In: J. Gallagher (Ed.) Proceedings of the Sixth International Workshop on Logic Program Synthesis and Transformation (LoPSTr '96), Stockholm, Sweden, August 28–30, 1996, Lecture Notes in Computer Science 1207, Springer, 1997.

Enhancing Partial Deduction via Unfold/Fold Rules

Alberto Pettorossi [1] Maurizio Proietti [2] Sophie Renault [1]

[1] Department of Informatics, University of Roma Tor Vergata,
Via della Ricerca Scientifica, 00133 Roma, Italy. {adp,renault}@iasi.rm.cnr.it
[2] IASI-CNR, Viale Manzoni 30, 00185 Roma, Italy. proietti@iasi.rm.cnr.it

Abstract. We show that sometimes partial deduction produces poor program specializations because of its limited ability in (i) dealing with conjunctions of recursively defined predicates, (ii) combining partial evaluations of alternative computations, and (iii) taking into account unification failures. We propose to extend the standard partial deduction technique by using versions of the definition rule and the folding rule which allow us to specialize predicates defined by disjunctions of conjunctions of goals. We also consider a case split rule to take into account unification failures. Moreover, in order to perform program specialization via partial deduction in an automatic way, we propose a transformation strategy which takes as parameters suitable substrategies for directing the application of every transformation rule.
Finally, we show through two examples that our partial deduction technique is superior to standard partial deduction. The first example refers to the automatic derivation of the Knuth-Morris-Pratt string matching algorithm, and the second example refers to the construction of a parser for a given regular expression. In both examples, the specialized programs are derived starting from naive, non-deterministic initial programs, whereas standard partial deduction can derive similar specialized programs only when complex, deterministic initial programs are provided.

1 Introduction

Partial deduction is a powerful tool for program specialization and it has been successfully applied for the derivation of very efficient specialized programs. In practice, however, some efficiency improvements in the specialized programs cannot be realized, because partial deduction, as formalized by Lloyd and Shepherdson [13], here also called standard partial deduction, is not capable to perform specializations w.r.t. a conjunction of two or more atoms at the same time. Moreover, standard partial deduction cannot combine together the evaluation of several alternative computations, nor it takes into account the negative information which is generated by unification failures. In this paper we study the problem of how to overcome these limitations, and the solution we propose is based on the enhancement of the basic rules used for performing partial deduction. We also propose a strategy for applying these enhanced rules. This strategy allows us to achieve high levels of efficiency by specializing *disjunctions of conjunctions of goals*. For this reason the novel technique we propose is called *disjunctive partial deduction*. The specialized programs we derive avoid the use of intermediate data structures and unnecessary nondeterministic predicates.

In Section 2 we first illustrate the power of the standard partial deduction technique by recalling the derivation of the programs corresponding to the finite automata generated by the Knuth-Morris-Pratt string matching algorithm. Then we illustrate the limitations of standard partial deduction by showing that the same efficient matching algorithms can only be derived by starting from initial programs which already specify some suitable details on the matching process itself. In Section 3 we indicate how within the framework of program transformation, an improved version of the partial deduction technique can be obtained by providing some enhanced transformation rules and a related strategy for their application. In Section 4 we present two examples which illustrate the use of those rules and strategy. Finally, in Section 5 we briefly compare our techniques with related proposals reported in the literature.

2 Partial Deduction and Unfold/Fold Transformations

We recall first how to recast in the unfold/fold transformation framework [2, 15, 25] the standard techniques for partial deduction of definite programs, as they have been formalized by Lloyd and Shepherdson [6, 13].

2.1 Transformation Rules and Strategy for Standard Partial Deduction

It is well known that partial deduction can be seen as a particular instance of the unfold/fold transformation methodology where the following three rules are available (see, for instance, [17, 20]) for transforming a given program P into a new one.

- **Atomic Definition**. It consists in adding to P a clause, called a *definition clause*, of the form $newp(X_1, \ldots, X_m) \leftarrow A$, where $newp$ is a new predicate symbol, A is an atom, and $X_1, \ldots, X_m$ are the variables occurring in A.
- **Unfolding** of a clause C in P w.r.t. an atom A of its body. It consists in replacing C by all clauses obtained by applying SLD-resolution to C and every clause in P whose head is unifiable with A. In particular, the unfolding of a clause w.r.t. an atom which is not unifiable with any head in P, amounts to the deletion of that clause.
- **Atomic Folding** of an atom A in the body of a clause C using a definition clause D. It consists in replacing A, if it is an instance of the body of D, by the corresponding instance of the head of D. Atomic folding of an atom in the body of a definition clause is not allowed.

These rules are instances of Tamaki and Sato's transformation rules and in the case of definite programs, they preserve the success set and the finite failure set semantics [22, 25].

In order to perform partial deduction of a given program whereby getting a specialized program, these three rules have to be applied according to a suitable strategy. We now present one such strategy, which we call *standard strategy*, or simply $\mathcal{S}$. This strategy, depicted in Figure 1, is very similar to the one illustrated by Gallagher in [6]. In Section 3.1 we will propose an enhanced version of this strategy for performing disjunctive partial deduction.

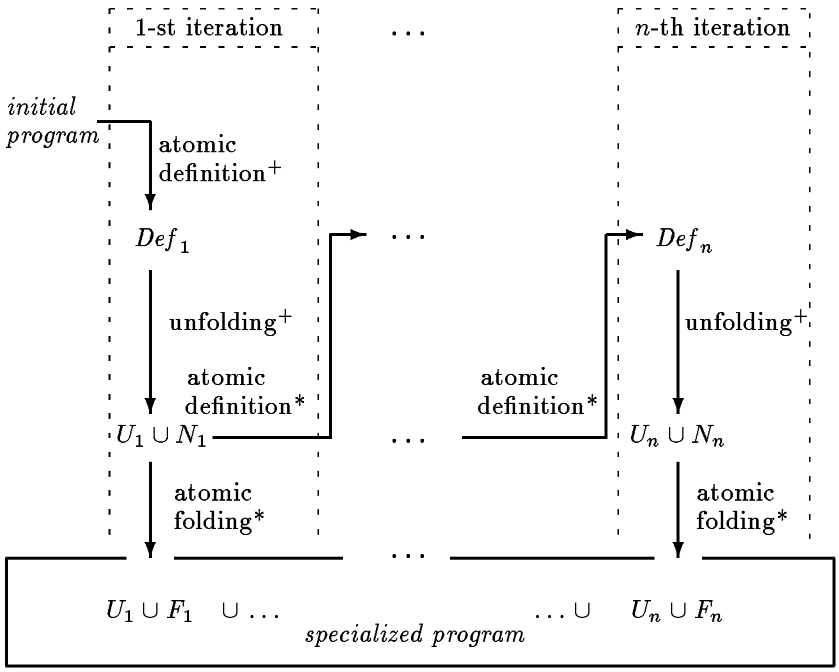

Fig. 1. The strategy $\mathcal{S}$ for standard partial deduction. r^+ denotes one or more applications of the rule r, and r^* denotes zero or more applications of r.

Standard Strategy $\mathcal{S}$ for Partial Deduction. Given a definite program P and a goal G w.r.t. which we want to specialize P, we start off by introducing a set Def_1 consisting of a definition clause of the form $newp(X_1, \ldots, X_m) \leftarrow A$ for each atom A in G, where $X_1, \ldots, X_m$ are the variables occurring in A. Then, from the program $P \cup Def_1$ we construct a sequence of programs of the form: $P \cup Def_i \cup Q_i$, for $i \geq 1$, with $Q_1 = \{\}$. We construct the program $P \cup Def_{i+1} \cup Q_{i+1}$ from the program $P \cup Def_i \cup Q_i$, during the i-th iteration of the strategy, as follows.

We unfold once or more times each clause in Def_i and we derive a new program $P \cup U_i \cup N_i \cup Q_i$, where U_i and N_i are the unit and non-unit clauses, respectively, derived by unfolding the clauses of Def_i. We then introduce a new set Def_{i+1} of definitions whose bodies are *generalizations* of the atoms occurring in the bodies of the clauses in N_i which cannot be folded using already introduced definition clauses. We fold all these atoms using the clauses in $Def_1 \cup \ldots \cup Def_{i+1}$, thereby obtaining the set F_i of clauses and the new program $P \cup Def_{i+1} \cup Q_{i+1}$, where Q_{i+1} is $U_i \cup F_i \cup Q_i$.

Partial deduction terminates for $i = n$ iff all atoms occurring in the bodies of the clauses of N_n can be folded using the definitions occurring in $Def_1 \cup \ldots \cup Def_n$, that is, Def_{n+1} is empty because no new definition is needed for folding the atoms in the bodies of N_n. The final program is $P \cup Q_{n+1}$, and P can be dropped if we are interested only in goals relative to predicates defined by Def_1, which are the goals w.r.t. which P has been specialized, because during the evaluation of these goals, only calls of predicates in Q_{n+1} may be generated. □

This strategy is parametric in the sense that it takes as parameters subsidiary strategies for directing the unfolding steps and the atomic definition steps.

2.2 Partial Deduction and the KMP Test

The success w.r.t. the so called KMP test [24] is regarded as a significant achievement of partial deduction, and indeed it is possible to derive a program which behaves like the deterministic finite automaton generated by the Knuth-Morris-Pratt string matching algorithm by applying the standard strategy for partial deduction to a general string matching program like the one of Figure 2(A) [5, 6].

Initial matching program for the KMP test		Specialized program w.r.t. $match([a,a,b],S)$
$match(P,S) \leftarrow$ $\quad match1(P,S,P,S)$ $match1([\,],X,Y,Z) \leftarrow$ $match1([A\|Ps],[A\|Ss],P,S) \leftarrow$ $\quad match1(Ps,Ss,P,S)$ $match1([A\|Ps],[B\|Ss],P,[C\|S]) \leftarrow$ $\quad A \neq B,\ match1(P,S,P,S)$	by standard partial deduction $\longrightarrow$	$pd\_match(S) \leftarrow m1(S)$ $m1([a\|S]) \leftarrow m2(S)$ $m1([X\|S]) \leftarrow a \neq X,\ m1(S)$ $m2([a\|S]) \leftarrow m3(S)$ $m2([X\|S]) \leftarrow a \neq X,\ m1(S)$ $m3([b\|S]) \leftarrow$ $m3([X\|S]) \leftarrow b \neq X,\ m2([X\|S])$
(A)		(B)

Fig. 2. (A) Initial string matching program for the KMP test. (B) Specialized program derived by standard partial deduction, where $pd\_match(S)$ is defined by the clause $pd\_match(S) \leftarrow match([a,a,b],S)$. It behaves like the deterministic finite automaton generated by the KMP string matching algorithm.

The aim of partial deduction in the KMP test is to specialize the program of Figure 2(A) w.r.t. a particular goal. If we choose the goal $match([a,a,b],S)$, that is, we want to derive a program which tells us whether or not the pattern $[a,a,b]$ occurs in string S, we introduce the definition clause:

$$pd\_match(S) \leftarrow match([a,a,b],S)$$

Then by using the standard strategy for partial deduction we get the program given in Figure 2(B) which performs the desired task and corresponds to the deterministic finite automaton generated by the KMP string matching algorithm accepting the language $\Sigma^* aab \Sigma^*$ [6]. In this sense we say that standard partial deduction passes the KMP test.

2.3 Some Limitations of Standard Partial Deduction

This ability of standard partial deduction to pass the KMP test is related to the initial string matching program which has been considered. In particular, let us consider the case, call it *strong* KMP *test*, where instead of the program of Figure 2(A), the initial program is the naive string matching program depicted in Figure 3(A).

Initial matching program for the strong KMP test

$$\begin{array}{l} naive\_match(P,S) \leftarrow \\ \quad append(Left,P,X), \\ \quad append(X,Right,S) \\ append([\,],Y,Y) \leftarrow \\ append([C|X],Y,[C|Z]) \leftarrow \\ \quad append(X,Y,Z) \end{array}$$

(A)

by standard partial deduction $\longrightarrow$

Specialized program w.r.t. $naive\_match([a,a,b],S)$

$$\begin{array}{l} pd\_naive\_match([a,a,b|S]) \leftarrow \\ pd\_naive\_match([A|S]) \leftarrow \\ \qquad new(X,Y), \\ \qquad append(Y,Z,S) \\ new([\,],[a,a,b]) \leftarrow \\ new([C|X],[C|Y]) \leftarrow \\ \qquad new(X,Y) \\ append([\,],Y,Y) \leftarrow \\ append([C|X],Y,[C|Z]) \leftarrow \\ \qquad append(X,Y,Z) \end{array}$$

(B)

Fig. 3. (A) Initial string matching program for the strong KMP test. (B) Specialized program derived by standard partial deduction, where $pd\_naive\_match(S)$ is defined by the clause $pd\_naive\_match(S) \leftarrow naive\_match([a,a,b],S)$. It does *not* behave like the deterministic finite automaton generated by the KMP string matching algorithm.

This program checks whether or not the pattern P occurs in the string S by looking for two strings *Left* and *Right* such that S is the concatenation of *Left*,

P, and *Right*. In this case, in order to derive the specialized program w.r.t. the goal $naive\_match([a,a,b],S)$, we start off from the definition clause:

1. $pd\_naive\_match(S) \leftarrow naive\_match([a,a,b],S)$

Then by using the standard partial deduction strategy $\mathcal{S}$, we get the specialized program given in Figure 3(B). The particular subsidiary unfolding strategy which we have used, consists in (i) unfolding the leftmost atom in the body and then (ii) unfolding as long as possible the resulting clauses w.r.t. any atom which unifies with exactly one clause head. Unfortunately, the program of Figure 3(B) does *not* behave like the deterministic finite automaton generated by the KMP string matching algorithm. Indeed, when a mismatch occurs, the search for the pattern P within the string S is always restarted after a shift of exactly one character from where the pattern was assumed to occur.

Moreover, no sequence of applications of the atomic definition, unfolding, and atomic folding rules can produce from clause 1 and the clauses in Figure 3(A) a program which behaves like the deterministic finite automaton generated by the KMP algorithm.

Thus, we say that standard partial deduction does *not* pass the strong KMP test. This limitation also applies to the automatic derivations based on partial evaluation techniques [9] of specialized KMP matching programs both in functional and logic languages, which have been published in the literature. Also Glück and Klimov in [8], who apply the supercompilation technique, consider an initial program which is similar to the one presented in Figure 2(A).

The inability of standard partial deduction to pass the strong KMP test is due to the following facts:

- standard partial deduction may fail to propagate information across conjunctions of recursively defined predicates, because it is not able to combine separate recursive computations into a unique computation;
- standard partial deduction may fail to propagate information across disjunctions of atoms, because it does not combine partial evaluations of several alternative computations; and
- standard partial deduction may not use negative information due to unification failures.

Technically, the weakness of standard partial deduction is due to the fact that it enforces a very restricted way of introducing new definition clauses, that is, a new predicate may be defined only by introducing exactly one clause which has exactly one atom in its body. This restriction also affects the folding rule, because folding steps are applied by using definition clauses of that form only. The expert reader may realize that these limitations indeed correspond to the fact that partial evaluation à la Lloyd-Shepherdson is performed w.r.t. atomic goals only.

Moreover, standard partial deduction does not provide any support for reasoning by cases, and there is no way of introducing inequalities to take into account unification failures. Consequently, it is impossible to obtain the program of Figure 2(B) if inequalities are not already present in the initial program, as it is the case for the program of Figure 2(A).

In the next section, we show that these limitations on the power of standard partial deduction can be overcome by some transformation techniques based on the use of unfolding/folding rules which are more powerful than the three rules presented in Section 2.1.

3 Disjunctive Partial Deduction via Enhanced Unfold/Fold Rules

Now we introduce our enhanced transformation rules for making the partial deduction technique more powerful, and we present a new strategy for applying these rules. We will then briefly discuss the issue of the mechanization of this new strategy and also the issue of which semantics are preserved while applying the enhanced rules.

3.1 Transformation Rules and Strategy for Disjunctive Partial Deduction

The main improvement to the partial deduction techniques we propose, relies on the enhancement of both the definition rule and the folding rule. For the definition rule we allow new predicates to be defined by means of *one or more* non-recursive clauses which contain *one or more* atoms in their bodies. When combined with folding, this rule allows us to specialize a program w.r.t. a *disjunction of conjunctions of atoms*. For the folding rule we allow to fold *several* clauses at a time, and thus, our extension is similar to the one proposed in [7]. We also use the standard unfolding rule which we have presented in Section 2.1. In addition to the definition, unfolding, and folding rules, we use a subsumption rule, and a case split rule. The subsumption rule is used to eliminate clauses which are implied by other clauses in the program. By the case split rule we replace a clause by two new clauses corresponding to mutually exclusive and exhaustive instantiations of its head. The aim of case split is to increase determinism of programs, as we will indicate below.

Now we formally present our enhanced rules for transforming a given program P into a new one.

- **Definition**. It consists in adding to P the following n clauses, called *definition clauses*, of the form:
  $newp(X_1,\ldots,X_m) \leftarrow Body_1,\ldots, newp(X_1,\ldots,X_m) \leftarrow Body_n$,
  where $newp$ is a new predicate symbol, $Body_1,\ldots, Body_n$ contain only predicates occurring in P, and $X_1,\ldots,X_m$ are variables occurring in $Body_1,\ldots,$ $Body_n$.
- **Unfolding** of a clause C in P w.r.t. an atom A of its body. It consists in replacing C by all clauses obtained by applying SLD-resolution to C and every clause in P whose head is unifiable with A. In particular, the unfolding

of a clause w.r.t. an atom which is not unifiable with any head in P, amounts to the deletion of that clause.

– **Folding** of goals $B_1, \ldots, B_n$ in the bodies of clauses $C_1, \ldots, C_n$, respectively, using definition clauses $D_1, \ldots, D_n$. Suppose that:
  1. $C_1, \ldots, C_n$ are clauses in P of the form: $H \leftarrow F, B_1, G, \ldots, H \leftarrow F, B_n, G$,
  2. $D_1, \ldots, D_n$ are definition clauses of the form: $newp(X_1, \ldots, X_m) \leftarrow Body_1, \ldots, newp(X_1, \ldots, X_m) \leftarrow Body_n$ and let these clauses be exactly those which have been introduced for the predicate $newp$ by an application of the definition rule,
  3. $B_1 = Body_1\theta, \ldots, B_n = Body_n\theta$, for some substitution θ, and
  4. for $i = 1, \ldots, n$, for every variable X in D_i and not in $\{X_1, \ldots, X_m\}$ we have that:
     - $X\theta$ is a variable which does not occur in (H, F, G) and
     - the variable $X\theta$ does not occur in the term $Y\theta$, for each variable Y occurring in $Body_i$ and different from X.

  Folding consists in replacing clauses $C_1, \ldots, C_n$ by the single clause $H \leftarrow F, newp(X_1, \ldots, X_m)\theta, G$.

– **Subsumption**. It consists in the deletion of a clause of P subsumed by another clause in P. (Recall that the clause $(H \leftarrow Body, A)\theta$ is subsumed by $H \leftarrow Body$.)

– **Case Split** of a clause C w.r.t. a substitution $\{X/u\}$. It consists in replacing a clause C of the form $H \leftarrow Body$, by the two mutually exclusive clauses: $(H \leftarrow Body)\ \{X/u\}$ and $H \leftarrow X \neq u,\ Body$, where X is a variable occurring in H.

Notice also that the set of clauses for $newp$ in the definition and folding rules is equivalent to the following formula: $newp(X_1, \ldots, X_m) \leftarrow Body_1 \vee \ldots \vee Body_n$ and in this sense we say that our rules allow us to specialize a program w.r.t. a disjunction of conjunctions of atoms. The following example clarifies how the folding rule works.

Example 1. Let us consider the two clauses C_1: $p(X) \leftarrow q(X), r(Y)$ and C_2: $p(X) \leftarrow q(X), s(X)$. They can be folded using the clauses D_1: $new(X,Y) \leftarrow r(Y)$ and D_2: $new(X,Y) \leftarrow s(X)$, thereby deriving clause C: $p(X) \leftarrow q(X), new(X,Y)$. Notice that by unfolding clause C using D_1 and D_2, we get again clauses C_1 and C_2. □

Now we propose a strategy for disjunctive partial deduction which directs the application of those enhanced transformation rules. This strategy, called $\mathcal{D}$, is depicted in Figure 4. It is an enhancement of the strategy $\mathcal{S}$ for standard partial deduction depicted in Figure 1.

In the strategy $\mathcal{D}$ we will allow ourselves to simplify inequalities in bodies of clauses. Every simplification consists in replacing conjunctions of inequalities by equivalent (w.r.t. Clark's equality theory) conjunctions of inequalities. Thus, for instance, (i) $X \neq a, X \neq a$ is replaced by $X \neq a$, (ii) $[\,] \neq [a|X]$ is replaced by

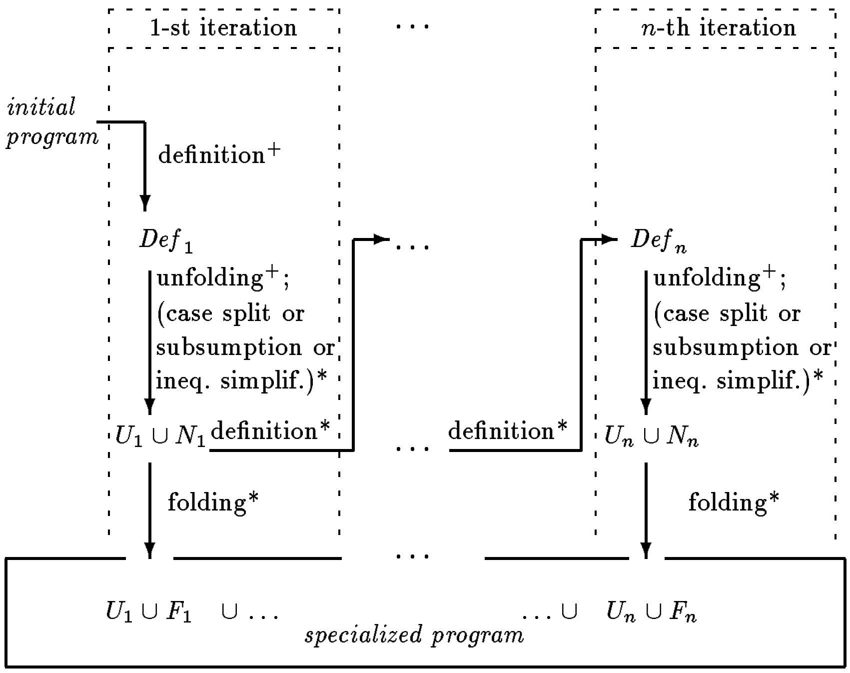

Fig. 4. The strategy $\mathcal{D}$ for disjunctive partial deduction.

true, and (iii) $a \neq a$ is replaced by *false*. Clauses with occurrences of *false* in the body are deleted.

In the strategy $\mathcal{D}$ we will also make use of the following concept of *packet* of clauses for a given predicate. The set of packets of clauses for a predicate, say *newp*, is a partition of the clauses for that predicate such that the following property holds: for any two clauses C_1 and C_2 for *newp* of the form $H_1 \leftarrow$ Ineqs_1, B_1 and $H_2 \leftarrow \mathit{Ineqs}_2, B_2$, respectively, where Ineqs_1 and Ineqs_2 are the conjunctions of all inequalities in their bodies, we have that C_1 and C_2 belong to the same packet iff $H_1 \leftarrow \mathit{Ineq}_1$ and $H_2 \leftarrow \mathit{Ineq}_2$ are equal, modulo variable renaming. We also say that two packets are *mutually exclusive* iff given a clause $H_1 \leftarrow \mathit{Ineqs}_1, B_1$ in the first packet and a clause $H_2 \leftarrow \mathit{Ineqs}_2, B_2$ in the second packet, we have that H_1 and H_2 are *either* not unifiable *or* unifiable via an mgu θ and at least one inequality in $(\mathit{Ineqs}_1\theta, \mathit{Ineqs}_2\theta)$ fails.

Strategy $\mathcal{D}$ for Disjunctive Partial Deduction. Given a program P and a goal G w.r.t. which we want to specialize P, we start off by introducing a set Def_1 consisting of a definition clause of the form $\mathit{newp}(X_1, \ldots, X_m) \leftarrow G$, where $X_1, \ldots, X_m$ are the variables occurring in G. Then, from the program $P \cup \mathit{Def}_1$ we construct a sequence of programs of the form $P \cup \mathit{Def}_i \cup Q_i$, for $i \geq 1$, with $Q_1 = \{\}$. We construct the program $P \cup \mathit{Def}_{i+1} \cup Q_{i+1}$ from the program $P \cup \mathit{Def}_i \cup Q_i$, during the i-th iteration of the strategy, as follows.

We unfold once or more times each clause in Def_i and then for each predicate *newp* occurring in the heads of the clauses derived by unfolding, we apply zero or more times the case split rule so to obtain a new set of clauses which can be partitioned into *mutually exclusive* packets for *newp*.

These applications of the case split rule are interleaved with applications of the subsumption rule and also with steps of simplification of inequalities, whenever possible. Notice that after the applications of the case split and subsumption rules, packets containing unit clauses are singletons.

Let U_i be the set of unit clauses and N_i be the set of packets each of them containing non-unit clauses only, derived from Def_i by the steps of unfolding, case split, subsumption, and simplification we have described. We then introduce a (possibly empty) set Def_{i+1} of new definition clauses such that each packet of clauses in N_i can be folded using clauses in $Def_1 \cup \ldots \cup Def_{i+1}$ producing exactly one clause which has no occurrences of predicates in common with P, with the possible exception of '$\neq$'. Each new predicate is defined by one or more clauses with one or more atoms in their bodies, because we use our enhanced definition rule. We fold the clauses in N_i using the definition clauses belonging to $Def_1 \cup \ldots \cup Def_{i+1}$ and we derive a set F_i of mutually exclusive clauses and a new program $P \cup Def_{i+1} \cup Q_{i+1}$, where Q_{i+1} is $U_i \cup F_i \cup Q_i$.

Analogously to the strategy $\mathcal{S}$, disjunctive partial deduction terminates at the end of the n-th iteration for which it is possible to fold all clauses in N_n using the definition clauses belonging to $Def_1 \cup \ldots \cup Def_n$. Thus, we have that Def_{n+1} is empty, because no new definition is needed for folding the goals in the bodies of N_n. The final specialized program is $P \cup Q_{n+1}$, and P can be dropped if we are interested only in goals relative to predicates defined by Def_1. □

Notice that, in the above strategy $\mathcal{D}$ we apply the case split and the subsumption rules so to derive clauses which have mutually exclusive heads. This requirement may be somewhat restrictive when we have to deal with predicates with several arguments. However, our strategy can easily be extended so that one may derive clauses which are mutually exclusive w.r.t. a chosen *subset* of the arguments of their heads.

Like the strategy $\mathcal{S}$ also the strategy $\mathcal{D}$ is parametric in the sense that it takes as parameters subsidiary strategies for directing the application of each enhanced transformation rule. In Section 4.1 we will see some examples of these subsidiary strategies.

As it will be clear also from the examples given in Section 4 our enhanced definition and folding rules allow us to achieve the effects of other transformation strategies, such as *conjunctive partial deduction* [10] for combining recursive computations, *argument filtering* [12] for eliminating unnecessary arguments, and *renaming* [6] for structure specialization.

Moreover, the combination of the case split rule which realizes a form of reasoning by cases, and the folding rule which allows us to fold several clauses at a time and in particular all clauses belonging to every packet, has the effect of producing specialized programs which are deterministic, from possibly non-deterministic initial programs. The concept of determinism we use, is related to the operational semantics and it corresponds to the fact that at runtime, there exists at most one clause whose head unifies with the current goal and whose body does not fail because of an unsatisfiable inequality.

3.2 Mechanization of the Strategy for Disjunctive Partial Deduction

In order to fully mechanize our strategy for disjunctive partial deduction, we need subsidiary strategies for directing the application of every transformation rule.

In particular, we need a strategy to select the clauses and the atoms w.r.t. which the unfolding rule has to be applied, and since that rule may, in general, be applied infinitely often, that strategy should also indicate when to stop unfolding. For instance, one may use the unfolding strategies which have been proposed for standard partial deduction [6] or for more general transformation techniques [15].

For the definition rule, we need a suitable strategy for suggesting the introduction of generalized definition clauses, because in order to fold the clauses derived by unfolding and case split, some subsets of the bodies of these clauses should be instances of the bodies of the new definition clauses to be introduced. Similarly to the standard partial deduction strategy $\mathcal{S}$, the termination of the disjunctive partial deduction strategy $\mathcal{D}$ strongly depends on the chosen generalized definition clauses one introduces. We will not investigate further this issue, which moreover, is not relevant in the examples we will give in Section 4.

3.3 On the Semantics Preserved During Partial Deduction

Without further restrictions, our rules are only *partially correct* w.r.t. the least Herbrand model semantics, that is, if program P_1 is transformed into program P_2 and the set D of new definition clauses is introduced during this transformation, then $M(P_2) \subseteq M(P_1 \cup D)$. Here we assume that the least Herbrand model of the $\neq$ predicate is the set $\{t_1 \neq t_2 \mid t_1$ and t_2 are different ground terms$\}$. However, it can easily be shown that, if the rules are applied according to our strategy for standard and disjunctive partial deduction, then they are *totally correct* w.r.t. the least Herbrand model semantics, that is, $M(P_2) = M(P_1 \cup D)$, because each clause is unfolded at least once before folding (see [7] for a similar condition).

As a consequence of the total correctness w.r.t. the least Herbrand model semantics, the success set semantics is preserved when we transform programs according to our strategy for disjunctive partial deduction. We assume here that the success set is computed using SLDNF-resolution where a possibly non-ground inequality succeeds iff the corresponding equality fails.

However, our transformation rules may affect the finite failure set semantics. In particular, the subsumption rule may eliminate infinite failures and folding may turn finite failures into infinite failures. While the elimination of infinite failures can be considered useful in many circumstances, conditions to prevent the introduction of infinite computations should be provided. To this purpose we may extend the conditions given in [22], where the preservation of the finite failure set is proved for a version of the definition and folding rules less general than ours. The preservation of other termination properties, such as left termination, when we use the enhanced rules we have introduced, is left for future studies. However, we believe that one can provide versions of our rules which preserve such properties, by extending the approaches proposed for Tamaki and Sato's rules in [1, 3, 18].

Finally, notice that for an efficient evaluation of the specialized programs, one may remove during a post-processing phase the inequalities which have been introduced by the application of the case split rule. This can be done by replacing inequalities in favour of cuts as specified by the following rule:

Cut Introduction [4]. It consists in replacing the two clauses:
$(H \leftarrow Body_1)\ \{X/u\}$ and $H \leftarrow X \neq u,\ Body_2$ by the new clauses:
$(H \leftarrow !,\ Body_1)\ \{X/u\}$ and $H \leftarrow Body_2$

where obviously, the resulting program requires a Prolog evaluator. In the above cut introduction rule we assume that the two clauses occur in the program in the order specified by the rule, but not necessarily in consecutive positions. We also assume that any duplicate consecutive occurrence of a cut can be deleted. The correctness of the cut introduction rule w.r.t. the success set semantics, computed by SLDNF-resolution with Prolog strategy, is straightforward if all calls of predicates are ground at runtime, because all clauses of the specialized program are mutually exclusive. This groundness condition which may be checked by standard abstract interpretation techniques, can be relaxed by considering groundness only w.r.t. to a subset of the arguments of each predicate.

4 Examples of Application of the Strategy for Disjunctive Partial Deduction

Now we present two examples of application of our strategy $\mathcal{D}$ for disjunctive partial deduction by which we show that one may overcome the limitations of standard partial deduction which we have mentioned in Section 2.3. In the first example it is shown that our strategy $\mathcal{D}$ is capable to pass the strong KMP test which is not passed by the standard strategy $\mathcal{S}$. In the second example an efficient parser for a given regular expression is automatically derived by partial deduction.

4.1 Strong KMP Test Example

Given the initial program depicted in Figure 5(A), in order to derive a specialized program w.r.t. the goal $naive\_match([a,a,b],S)$, we introduce by the definition rule the following clause:

1. $new1(S) \leftarrow naive\_match([a,a,b],S)$

We then apply in a fully automatic way the strategy $\mathcal{D}$ for disjunctive partial deduction by using the following *subsidiary strategies* for directing the application of every rule. In introducing these strategies we refer to the i-th iteration of

Initial matching program
for the strong KMP test

$$\left[\begin{array}{l} naive\_match(P,S) \leftarrow \\ \quad append(Left,P,X), \\ \quad append(X,Right,S) \\ append([\,],Y,Y) \leftarrow \\ append([C|X],Y,[C|Z]) \leftarrow \\ \quad append(X,Y,Z) \end{array}\right. \xrightarrow[\text{deduction}]{\text{by disjunctive partial}}$$

(A)

Specialized program
w.r.t. $naive\_match([a,a,b],S)$

$$\left[\begin{array}{l} 4.\ new1(S) \leftarrow new2(S) \\ 9.\ new2([a|S]) \leftarrow new3(S) \\ 10.\ new2([C|S]) \leftarrow C \neq a, new2(S) \\ 17.\ new3([a|S]) \leftarrow new4(S) \\ 18.\ new3([C|S]) \leftarrow C \neq a, new2(S) \\ 26.\ new4([b|S]) \leftarrow new5(S) \\ 27.\ new4([a|S]) \leftarrow new4(S) \\ 28.\ new4([C|S]) \leftarrow C \neq b, C \neq a, new2(S) \\ 29.\ new5(S) \leftarrow \end{array}\right.$$

(B)

Fig. 5. (A) Initial string matching program for the strong KMP test. (B) Specialized program derived by disjunctive partial deduction, where $new1(S)$ is defined by the clause $new1(S) \leftarrow naive\_match([a,a,b],S)$. It behaves like the deterministic finite automaton generated by the KMP string matching algorithm.

the strategy $\mathcal{D}$ and by 'the current program' we mean $P \cup Def_i \cup Q_i$, where P is the initial program. We also assume that at the i-th iteration Def_{i+1} is initially empty.

Unfolding strategy: for every clause in Def_i which has been introduced by the definition rule we unfold it once w.r.t. the leftmost atom in its body, whereby deriving the set V of clauses. Then starting from this set V, we apply as long as possible the following transformation which produces a new set of clauses from an old set of clauses:
"let C be a clause in the set at hand, and B be the leftmost atom, if any, in the body of C such that B unifies with the head of exactly one clause in the current program,
if there exists an atom in the body of C which does not unify with the head of any clause in the current program *then* we delete C *else if* B exists and the argument of the head of C is a variable *then* we replace C by the clauses obtained by unfolding C w.r.t. B".

This strategy is basically the 'determinate unfolding' considered in [6] with the condition that a determinate unfolding step is not performed if the argument of the head of the clause to be unfolded is not a variable.

Notice that this unfolding strategy might not terminate, and in order to avoid infinite loops, the stopping criterium must be strengthened. One may use, for instance, some available techniques for loop-checking (see [6] for a discussion on this issue).

Case split strategy: we apply case split as long as there exist two clauses whose heads H_1 and H_2 are not equal, also considering the inequalities in their bodies, and they are unifiable via an mgu which is not a variable renaming. Without loss of generality, we may assume that there exists a binding $\{X/u\}$ in the mgu of H_1 and H_2 such that $H_1\{X/u\}$ is not a variant of H_1. Then we apply the case split rule to the clause whose head is H_1 w.r.t. $\{X/u\}$ by considering the two cases: $X = u$ and $X \neq u$.

Definition and folding strategy: we consider every packet with non-unit clauses and we fold all clauses in the packet by using the set of definition clauses of a predicate in $Def_1 \cup \ldots \cup Def_i$, such that their bodies are exactly the bodies of the clauses in the packet without the inequalities. If such a predicate does not exist then we introduce it and we add its definition clauses to Def_{i+1}. Thus, no generalization is required in our case during an application of the definition rule.

For the subsumption rule we do not give any particular strategy because we will apply it whenever possible.

We now present the various iterations of the strategy $\mathcal{D}$ for the strong KMP test example. During the first few steps we also provide some comments on the derivation we have performed.

First iteration. By definition we introduce the clause:

1. $new1(S) \leftarrow naive\_match([a,a,b],S)$

Thus, $Def_1 = \{1\}$ and $Q_1 = \{\}$. By unfolding clause 1 we get:

2. $new1(S) \leftarrow append(Left,[a,a,b],X),\ append(X,Right,S)$

Clause 2 has not been further unfolded because each atom in its body unifies with more than one head in the current program (indeed, the predicate *append* has two clauses). For folding clause 2 we introduce the following definition clause:

3. $new2(S) \leftarrow append(Left,[a,a,b],X),\ append(X,Right,S)$

By folding clause 2 using clause 3 we get:

4. $new1(S) \leftarrow new2(S)$

Clause 4 is a clause of the specialized program to be derived. We have that $U_1 = \{\}$, $N_1 = \{2\}$, $F_1 = \{4\}$, $Def_2 = \{3\}$ and $Q_2 = \{4\}$. In the following iteration we deal with the new definition clause 3.

Second iteration. By unfolding clause 3 we get:

5. $new2([a|S]) \leftarrow append([a,b],Right,S)$
6. $new2([C|S]) \leftarrow append(Left,[a,a,b],X),\ append(X,Right,S)$

Clause 5 has not been further unfolded because $[a|S]$ is not a variable, and clause 6 has not been further unfolded because each atom in its body unifies with more than one head in the current program. Now the heads of clauses 5 and 6 are unifiable by the substitution $\{C/a\}$. Thus, by case split of clause 6 w.r.t. $\{C/a\}$ we get:

6.1 $new2([a|S]) \leftarrow append(Left,[a,a,b],X),\ append(X,Right,S)$
6.2 $new2([C|S]) \leftarrow C \neq a,\ append(Left,[a,a,b],X),\ append(X,Right,S)$

For folding clauses (5, 6.1) which belong to the same packet because they have equal heads, we introduce the following definition clauses:

7. $new3(S) \leftarrow append([a,b], Right, S)$
8. $new3(S) \leftarrow append(Left, [a,a,b], X),\ append(X, Right, S)$

By folding clauses (5, 6.1) using clauses (7, 8), and by folding clause 6.2 using clause 3, we get:

9. $new2([a|S]) \leftarrow new3(S)$
10. $new2([C|S]) \leftarrow C \neq a,\ new2(S)$

Clauses 9 and 10 are clauses of the specialized program to be derived. We have that $U_2 = \{\}$, $N_2 = \{5, 6.1, 6.2\}$, $F_2 = \{9, 10\}$, and $Def_3 = \{7, 8\}$, and $Q_3 = Q_2 \cup \{9, 10\}$. In the following iteration we deal with the new definition clauses 7 and 8.

Third iteration. By unfolding clauses 7 and 8 we get:

11. $new3([a|S]) \leftarrow append([b], Right, S)$
12. $new3([a|S]) \leftarrow append([a,b], Right, S)$
13. $new3([C|S]) \leftarrow append(Left, [a,a,b], X),\ append(X, Right, S)$

By case split of clause 13 w.r.t. the substitution $\{C/a\}$ we get:

13.1 $new3([a|S]) \leftarrow append(Left, [a,a,b], X),\ append(X, Right, S)$
13.2 $new3([C|S]) \leftarrow C \neq a,\ append(Left, [a,a,b], X),\ append(X, Right, S)$

For folding clauses (11, 12, 13.1) which belong to the same packet because they have equal heads, we introduce the following definition clauses:

14. $new4(S) \leftarrow append([b], Right, S)$
15. $new4(S) \leftarrow append([a,b], Right, S)$
16. $new4(S) \leftarrow append(Left, [a,a,b], X),\ append(X, Right, S)$

By folding clauses (11, 12, 13.1) using clauses (14, 15, 16), and by folding clause 13.2 using clause 3, we get:

17. $new3([a|S]) \leftarrow new4(S)$
18. $new3([C|S]) \leftarrow C \neq a,\ new2(S)$

Clauses 17 and 18 are clauses of the specialized program to be derived. We have that $U_3 = \{\}$, $N_3 = \{11, 12, 13.1, 13.2\}$, $F_3 = \{17, 18\}$, $Def_4 = \{14, 15, 16\}$, and $Q_4 = Q_3 \cup \{17, 18\}$. In the following iteration we deal with the new definition clauses 14, 15, and 16.

Fourth iteration. By unfolding clauses 14, 15, and 16 we get:

19. $new4([b|S]) \leftarrow append([\,], Right, S)$
20. $new4([a|S]) \leftarrow append([b], Right, S)$
21. $new4([a|S]) \leftarrow append([a,b], Right, S)$
22. $new4([C|S]) \leftarrow append(Left, [a,a,b], X),\ append(X, Right, S)$

By a case split step of clause 22 w.r.t. $\{C/b\}$ and then by a second case split step w.r.t. $\{C/a\}$ we get:

22.1 $new4([b|S]) \leftarrow append(Left, [a,a,b], X),\ append(X, Right, S)$
22.2 $new4([a|S]) \leftarrow b \neq a,\ append(Left, [a,a,b], X),\ append(X, Right, S)$

22.3 $new4([C|S]) \leftarrow C \neq a, C \neq b, append(Left, [a, a, b], X),\; append(X, Right, S)$

By simplification in the body of (22.2) we get:

23. $new4([a|S]) \leftarrow\; append(Left, [a, a, b], X),\; append(X, Right, S)$

For folding clauses (19, 22.1) we introduce the following definition clauses:

24. $new5(S) \leftarrow append([\,], Right, S)$
25. $new5(S) \leftarrow append(Left, [a, a, b], X), append(X, Right, S)$

By folding clauses (19, 22.1) using clauses (24, 25), by folding clauses (20, 21, 23) using clauses (14, 15, 16), and by folding clause 22.3 using clause 3, we get:

26. $new4([b|S]) \leftarrow new5(S)$
27. $new4([a|S]) \leftarrow new4(S)$
28. $new4([C|S]) \leftarrow C \neq b,\; C \neq a,\; new2(S)$

Clauses 26, 27, and 28 are clauses of the specialized program to be derived. We have that $U_4 = \{\}$, $N_4 = \{19, 20, 21, 22.1, 23, 22.3\}$, $F_4 = \{26, 27, 28\}$, $Def_5 = \{24, 25\}$, and $Q_5 = Q_4 \cup \{26, 27, 28\}$. In the following iteration we deal with the new definition clauses 24 and 25.

Fifth iteration. By unfolding clauses 24 and 25 we get:

29. $new5(S) \leftarrow$
30. $new5([a|S]) \leftarrow append([a, b], Right, S)$
31. $new5([C|S]) \leftarrow append(Left, [a, a, b], X),\; append(X, Right, S)$

We discard clauses 30 and 31 because they are subsumed by clause 29. We also have that $U_5 = \{29\}$, $N_5 = \{\}$, $F_5 = \{\}$, $Def_6 = \{\}$, and $Q_6 = Q_5 \cup \{29\}$.
After this fifth iteration we end the application of our strategy $\mathcal{D}$ for disjunctive partial deduction, because we did not introduce any extra definition clause for performing folding steps. The final program is depicted in Figure 5(B). It corresponds to a finite automaton which is deterministic, and in fact, clauses 9, 10, 17, 18, 26, 27, 28, and 29 correspond to the minimal finite automaton. However, in general, our strategy does not ensure the minimality of the derived finite automaton. We leave this problem for future research.

If in the final program we get rid of inequalities by using the cut introduction rule, we get:

$new1(S) \leftarrow new2(S)$
$new2([a|S]) \leftarrow !,\; new3(S)$
$new2([C|S]) \leftarrow new2(S)$
$new3([a|S]) \leftarrow !,\; new4(S)$
$new3([C|S]) \leftarrow new2(S)$
$new4([b|S]) \leftarrow !,\; new5(S)$
$new4([a|S]) \leftarrow !,\; new4(S)$
$new4([C|S]) \leftarrow new2(S)$
$new5(S) \leftarrow$

4.2 A Parsing Example

In this example we show the derivation of an efficient parser for a given regular expression from a general parser by using the disjunctive partial deduction technique. The initial program, which is a general parser for regular expressions over the alphabet $\{a, b\}$, is the one given in Figure 6(A), where $accepts(E, X, Y)$ holds iff the word represented by the difference list $X \setminus Y$ is in the language denoted by the regular expression E. In particular, $accepts(E, X, [\,])$ means that E accepts X.

Initial program for parsing regular expressions		Specialized program w.r.t. $accepts(a{\cdot}a^*{\cdot}b + a{\cdot}a^*, X, [\,])$
$accepts(a, [a\|X], X) \leftarrow$		6. $new1([a\|X]) \leftarrow new2(X)$
$accepts(b, [b\|X], X) \leftarrow$	by	8. $new2([\,]) \leftarrow$
$accepts(E1{\cdot}E2, S, X) \leftarrow$	disjunctive	14. $new2([a\|X]) \leftarrow new3(X)$
$\quad accepts(E1, S, X1),$	partial	15. $new2(X) \leftarrow X \neq [a\|Y], X \neq [\,], new4(X)$
$\quad accepts(E2, X1, X)$	deduction $\longrightarrow$	16. $new3([\,]) \leftarrow$
$accepts(E1 + E2, S, X) \leftarrow$		22. $new3([a\|X]) \leftarrow new3(X)$
$\quad accepts(E1, S, X)$		23. $new3(X) \leftarrow X \neq [a\|Y], X \neq [\,], new4(X)$
$accepts(E1 + E2, S, X) \leftarrow$		19. $new4([b]) \leftarrow$
$\quad accepts(E2, S, X)$		24. $new4([a\|X]) \leftarrow new4(X)$
$accepts(E^*, S, S) \leftarrow$		
$accepts(E^*, S, X) \leftarrow$		
$\quad accepts(E{\cdot}E^*, S, X)$		
(A)		(B)

Fig. 6. (A) General parser for regular expressions. (B) Specialized program derived by disjunctive partial deduction, where $new1(X)$ is defined by the clause $new1(X) \leftarrow accepts(a{\cdot}a^*{\cdot}b + a{\cdot}a^*, X, [\,])$.

We use our strategy $\mathcal{D}$ to derive a deterministic program which tests whether or not a word X is in the language denoted by the regular expression $a{\cdot}a^*{\cdot}b + a{\cdot}a^*$. Thus, we begin by introducing the new definition clause:

1. $new1(X) \leftarrow accepts(a{\cdot}a^*{\cdot}b + a{\cdot}a^*, X, [\,])$

We use the same subsidiary strategies for unfolding, case split, definition, and folding which we have used in the example of the strong KMP test. The various iterations of the strategy $\mathcal{D}$ are given below.

First iteration. By definition we introduce the clause:

1. $new1(X) \leftarrow accepts(a{\cdot}a^*{\cdot}b + a{\cdot}a^*, X, [\,])$

Thus, $Def_1 = \{1\}$ and $Q_1 = \{\}$. By unfolding clause 1 we get:

2. $new1([a|X]) \leftarrow accepts(a^*{\cdot}b, X, [\,])$

3. $new1([a|X]) \leftarrow accepts(a^*, X, [\,])$

For folding clauses (2, 3) we introduce the following definition clauses:

4. $new2(X) \leftarrow accepts(a^* \cdot b, X, [\,])$
5. $new2(X) \leftarrow accepts(a^*, X, [\,])$

By folding clauses (2, 3) using clauses (4, 5) we get:

6. $new1([a|X]) \leftarrow new2(X)$

Clause 6 is a clause of the specialized program to be derived. We have that $U_1 = \{\}$, $N_1 = \{2, 3\}$, $F_1 = \{6\}$, $Def_2 = \{4, 5\}$, and $Q_2 = \{6\}$. In the following iteration we deal with the new definition clauses 4 and 5.
Second iteration. By unfolding clauses 4 and 5 we get:

7. $new2(X) \leftarrow accepts(a^*, X, [b])$
8. $new2([\,]) \leftarrow$
9. $new2([a|X]) \leftarrow accepts(a^*, X, [\,])$

By a case split step of clause 7 w.r.t. $\{X/[\,]\}$ and then by a second case split step w.r.t. $\{X/[a|Y]\}$ we get:

7.1 $new2([\,]) \leftarrow accepts(a^*, [\,], [b])$
7.2 $new2([a|X]) \leftarrow [a|X] \neq [\,],\ accepts(a^*, [a|X], [b])$
7.3 $new2(X) \leftarrow X \neq [a|Y],\ X \neq [\,],\ accepts(a^*, X, [b])$

We delete clause 7.1 because it is subsumed by clause 8. Then, by simplification in the body of clause 7.2, we get:

10. $new2([a|X]) \leftarrow\ accepts(a^*, [a|X], [b])$

For folding clauses (9, 10) and clause (7.3) we introduce the following definition clauses:

11. $new3(X) \leftarrow accepts(a^*, X, [\,])$
12. $new3(X) \leftarrow accepts(a^*, [a|X], [b])$
13. $new4(X) \leftarrow accepts(a^*, X, [b])$

By folding clauses (9, 10) using clauses (11, 12) and by folding clause 7.3 using clause 13, we get:

14. $new2([a|X]) \leftarrow new3(X)$
15. $new2(X) \leftarrow X \neq [a|Y],\ X \neq [\,],\ new4(X)$

Clauses 14 and 15 are clauses of the specialized program to be derived. We have that $U_2 = \{8\}$, $N_2 = \{7.3, 9, 10\}$, $F_2 = \{14, 15\}$, $Def_3 = \{11, 12, 13\}$, and $Q_3 = Q_2 \cup \{8, 14, 15\}$. In the following iteration we deal with the new definition clauses 11, 12, and 13.
Third iteration. By unfolding clauses 11, 12, and 13 we get:

16. $new3([\,]) \leftarrow$
17. $new3([a|X]) \leftarrow accepts(a^*, X, [\,])$
18. $new3(X) \leftarrow accepts(a^*, X, [b])$

19. $new4([b]) \leftarrow$
20. $new4([a|X]) \leftarrow accepts(a^*, X, [b])$

By a case split step of clause 18 w.r.t. $\{X/[\,]\}$ and then by a second case split step w.r.t. $\{X/[a|Y]\}$ we get:

18.1 $new3([\,]) \leftarrow accepts(a^*, [\,], [b])$
18.2 $new3([a|X]) \leftarrow [a|X] \neq [\,],\ accepts(a^*, [a|X], [b])$
18.3 $new3(X) \leftarrow X \neq [a|Y],\ X \neq [\,],\ accepts(a^*, X, [b])$

We delete clause 18.1 because it is subsumed by clause 16. Then, by simplification in the body of clause 18.2, we get:

21. $new3([a|X]) \leftarrow accepts(a^*, [a|X], [b])$

By folding clauses (17, 21) using clauses (11, 12), by folding clause 18.3 using clause 13, and by folding clause 20 using clause 13, we get:

22. $new3([a|X]) \leftarrow new3(X)$
23. $new3(X) \leftarrow X \neq [a|Y],\ X \neq [\,],\ new4(X)$
24. $new4([a|X]) \leftarrow new4(X)$

Since no new definition clause has been introduced, the partial deduction process is ended. We have that $U_3 = \{16, 19\}$, $N_3 = \{17, 18.3, 20, 21\}$, $F_3 = \{22, 23, 24\}$, $Def_4 = \{\}$, and $Q_4 = Q_3 \cup \{16, 19, 22, 23, 24\}$. The specialized final program is depicted in Figure 6(B). Also in this final program we could replace the inequalities in favour of cuts. We leave this task to the reader.

Notice that if one uses the strategy for standard partial deduction, instead of the one for the disjunctive partial deduction, it is impossible to derive a final program which is deterministic, because by the atomic folding rule one cannot generate a single clause by folding two different clauses.

5 Discussion and Related Work

We have identified some transformation rules which allow us to derive very efficient specialized programs when in order to propagate some given information about the input data, it is necessary to: (i) deal with conjunctions of recursively defined predicates, (ii) combine alternative computations, and (iii) take into account unification failures. Although these rules are similar to the transformation rules already considered in the literature (see [15] for a survey), their usefulness for the specialization of logic programs has not been yet fully investigated in previous work.

We have proposed a way for combining our transformation rules into a strategy (see Figure 4) for disjunctive partial deduction which is parametric w.r.t. the subsidiary strategies to be used for applying the various transformation rules. Thus, in particular, any unfolding strategy and any strategy for the definition rule, also called the generalization strategy, already developed for program specialization (see, for instance, [6, 11, 20]) may be adapted to our strategy.

The particular unfolding strategy we apply is different from the one in [16] and it works both for the derivation of a program which behaves like the deterministic finite automaton generated by the KMP string matching algorithm, and for the derivation of regular expression parsers. These two program derivations which are fully presented here do not appear in [16]. With respect to that paper, here (i) in Sections 2.1 and 2.2 we have presented a detailed description of standard partial deduction in terms of unfold/fold rules and strategies; (ii) we have explained in Section 2.3 the reasons why standard partial deduction fails to produce a program which behaves like the finite automaton generated by the KMP matching algorithm; and also (iii) we have introduced in Section 3 the cut introduction rule. This rule is used for replacing negative equality tests and deriving very efficient residual programs.

The strategy we presented here is an extension of the one proposed for partial deduction in [6] (see Figure 1). It may also be used for deriving deterministic programs from nondeterministic ones and for avoiding intermediate data structures through the elimination of unnecessary variables [19].

A proposal for integrating partial deduction and the elimination of unnecessary variables has been presented in [10]. Indeed, that work which introduces the conjunctive partial deduction technique, recasts in the style of Lloyd and Shepherdson's approach [6, 13] some known ideas for the elimination of intermediate data structures based on unfolding/folding rules. However, by conjunctive partial deduction it is impossible to combine partial evaluations of alternative branches of SLD-trees, and it is also impossible to introduce inequalities and to simulate the application of the case split rule.

Some issues concerning the program transformation technique we have described here, are further developed in [16] where we introduce *modes* to be used for (i) ensuring that the derived programs are *semi-deterministic*, in the sense that if one considers also the inequalities in the bodies of clauses, at runtime at most one non-unit clause is unifiable with the current goal; (ii) guiding the applications of the case split rule so that case analysis is performed w.r.t. the so called *input arguments* only (they, in general, are a subset of all arguments); (iii) providing an unfolding strategy according to a *producer-consumer* model (only calls with instantiated input arguments are unfolded); and (iv) guiding the generation of the new eureka definitions required for folding.

In [16] we also prove that the transformation rules preserve the given mode information, so that no mode analysis is needed during the transformation process itself. We also use an extra transformation rule, called *head generalization*, whereby terms in the head of a clause are replaced by variables and the corresponding equalities are added to the body. Moreover, in [16] there is an example where it is shown that exponential improvements are possible and there is a fully automatic derivation of a specialized version of a multi-pattern matching program.

A transformation technique which is closely related to ours is supercompilation [8, 26]. Also by this technique it is possible to achieve program specialization and the elimination of intermediate data structures. One major difference with our work is that supercompilation basically deals with functional programs which are deterministic.

Now we want to make some final remarks about the examples of program specialization we have presented. As already mentioned, the known derivations of specialized the KMP matching programs by automatic methods based on partial deduction, are in a sense, less powerful than ours because we use an initial program which is very simple (see clause 1 in Section 4.1), instead of the initial program of Figure 2(A). A relevant exception is the one presented in [21]. In that paper the authors use the unfolding and folding rules together with a case split rule based on type information, in the sense that they assume that the arguments of the predicate *match* are lists. We do not use any type information at all. Moreover, the derivation in [21] is not guided by an automatic strategy. Very simple initial programs are considered also in [23], but in that paper the derivation requires a preliminary transformation into the so called 'failure continuation passing style'.

Similarly to the KMP example, also the published derivations by partial deduction of specialized parsers from general parsers, use rather clever deterministic initial programs (see, for instance, [14]).

Acknowledgments

We would like to thank D. De Schreye, J. Gallagher, R. Glück, M. Leuschel, B. Martens, M. H. Sørensen for stimulating discussions about logic program specialization. Experiments done using the SP specializer by J. Gallagher, were extremely useful to test the ability of the currently used techniques for partial deduction. We also thanks the referees for their useful suggestions and comments.

This work has been partly supported by the EC under the Human Capital and Mobility Project 'Logic Program Synthesis and Transformation', and also by the INTAS Project 93-1702, Murst 40% (Italy), and the Progetto Coordinato 'Programmazione Logica' of the CNR (Italy).

References

1. A. Bossi and N. Cocco. Preserving universal termination through unfold/fold. In *Proceedings ALP '94*, Lecture Notes in Computer Science 850, pages 269–286. Springer-Verlag, 1994.
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. J. Cook and J. P. Gallagher. A transformation system for definite programs based on termination analysis. In L. Fribourg and F. Turini, editors, *Proceedings of LOPSTR'94 and META'94, Pisa, Italy*, Lecture Notes in Computer Science 883, pages 51–68. Springer-Verlag, 1994.
4. Y. Deville. *Logic Programming: Systematic Program Development.* Addison-Wesley, 1990.
5. H. Fujita. An algorithm for partial evaluation with constraints. Technical Memorandum TM-0367, ICOT, Tokyo, Japan, 1987.

6. J. P. Gallagher. Tutorial on specialization of logic programs. In *Proceedings of ACM SIGPLAN Symposium on Partial Evaluation and Semantics Based Program Manipulation, PEPM '93, Copenhagen, Denmark*, pages 88–98. ACM Press, 1993.
7. M. Gergatsoulis and M. Katzouraki. Unfold/fold transformations for definite clause programs. In M. Hermenegildo and J. Penjam, editors, *Proceedings Sixth International Symposium on Programming Language Implementation and Logic Programming (PLILP '94)*, Lecture Notes in Computer Science 844, pages 340–354. Springer-Verlag, 1994.
8. R. Glück and A.V. Klimov. Occam's razor in metacomputation: the notion of a perfect process tree. In P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, editors, *3rd International Workshop on Static Analysis, Padova, Italy, September 1993*, Lecture Notes in Computer Science 724, pages 112–123. Springer-Verlag, 1993.
9. N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice Hall, 1993.
10. M. Leuschel, D. De Schreye, and A. de Waal. A conceptual embedding of folding into partial deduction: Towards a maximal integration. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming, Bonn, Germany*, pages 319–332. MIT Press, 1996.
11. M. Leuschel and B. Martens. Global control for partial deduction through characteristic atoms and global trees. Report CW 220, K.U. Leuven, Belgium, 1995.
12. M. Leuschel and M. H. Sørensen. Redundant argument filtering of logic programs. In J. Gallagher, editor, *Logic Program Synthesis and Transformation, Proceedings LOPSTR '96, Stockholm, Sweden. Report no. 96-020 Stockholm University.*, pages 63–77. Springer-Verlag, 1996. (To appear).
13. J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *Journal of Logic Programming*, 11:217–242, 1991.
14. T. Mogensen and A. Bondorf. Logimix: A self-applicable partial evaluator for Prolog. In K.-K. Lau and T. Clement, editors, *Logic Program Synthesis and Transformation, Proceedings LOPSTR '92, Manchester, U.K.*, Workshops in Computing, pages 214–227. Springer-Verlag, 1993.
15. A. Pettorossi and M. Proietti. Transformation of logic programs: Foundations and techniques. *Journal of Logic Programming*, 19,20:261–320, 1994.
16. A. Pettorossi, M. Proietti, and S. Renault. Reducing nondeterminism while specializing logic program. In *Proc. 24-th ACM Symposium on Principles of Programming Languages, Paris, France.* ACM Press, 1997. (To appear).
17. S. Prestwich. Online partial deduction of large programs. In *Proceedings ACM Sigplan Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM '93, Copenhagen, Denmark*, pages 111–118. ACM Press, 1993.
18. M. Proietti and A. Pettorossi. Semantics preserving transformation rules for Prolog. In *ACM Symposium on Partial Evaluation and Semantics Based Program Manipulation, PEPM '91, Yale University, New Haven, Connecticut, USA*, pages 274–284. ACM Press, 1991.
19. M. Proietti and A. Pettorossi. Unfolding-definition-folding, in this order, for avoiding unnecessary variables in logic programs. *Theoretical Computer Science*, 142(1):89–124, 1995.
20. D. Sahlin. Mixtus: An automatic partial evaluator for full Prolog. *New Generation Computing*, 12:7–51, 1993.
21. T. Sato and H. Tamaki. Examples of logic program transformation and synthesis. Case studies of transformation and synthesis of logic programs done up to Feb. 85, 1985.

22. H. Seki. Unfold/fold transformation of stratified programs. *Theoretical Computer Science*, 86:107–139, 1991.
23. D. A. Smith. Partial evaluation of pattern matching in constraint logic programming languages. In *Proceedings ACM Symposium on Partial Evaluation and Semantics Based Program Manipulation, PEPM '91, New Haven, CT, USA*, SIGPLAN Notices, 26, 9, pages 62–71. ACM Press, 1991.
24. M. H. Sørensen, R. Glück, and N. D. Jones. Towards unifying partial evaluation, deforestation, supercompilation, and GPC. In D. Sannella, editor, *Fifth European Symposium on Programming Languages and Systems, ESOP '94*, Lecture Notes in Computer Science 788, pages 485–500. Springer-Verlag, 1994.
25. H. Tamaki and T. Sato. Unfold/fold transformation of logic programs. In S.-Å. Tärnlund, editor, *Proceedings of the Second International Conference on Logic Programming, Uppsala, Sweden*, pages 127–138. Uppsala University, 1984.
26. V. F. Turchin. The concept of a supercompiler. *ACM TOPLAS*, 8(3):292–325, 1986.

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