation of logic programs: Foundations and techniques. *Journal of Logic Programming*, 19,20:261–320, 1994.
14. W. L. Scherlis. Program improvement by internal specialization. In *Proc. 8th ACM Symposium on Principles of Programming Languages, Williamsburgh, Va*, pages 41–49. ACM Press, 1981.
15. H. Tamaki and T. Sato. A generalized correctness proof of the unfold/fold logic program transformation. Technical Report 86-4, Ibaraki University, Japan, 1986.