Rules and Strategies for Contextual Specialization of Constraint Logic Programs*

Fabio Fioravanti[1], Alberto Pettorossi[2], Maurizio Proietti[1]
(1) IASI-CNR, Viale Manzoni 30, I-00185 Roma, Italy
(2) DISP, University of Roma Tor Vergata, I-00133 Roma, Italy
{fioravanti,adp,proietti}@iasi.rm.cnr.it

1 Introduction

The goal of *program specialization* is the automatic adaptation of a program to the context where it is used. Several program transformation techniques have been proposed in the literature for achieving program specialization. One of the most well known among these techniques is *partial evaluation* [9]. In this paper we consider *constraint logic programs* [7] over a domain $\mathcal{D}$ (CLP($\mathcal{D}$), for short). For this class of programs we assume that the context of use is given by a *constrained atom*, that is, a conjunction c, A, where c is a constraint over $\mathcal{D}$ and A is an atom.

We address the problem of *contextual specialization* w.r.t. a constrained atom which is defined as follows: Given a CLP($\mathcal{D}$) program P and a constrained atom c, A, we want to derive a program P_s and an atom A_s such that, for every variable assignment σ that assigns to each variable an element of $\mathcal{D}$, we have that:

$lm(P, \mathcal{D}) \models_\sigma c, A$ iff $lm(P_s, \mathcal{D}) \models_\sigma c, A_s$ (1: Contextual Specialization)

where, for any program P and domain $\mathcal{D}$, $lm(P, \mathcal{D})$ denotes the *least $\mathcal{D}$-model* of P [8].

Thus, contextual specialization is more general than standard specialization performed via partial evaluation [10, 20]. Indeed, in the case of partial evaluation, given a CLP($\mathcal{D}$) program P and a constrained atom c, A, we want to derive a program P_{pe} and an atom A_{pe} such that, for every variable assignment σ, we have that:

$lm(P, \mathcal{D}) \models_\sigma c, A$ iff $lm(P_{pe}, \mathcal{D}) \models_\sigma A_{pe}$ (2: Partial Evaluation)

Notice that every solution (P_{pe}, A_{pe}) of the partial evaluation problem is also a solution of the contextual specialization problem, that is, it satisfies the contextual specialization equivalence (1) when substituted for (P_s, A_s).

To illustrate the difference between contextual specialization of CLP programs and partial evaluation, let us consider the following CLP($\mathcal{R}$) program P over the domain $\mathcal{R}$ of real numbers:

$p(X) \leftarrow X \geq 0,\ q(X)$ (Program P)
$p(X) \leftarrow X \leq 0,\ r(X)$
$p(X) \leftarrow X \leq 3,\ u(X)$

where q, r, and u are predicates that do not depend on p.
By contextual specialization of P w.r.t. the constrained atom $X \geq 3, p(X)$ we derive the following program P_s:

1. $p_s(X) \leftarrow q(X)$ (Program P_s)
2. $p_s(X) \leftarrow X = 3,\ u(X)$

*In: M. Leuschel (ed.) Proceedings of the ICLP Workshop on Optimization and Implementation of Declarative Programming Languages (WOID'99), Las Cruces, New Mexico, USA, Dec.2-3, 1999.

together with the atom $p_s(X)$. On the other hand, by partial evaluation of P w.r.t. the same constrained atom $X \geq 3, p(X)$ we get the program P_{pe}:

3. $p_{pe}(X) \leftarrow X \geq 3,\ q(X)$ (Program P_{pe})
4. $p_{pe}(X) \leftarrow X = 3,\ u(X)$

together with the atom $p_{pe}(X)$.
Notice that in clause 1 of program P_s the constraint $X \geq 3$ does not occur while it occurs in clause 3 of program P_{pe}. In fact, for every variable assignment σ, $lm(P, \mathcal{R}) \models_\sigma X \geq 3, p(X)$ iff $lm(P_s, \mathcal{R}) \models_\sigma X \geq 3, p_s(X)$. In contrast, for every σ such that $\sigma(X) = 2$ and $lm(P, \mathcal{R}) \models_\sigma q(X)$, we have that: $lm(P, \mathcal{R}) \models_\sigma X \geq 3, p(X)$ iff $lm(P_s, \mathcal{R}) \models_\sigma p_s(X)$ does not hold. Thus, $(P_s, p_s(X))$ does *not* satisfy the partial evaluation equivalence (2) when substituted for (P_{pe}, A_{pe}).

The technique for specializing CLP programs which we present in this paper, follows the approach based on *transformation rules* and *strategies*. This approach has the advantage of separating the issue of proving the *correctness* of the program specialization process from that of guaranteeing the *efficiency* of the specialized programs.

The plan of the paper is as follows. In Section 2 we present a reconstruction of the partial evaluation of CLP programs by means of transformation rules and strategies, similarly to what has been done in [16] for the case of partial evaluation of definite logic programs. To this end, we consider variants of the *unfolding* and *folding* rules for CLP programs presented in [1, 4, 13].

In Section 3 we present an extra rule, called *contextual constraint replacement*, and an extra strategy, called *context propagation strategy*, which allow us to specialize a set of clauses defining a predicate, say p, with respect to a constraint when this is known to hold for all calls of p. As we will see, this contextual constraint replacement rule is indeed applied during the derivation of the specialized program P_s we have presented above.

In Section 4 we introduce the so called *invariant promotion strategy* for avoiding redundant checks of constraint satisfiability, and reducing the number of arguments of predicates. In Section 5 we present the related work and some concluding remarks.

2 Rules and Strategies for Partial Evaluation of CLP

We will introduce here a set of transformation rules and a strategy for partial evaluation of CLP programs over the domain $\mathcal{D}$. Our rules differ from those studied in previous papers [1, 4, 13] in that they are tailored to partial evaluation.

Throughout the paper we will use the following conventions. $FV(\varphi)$ denotes the set of the free variables of the formula φ. We will write $\forall X_1 \ldots X_m\, \psi$ also as $\forall X\, \psi$, where X denotes the set $\{X_1, \ldots, X_m\}$. Analogously for $\exists$, instead of $\forall$. Similarly, we will write the atom $p(t_1, \ldots, t_m)$ also as $p(t)$ where t denotes the sequence $t_1, \ldots, t_m$ of terms. Constraints over the domain $\mathcal{D}$ are denoted by the letters $c, d, \ldots$ (possibly with subscripts). Atoms are denoted by the letters $A, B, \ldots, H, \ldots$ (possibly with subscripts).

2.1 Transformation Rules for Partial Evaluation

The process of partially evaluating program P whereby deriving program P_s, can be formalized as a sequence $P_0, \ldots, P_n$ of programs, called a *transformation sequence*, where $P_0 = P$, $P_n = P_s$ and, for $k = 0, \ldots, n-1$, program P_{k+1} is obtained from program P_k by applying one of the following transformation rules: (R1) *constrained atomic definition*, (R2) *unfolding*, (R3) *constrained atomic folding*, (R4) *clause removal*, (R5) *constraint replacement*, and (R6) *equality introduction and equality elimination*.

R1. Constrained Atomic Definition. By *constrained atomic definition* (or *definition*, for short), we introduce the new clause

$$\delta: \quad newp(X_1, \ldots, X_h) \leftarrow c, p(t_1, \ldots, t_m)$$

which is said to be a *definition*, where: (i) *newp* is a predicate symbol not occurring in $P_0, \ldots, P_k$, (ii) $X_1, \ldots, X_h$ are distinct variables occurring in $c, p(t_1, \ldots, t_m)$, and (iii) p is a predicate symbol occurring in P_0. From program P_k we derive the new program $P_{k+1} = P_k \cup \{\delta\}$.

For $i \geq 0$, Def_i denotes the set of definitions introduced during the transformation sequence $P_0, \ldots, P_i$.

R2. Unfolding. Let $\gamma: \ A \leftarrow c, A_1, \ldots, A_i, \ldots, A_n$ be a renamed apart clause of P_k and let $\{B_j \leftarrow d_j, B_{j1}, \ldots, B_{jn_j} \mid j = 1, \ldots, m\}$ be the set of *all* clauses in P_k such that A_i and B_j have the same predicate symbol. For $j = 1, \ldots, m$, let us consider the clause

$$\gamma_j: \ A \leftarrow c,\ A_i{=}B_j,\ d_j,\ A_1, \ldots, A_{i-1}, B_{j1}, \ldots, B_{jn_j}, A_{i+1}, \ldots, A_n$$

where $A_i{=}B_j$ stands for the conjunction of the equalities between the corresponding arguments. Then, by *unfolding* clause γ w.r.t. the atom A_i, we derive the new program $P_{k+1} = (P_k - \{\gamma\}) \cup \{\gamma_j \mid j = 1, \ldots, m\}$.

R3. Constrained Atomic Folding. Let $\gamma: \ A \leftarrow c, d, A_1, \ldots, A_{i-1}, p(t), A_{i+1}, \ldots, A_n$ where t is a sequence of terms, be a clause of P_k. Let δ: $newp(X) \leftarrow d, p(t)$, where X is a sequence of distinct variables, be a variant of a clause in Def_k. Suppose that each variable in $FV(\delta) - X$ does not occur in $FV(A, c, A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n)$. By *folding* clause γ w.r.t. the constrained atom $d, p(t)$ by using δ, we derive the new clause

$$\varphi: \ A \leftarrow c, A_1, \ldots, A_{i-1}, newp(X), A_{i+1}, \ldots, A_n$$

and we derive the new program $P_{k+1} = (P_k - \{\gamma\}) \cup \{\varphi\}$.

R4. Clause Removal. Let $\gamma: \ A \leftarrow c, body$ be a clause of P_k. If $\mathcal{D} \models \forall X \neg c$, where $X = FV(c)$, then we say that the constraint c is *unsatisfiable* and we derive the new program $P_{k+1} = P_k - \{\gamma\}$.

R5. Constraint Replacement. Let $\gamma: \ A \leftarrow c_1, body$ be a renamed apart clause of P_k. Assume that $\mathcal{D} \models \forall X\, (\exists Y\, c_1 \leftrightarrow \exists Z\, c_2)$ for some constraint c_2, where $Y = FV(c_1) - FV(A, body)$, $Z = FV(c_2) - FV(A, body)$, and $X = FV(c_1, c_2) - (Y \cup Z)$. Then we derive the new program $P_{k+1} = (P_k - \{\gamma\}) \cup \{A \leftarrow c_2, body\}$.

R6. Equality Introduction and Equality Elimination. Let us consider the clauses $\gamma_1: \ A \leftarrow body_1$ and $\gamma_2: \ B \leftarrow (X = t), body_2$, such that X does not occur in t, and $(A \leftarrow body_1) = (B \leftarrow body_2)\{X/t\}$.
If $\gamma_1 \in P_k$, by *equality introduction* we derive γ_2 from γ_1, and we derive $P_{k+1} = (P_k - \{\gamma_1\}) \cup \{\gamma_2\}$.
If $\gamma_2 \in P_k$, by *equality elimination* we derive γ_1 from γ_2, and we derive $P_{k+1} = (P_k - \{\gamma_2\}) \cup \{\gamma_1\}$.

2.2 The Partial Evaluation Strategy

We now describe a general strategy for deriving efficient CLP programs by partial evaluation. Our strategy is a generalization of the strategies for the partial evaluation of definite logic programs using unfold/fold transformation rules presented in [6, 17, 19]. In our Partial Evaluation Strategy the unfolding steps are performed according to a slightly modified version of the *determinate unfolding* strategy proposed in [5].

Partial Evaluation Strategy
Input: A CLP($\mathcal{D}$) program P and a constrained atom c, A.
Output: A CLP($\mathcal{D}$) program P_{pe} and an atom A_{pe} of the form $p_{pe}(X_1, \ldots, X_h)$.

Initialization:
$Def := \{p_{pe}(X_1, \ldots, X_h) \leftarrow c, A\}$ where $\{X_1, \ldots, X_h\} = FV(c, A)$;
$NewDef := Def$;

while $NewDef \neq \{\}$ **do**

1. (UNFOLDING)
   Unfold each clause in $NewDef$ w.r.t. the leftmost atom in its body. Apply the clause removal rule and the constraint replacement rule. Let Γ be the resulting set of clauses derived from $NewDef$.
   while
   there is a clause γ in Γ such that by applying the unfolding rule w.r.t. an atom A in the body of γ, we get *at most one* clause whose constraint is satisfiable

do unfold γ w.r.t. A and then apply the clause removal, constraint replacement, equality introduction, and equality elimination rules. Let Γ_u be the resulting set of clauses derived from γ (Γ_u is either empty or a singleton). $\Gamma := (\Gamma - \{\gamma\}) \cup \Gamma_u$.

end

2. (DEFINITION)
   Assume that every clause γ in Γ is of the form: $H \leftarrow d_0, d_1, \ldots, d_k, A_1, \ldots, A_k$ (where for some $i = 0, \ldots, k$, the constraint d_i may be *true*).
   Apply the definition rule and introduce a set *NewDef* of new definitions, satisfying the following condition: for all $i = 0, \ldots, k$ there exists a clause $\delta :\ K \leftarrow d_i, A_i$ in $Def \cup NewDef$ such that we can fold γ using δ. *NewDef* may be empty.

   $Def := Def \cup NewDef$.
3. (FULL FOLDING)
   For each constrained atom d_i, A_i occurring in the body of a clause γ in Γ, perform a folding step by using a clause in Def.

end

The assumption at the beginning of Step 2 can be realized by suitable applications of the constraint replacement, equality introduction, and equality elimination rules at the end of Step 1 of our Partial Evaluation Strategy. These applications are nondeterministic, and different choices may lead to different applications of the definition and folding rules in Steps 2 and 3, respectively. To illustrate this point, let us assume that during Step 1 we have $\Gamma = \{\gamma\}$ and $Def = \{\delta_1, \delta_2\}$, where

γ: $r(X) \leftarrow X \geq 1, p(X), q(X)$

δ_1: $newp(X) \leftarrow X \geq 1, p(X)$

δ_2: $newq0(X) \leftarrow X \geq 0, q(X)$

Now we have, among others, the following alternative ways of applying our strategy:

1. (Step 1, continued) modify Γ by applying the constraint replacement rule to clause γ, thereby obtaining clause γ^*: $r(X) \leftarrow X \geq 1, X \geq 0, p(X), q(X)$,
   (Step 2) do nothing, and
   (Step 3) perform the full folding of γ^* by using δ_1 and δ_2, thereby obtaining
   $r(X) \leftarrow newp(X), newq0(X)$;

2. (Step 1, continued) derive γ^* as above,
   (Step 2) introduce the new definition δ_3: $newq(X) \leftarrow q(X)$, so that $NewDef = \{\delta_3\}$, and
   (Step 3) perform the full folding of γ^* by using δ_1 and δ_3, thereby obtaining
   $r(X) \leftarrow X \geq 0, newp(X), newq(X)$
   (when folding using clause δ_3 the constrained atom is: *true*, $q(X)$);

3. (Step 1, continued) do nothing,
   (Step 2) introduce the new definition δ_3: $newq(X) \leftarrow q(X)$, so that $NewDef = \{\delta_3\}$, and
   (Step 3) perform the full folding of γ by using δ_1 and δ_3, thereby obtaining
   $r(X) \leftarrow newp(X), newq(X)$.

We will not discuss here how to choose among different alternative ways of applying our strategy in order to derive more efficient residual programs.

Theorem 1. (Correctness of the Partial Evaluation Strategy). Let P be a CLP($\mathcal{D}$) program, and c, A be a constrained atom. Let us consider a transformation sequence $P, \ldots, P_{pe}$ constructed according to the Partial Evaluation Strategy. Let us also consider the atom $p_{pe}(X_1, \ldots, X_h)$ introduced during the Initialization phase of the Partial Evaluation Strategy. Then, for every variable assignment σ we have that:

$$lm(P, \mathcal{D}) \models_\sigma c, A \qquad \text{iff} \qquad lm(P_{pe}, \mathcal{D}) \models_\sigma p_{pe}(X_1, \ldots, X_h)$$

and thus, $lm(P, \mathcal{D}) \models_\sigma c, A$ iff $lm(P_{pe}, \mathcal{D}) \models_\sigma c, p_{pe}(X_1, \ldots, X_h)$.

Now we present an example of application of our Partial Evaluation Strategy. Let us consider the following CLP($\mathcal{R}$) program *SumMatch*:

summatch$(P, S) \leftarrow L1 = L2,\ R1 = R2,\ length(P, L1),\ sum(P, R1),$ (Program *SumMatch*)
$\quad prefix(G, S),\ length(G, L2),\ sum(G, R2)$
summatch$(P, [C|S]) \leftarrow C \geq 0,\ summatch(P, S)$
sum$([\,], A) \leftarrow A = 0$
sum$([A|B], C) \leftarrow A \geq 0,\ C \geq 0,\ C = A + D,\ sum(B, D)$
length$([\,], A) \leftarrow A = 0$
length$([A|B], C) \leftarrow A \geq 0,\ C > 0,\ C = D + 1,\ length(B, D)$
prefix$([\,], A) \leftarrow$
prefix$([A|B], [C|D]) \leftarrow A = C,\ prefix(B, D)$

Thus, *summatch*(P, S) holds iff there exists a substring G of S such that: (i) G and P have equal length, and (ii) the sum of the elements of G is equal to the sum of the elements of P. We want to partially evaluate the program *SumMatch* w.r.t. the constrained atom:

$$A + B + C \leq 5,\ A \geq 0,\ B \geq 0,\ C \geq 0,\ \ summatch([A, B, C], S)$$

In what follows we will write $k(A, B, C)$ to denote the constraint $A + B + C \leq 5,\ A \geq 0,\ B \geq 0,\ C \geq 0$. According to the Partial Evaluation Strategy we introduce the following definition:

$$summatch_{pe}(A, B, C, S) \leftarrow k(A, B, C),\ summatch([A, B, C], S).$$

The program derived by the Partial Evaluation Strategy is $SumMatch_{pe}$:

1. $summatch_{pe}(A, B, C, [D, E, F|S]) \leftarrow k(A, B, C),\ A + B + C = D + E + F,\ D \geq 0,\ E \geq 0,\ F \geq 0$
2. $summatch_{pe}(A, B, C, [D|S]) \leftarrow D \geq 0,\ summatch_{pe}(A, B, C, S)$.

3 Contextual Constraint Replacement

We now present an extra transformation rule which allows us to derive very efficient specialized programs by exploiting the knowledge of the contexts in which programs are used. As already mentioned, the notion of context is defined as a constrained atom. Below, in rule R7 and in Theorem 2, we will find useful to generalize that notion and consider a context to be defined as a *set* of constrained atoms.

R7. Contextual Constraint Replacement. Let γ be a renamed apart clause in P of the form $p(t) \leftarrow c_1, body$, where t is a sequence of terms. Let $\mathbf{C}$ be a set of constrained atoms. Suppose that for some constraint c_2 and for every $c, p(u)$ in $\mathbf{C}$,

$$\mathcal{D} \models \forall X\, ((c,\ u = t) \ \rightarrow\ (\exists Y\, c_1 \leftrightarrow \exists Z\, c_2))$$

where $Y = FV(c_1) - FV(t, body)$, $Z = FV(c_2) - FV(t, body)$, and $X = FV(c, u, t, c_1, c_2) - (Y \cup Z)$. Then, we derive program Q from program P by replacing clause γ by the clause $p(t) \leftarrow c_2, body$. In this case we say that Q has been derived from P by contextual constraint replacement w.r.t. $\mathbf{C}$.

This rule R7 is a generalization of rule R5.

Theorem 2. (Correctness of Contextual Constraint Replacement). Let us assume that from a CLP($\mathcal{D}$) program P we derive a program Q by contextual constraint replacement w.r.t. a set $\mathbf{C}$ of

constrained atoms. Assume also that $\mathbf{C}$ includes the set of all constrained atoms c, A such that $H \leftarrow c, A_1, \ldots, A_n$ is a clause of P and A is A_i for some $i = 1, \ldots, n$.
Then, for every constrained atom c, A in $\mathbf{C}$ and for every variable assignment σ we have that:

$$lm(P, \mathcal{D}) \models_\sigma c, A \qquad \text{iff} \qquad lm(Q, \mathcal{D}) \models_\sigma c, A.$$

An application of rule R7 allows us to derive program P_s from program P_{pe} (see Section 1) because, by the above Theorem 2, correctness is preserved for $\mathbf{C} = \{X \geq 3, p(X),\ X \geq 3, q(X),\ X = 3, u(X)\}$.

Now we give another example of application of rule R7. By applying rule R7 we will improve the program $SumMatch_{pe}$ derived by partial evaluation at the end of the previous section. By the correctness of the Partial Evaluation Strategy we have that, for every variable assignment σ,

$lm(SumMatch, \mathcal{D}) \models_\sigma k(A, B, C),\ summatch([A, B, C], S)$ iff
$lm(SumMatch_{pe}, \mathcal{D}) \models_\sigma k(A, B, C),\ summatch_{pe}(A, B, C, S)$.

Now we would like to apply rule R7 for performing contextual specialization of program $SumMatch_{pe}$ w.r.t. the constrained atom $k(A, B, C),\ summatch_{pe}(A, B, C, S)$, and in particular, in the body of clause 1 we would like to get rid of the constraint $k(A, B, C)$, that is, to replace it by $true$. Unfortunately, the application of rule R7 is not guaranteed to preserve correctness because the set $\mathbf{C}$ for $SumMatch_{pe}$ contains the constrained atom $D \geq 0,\ summatch_{pe}(A, B, C, S)$ and it is not the case that:

$$\mathcal{R} \models \forall A, B, C, D\ (D \geq 0\ \rightarrow\ (k(A, B, C) \leftrightarrow true)).$$

To avoid this difficulty, we apply the so called *Context Propagation Strategy* which works by exploiting the information of the context where the program is used. In our case we proceed as follows.
We introduce the following definition for the new predicate $summatch_c$:

$$summatch_c(A, B, C, S) \leftarrow k(A, B, C),\ summatch_{pe}(A, B, C, S)$$

By unfolding it w.r.t. $summatch_{pe}(A, B, C, S)$ we get (here and in the sequel we will omit to indicate the applications of the equality introduction and equality elimination rules):

3. $summatch_c(A, B, C, [D, E, F|S]) \leftarrow k(A, B, C),\ A+B+C = D+E+F,\ D \geq 0,\ E \geq 0,\ F \geq 0$
4. $summatch_c(A, B, C, [D|S]) \leftarrow k(A, B, C),\ D \geq 0,\ summatch_{pe}(A, B, C, S)$

Applying rule R5 we replace clause 4 by:

$$summatch_c(A, B, C, [D|S]) \leftarrow k(A, B, C),\ \underline{k(A, B, C)},\ D \geq 0,\ \underline{summatch_{pe}(A, B, C, S)}$$

This clause can be folded w.r.t. the underlined atoms, and we get:

5. $summatch_c(A, B, C, [D|S]) \leftarrow k(A, B, C),\ D \geq 0,\ summatch_c(A, B, C, S)$

In the derived program consisting of clauses 3 and 5, we can apply twice rule R7 with the objective of replacing $k(A, B, C)$ by $true$. These applications of rule R7 produce the following program $SumMatch_c$:

6. $summatch_c(A, B, C, [D, E, F|S]) \leftarrow A+B+C = D+E+F,\ D \geq 0,\ E \geq 0,\ F \geq 0$ (Program $SumMatch_c$)
7. $summatch_c(A, B, C, [D|S]) \leftarrow D \geq 0,\ summatch_c(A, B, C, S)$

While applying rule R7 correctness has been preserved because the set $\mathbf{C}$ includes for the predicate $summatch_c$ the following two constrained atoms only: (i) $k(A, B, C),\ summatch_c(A, B, C, S)$, and (ii) $k(A, B, C),\ D \geq 0,\ summatch_c(A, B, C, S)$, and for these two constrained atoms we have that:

(i) $\mathcal{R} \models \forall A, B, C, D\ (k(A, B, C)\ \rightarrow\ (k(A, B, C) \leftrightarrow true))$
(ii) $\mathcal{R} \models \forall A, B, C, D\ ((k(A, B, C),\ D \geq 0)\ \rightarrow\ (k(A, B, C) \leftrightarrow true))$.

4 Avoiding Redundant Constraints and Arguments

In this section we introduce a strategy, called *Invariant Promotion Strategy*, for avoiding redundant checks of constraint satisfiability and reducing the number of arguments of predicates. We will see this strategy in action by applying it to the program $SumMatch_c$ made out of clauses 6 and 7, which has been derived at the end of the previous section.

The improvement realized by our strategy consists in avoiding the recomputation of the expression $A+B+C$ depending on arguments whose value does not change during the evaluation of different calls of the predicate $summatch_c(A,B,C,S)$. In this sense we say that $A+B+C$ is an *invariant* for $summatch_c(A,B,C,S)$.

We start off by introducing the following definition:

8. $summatch_i(A,B,C,S) \leftarrow summatch_c(A,B,C,S)$

By constraint replacement, definition, and folding we derive from clause 8 the following two clauses (see Appendix B):

9. $summatch_i(A,B,C,S) \leftarrow I = A+B+C,\ newp(A,B,C,S,I)$
10. $newp(A,B,C,S,I) \leftarrow I = A+B+C,\ summatch_c(A,B,C,S)$

where in the variable I we store the value of the invariant expression $A+B+C$, and I is included among the arguments of *newp* for having the value of I at our disposal at every call of *newp*.

We now derive a recursive definition for the predicate *newp* as follows. By unfolding clause 10 w.r.t. $summatch_c(A,B,C,S)$, we get:

11. $newp(A,B,C,[D,E,F|S],I) \leftarrow I = A+B+C,\ A+B+C = D+E+F,\ D \geq 0,\ E \geq 0,\ F \geq 0$
12. $newp(A,B,C,[D|S],I) \leftarrow I = A+B+C,\ D \geq 0,\ summatch_c(A,B,C,S)$

By rule R5 we replace clauses 11 and 12 by the following ones:

13. $newp(A,B,C,[D,E,F|S],I) \leftarrow I = A+B+C,\ I = D+E+F,\ D \geq 0,\ E \geq 0,\ F \geq 0$
14. $newp(A,B,C,[D|S],I) \leftarrow I = A+B+C,\ \underline{I = A+B+C},\ D \geq 0,\ \underline{summatch_c(A,B,C,S)}$

By folding clause 14 w.r.t. the underlined atoms we get:

15. $newp(A,B,C,[D|S],I) \leftarrow I = A+B+C,\ D \geq 0,\ newp(A,B,C,S,I)$

Let us now consider the derived program for $summatch_i$ which consists of the clauses 9, 13, and 15. Since in the derived program, for the predicate *newp* the set $\mathbf{C}$ contains the two following constrained atoms only: (i) $I = A+B+C,\ newp(A,B,C,S,I)$, and (ii) $I = A+B+C,\ D \geq 0,\ newp(A,B,C,S,I)$, we may apply to clauses 13 and 15 the contextual clause replacement rule, and we get:

16. $newp(A,B,C,[D,E,F|S],I) \leftarrow I = D+E+F,\ D \geq 0,\ E \geq 0,\ F \geq 0$
17. $newp(A,B,C,[D|S],I) \leftarrow D \geq 0,\ newp(A,B,C,S,I)$

The program we have derived so far consists of the clauses 9, 16, and 17. By Theorems 1 and 2 we have that:

$lm(SumMatch_c, \mathcal{D}) \models summatch_c(A,B,C,S)$ iff $lm(\{9,16,17\}, \mathcal{D}) \models summatch_i(A,B,C,S)$.

Finally, we perform our last improvement by eliminating the first three arguments of the predicate $newp(A,B,C,S,I)$ because they do not change between any two calls of *newp*. The final program $SumMatch_i$ is as follows:

9.1 $summatch_i(A,B,C,S) \leftarrow I = A+B+C,\ newp1(S,I)$ (Program $SumMatch_i$)
16.1 $newp1([D,E,F|S],I) \leftarrow I = D+E+F,\ D \geq 0,\ E \geq 0,\ F \geq 0$
17.1 $newp1([D|S],I) \leftarrow D \geq 0,\ newp1(S,I)$.

The property that the first three arguments of $newp(A,B,C,S,I)$ are indeed redundant, may be discovered by a suitable program analysis (such as the redundant argument filtering technique presented in [11]). This property may also be proved by showing the equivalence:

$lm(\{16,17,16.1,17.1\}, \mathcal{D}) \models \forall A,B,C,S,I\,(newp(A,B,C,S,I) \leftrightarrow newp1(S,I))$

which, in turn, may easily be shown by using the unfold/fold proof method [15].

5 Related Work and Conclusions

We have presented some transformation rules and strategies for the specialization of constraint logic programs by taking into account their context of use. Our method extends related techniques for the *partial evaluation* of logic programs because the context of use is specified by means of a constraint which may be satisfied by a possibly infinite set of values, instead of a single value.

Indeed, our specialization method belongs to a family of methods which can be used for the specialization of programs w.r.t. *properties* of their context of use. Among these methods we would like to mention: (i) the methods for specializing logic programs presented in [2, 14], (ii) the method for *multiple specialization* of logic programs based on abstract interpretation presented in [18], and (iii) the method for *parameterized partial evaluation* of functional programs presented in [3].

In [2, 14] the authors present very general approaches to the specialization of logic programs w.r.t. properties which, however, do not seem to be amenable to full mechanization. In particular, most of the basic transformation rules introduced in [2] cannot be easily automated because their applicability conditions depend on undecidable properties. In this paper we have considered the larger class of CLP programs and we have described the properties of their context of use by means of constraints, instead of predicates defined by logic programs. By doing so, we were able to consider much more effective transformation rules, whose applicability conditions are decidable, if we assume that the *constraint theory is decidable*, as it is often the case.

A more practical approach is presented in [18], where a technique for the implementation of multiple specialization is proposed. This technique performs global analysis based on abstract interpretation and produces a polyvariant specialized program which contains different, optimized versions of the same predicate for different uses.

The work in [3] is concerned with a strict first order functional programming language and the main tool for specifying a property of the context w.r.t. which a program has to be specialized, is based on abstract interpretation.

Our transformation rules are intended for use within the program specialization techniques and, in this respect, they are more specific than the transformation rules for CLP programs considered in [1, 4, 13]. Moreover, we have an extra rule, the *contextual constraint replacement rule*, which allows us to replace a constraint c occurring in the body of a clause, by another constraint which is equivalent to c in the given context. For instance, consider the CLP($\mathcal{R}$) program P_0:

$p(X) \leftarrow X \geq 0,\ q(X)$ (Program P_0)

where q is a predicate that does not depend on p. By contextual constraint replacement w.r.t. the constrained atom $X \geq 3, p(X)$, we may derive P_1:

$p(X) \leftarrow q(X)$ (Program P_1)

because $X \geq 0$ is true in the context where $X \geq 3$ is true. Notice that P_1 cannot be obtained by the usual unfold/fold rules, because P_0 and P_1 may have different least $\mathcal{R}$-models.

It should be noticed that the folding rule R3 could have been made more powerful by allowing non-atomic folding (i.e., more than one atom in the body of the definition δ) and multiple folding (i.e., more than one definition clause), but we presented it in the weaker form for showing that this form is powerful enough to achieve the efficiency improvements obtainable by contextual specialization.

Notice also that in our rules we assume that the comma operator is commutative. This assumption may indeed be too restrictive for the existing implementations of CLP languages. We leave this issue for future studies.

The general strategy we have proposed in Section 2 for partially evaluating CLP programs, extends various techniques based on the unfolding and folding rules for partially evaluating logic programs and constraint logic programs (see, for instance, [6, 17, 19]), and, indeed, our folding rule is more powerful than the ones considered in those papers.

In order to achieve various program improvements by exploiting the knowledge of the context in which predicates are called, we have illustrated two more strategies: (i) the *context propagation* strategy, and (ii) the *invariant promotion* strategy. These strategies make use of the contextual constraint replacement rule.

Finally, the papers [10, 20] report on some correctness results for *partial deduction*, that is, partial evaluation of logic programs based on Lloyd and Shepherdson's approach [12]. Apart from some technical issues that make partial deduction different from program specialization based on unfolding/folding rules, we would like to point out that the frameworks described in [10, 20] do not allow for contextual constraint replacement. Indeed, from the constrained atom $X \geq 3, p(X)$ and program P_0 above we derive the *resultant*:

$p(X) \leftarrow X \geq 3,\ q(X)$

where the constraint $X \geq 3$ has to be solved at evaluation time, even if we know that it holds at specialization time. In contrast to partial deduction, our approach is able to fully exploit the statically available information provided by the context of use of a program.

Appendix A.

The following table shows some execution times of the programs derived in the previous sections.

Programs	LP = 3	LP = 10
$SumMatch$	276000 ms	762000 ms
$SumMatch_{pe}$	380 ms	720 ms
$SumMatch_c$	170 ms	330 ms
$SumMatch_i$	150 ms	260 ms

LP indicates the length of the pattern to be found in a string of 1000 real numbers. The experiments were performed on a Pentium II 300 MHz running Linux and SICStus Prolog.

Appendix B.

Derivation of clauses 9 *and* 10 *from clause* 8.

Since $\mathcal{D} \models \forall A, B, C\ (\exists I\ (I = A+B+C) \leftrightarrow true)$, by constraint replacement from clause 8 we get:

8.1 $summatch_i(A, B, C, S) \leftarrow I = A+B+C,\ I = A+B+C,\ summatch_c(A, B, C, S)$

By definition we introduce the following clause:

10. $newp(A, B, C, S, I) \leftarrow I = A+B+C,\ summatch_c(A, B, C, S)$

By folding clause 8.1 by using clause 10, we get:

9. $summatch_i(A, B, C, S) \leftarrow I = A+B+C,\ newp(A, B, C, S, I)$.

References

[1] N. Bensaou and I. Guessarian. Transforming constraint logic programs. *Theoretical Computer Science*, 206:81–125, 1998.

[2] 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.

[3] C. Consel and S.C. Khoo. Parameterized partial evaluation. *ACM Transactions on Programming Languages and Systems*, 15(3):463–493, 1993.

[4] S. Etalle and M. Gabbrielli. Transformations of CLP modules. *Theoretical Computer Science*, 166:101–146, 1996.

[5] 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.

[6] T. J. Hickey and D. A. Smith. Towards the partial evaluation of CLP languages. In *Proceedings ACM Symposium on Partial Evaluation and Semantics Based Program Manipulation, PEPM '91, New Haven, CT, USA*, SIGPLAN Notices, 26, 9, pages 43–51. ACM Press, 1991.

[7] J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[8] J. Jaffar, M. Maher K. Marriott, and P. Stuckey. The semantics of constraint logic programming. *Journal of Logic Programming*, 37:1–46, 1998.

[9] N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice Hall, 1993.

[10] M. Leuschel and D. De Schreye. Constrained partial deduction. In B. Freitag F. Bry and D. Seipel, editors, *Proceedings of the 12th Workshop Logische Programmierung (WLP'97)*, pages 116–126, Munich, Germany, September 1997.

[11] 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*, Lecture Notes in Computer Science 1207, pages 83–103. Springer-Verlag, 1996.

[12] J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *Journal of Logic Programming*, 11:217–242, 1991.

[13] M. J. Maher. A transformation system for deductive database modules with perfect model semantics. *Theoretical Computer Science*, 110:377–403, 1993.

[14] A. Pettorossi and M. Proietti. A theory of logic program specialization and generalization for dealing with input data properties. In O. Danvy, R. Glück, and P. Thiemann, editors, *Proceedings of the Dagstuhl Seminar on Partial Evaluation*, Lecture Notes in Computer Science 1110, pages 386–408. Springer-Verlag, 1996.

[15] A. Pettorossi and M. Proietti. Synthesis and transformation of logic programs using unfold/fold proofs. *Journal of Logic Programming*, 41(2&3):197–230, 1999.

[16] A. Pettorossi, M. Proietti, and S. Renault. Reducing nondeterminism while specializing logic programs. In *Proc. 24-th ACM Symposium on Principles of Programming Languages, Paris, France*, pages 414–427. ACM Press, 1997.

[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] G. Puebla and M. Hermenegildo. Abstract multiple specialization and its application to program parallelization. *J. of Logic Programming. Special Issue on Synthesis, Transformation and Analysis of Logic Programs*, 41(2&3):279–316, November 1999.

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

[20] A. Wrzos-Kaminska. Partial evaluation in constraint logic programming. In Z.W. Ras and M. Michalewicz, editors, *Proceedings of the 9th International Symposium on Foundations of Intelligent Systems*, Lecture Notes in Computer Science 1079, pages 98–1, Zakopane, Poland, 1996. Springer-Verlag.