Automated Strategies for Specializing 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

Program specialization is a powerful methodology for automated software engineering which allows for program reuse. Program specialization, in fact, may adapt a generic program to the particular context where it has to be used and this adaptation process may be done via automatic or semi-automatic techniques. One such technique is *partial evaluation*. Partial evaluation has been proposed and studied for various programming languages [13] and it has been shown to be very powerful in the sense that many sophisticated program optimizations can be performed by taking advantage of the context of use via partial evaluation. In this paper we address the problem of automating some techniques for the *contextual specialization* [9] of *constraint logic programs* over a domain $\mathcal{D}$ (CLP($\mathcal{D}$), for short) [11]. Contextual specialization 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 an element of $\mathcal{D}$ to each free variable in the constrained atom c, A, we have that:

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

where, $lm(P,\mathcal{D})$ denotes the *least* $\mathcal{D}$*-model* of P [12].

Thus, contextual specialization is more general than the partial evaluation methods [16, 25] of CLP($\mathcal{D}$) programs which are based on Lloyd and Shepherdson's approach [17]. Indeed, in the case of partial evaluation, given a CLP($\mathcal{D}$) program P and a constrained atom c, A, we derive a program P_{pe} and an atom A_{pe} such that, for every variable assignment σ, we have that:

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

The following example illustrates 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:

$$p(X) \leftarrow X \geq 0,\, q(X) \qquad \text{(Program } P\text{)}$$

* Extended Abstracts of LOPSTR 2000, Tenth International Workshop on Logic-based Program Synthesis and Transformation, 24-28 July 2000, London, UK. Technical Report Report Series, Department of Computer Science, University of Manchester, ISSN 1361-6161. Report number UMCS-00-6-1. http://www.cs.man.ac.uk/cstechrep/titles00.html

where q is a predicate which does not depend on p. By contextual specialization of P w.r.t. the constrained atom $X \geq 3, p(X)$ we derive the program P_s:

$p_s(X) \leftarrow q(X)$ (Program P_s)

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}:

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

together with the atom $p_{pe}(X)$.
Thus, the partially evaluated program P_{pe} is less efficient than the program P_s derived by contextual specialization because it checks whether or not the constraint $X \geq 3$ holds, even if we already know that it holds in the context where the program is used, which is indeed $X \geq 3$.

In [9] we have introduced some *transformation rules* and semi-automatic *strategies* for contextual specialization. Here we address various issues concerning the full automation of our strategies. In particular, we consider the problems of: (i) when and how to unfold, (ii) when and how to generalize, and (iii) when and how to apply the contextual specialization rule. Our automatic strategy for contextual specialization of CLP($\mathcal{D}$) programs is based on adaptations to our framework of notions used in the fields of constraint programming, partial evaluation, and abstract interpretation. In particular, we consider: (i) a suitable solver for simplifying constraints [11], (ii) two well-quasi-orders $\preceq_u$ and $\preceq_g$ between constrained goals, which are used, respectively, for ensuring the termination of the unfolding process and for activating the clause generalization process [14, 15, 24], and (iii) a *widening operator* [5] for ensuring the termination of that generalization process.

2 Rules for Contextual Specialization of CLP programs

In this section we introduce the set of transformation rules which we use for specializing CLP($\mathcal{D}$) programs. Our rules are slight modifications of the unfold/fold rules considered in [1, 8, 18] and they are specifically designed for performing contextual specialization.

Throughout the paper we will consider a fixed first-order language $\mathcal{L}$. We will use the following conventions and for notions not defined here the reader may refer to [11]. Let *Vars* be the (possibly infinite) set of variables in $\mathcal{L}$. Given a formula φ, the set of variables and the set of free variables of φ are denoted by $vars(\varphi)$ and $FV(\varphi)$, respectively. 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. We assume that the set $\mathcal{C}$ of constraints over $\mathcal{D}$ contains a given set of *atomic constraints* and this set of atomic constraints includes *true*, *false*, and the equalities between terms. We assume that $\mathcal{C}$ is closed w.r.t. conjunction (which we denote by comma) and existential quantification. The constraint *true* is the empty conjunction. *Atoms* are denoted by the letters $H, A, B, \ldots$, possibly with subscripts. A *constrained atom* is the conjunction of a constraint and an atom. *Goals* are conjunctions of atoms, and they are denoted by G, possibly with subscripts. A *constrained goal*, denoted by K, possibly with subscripts, is the conjunction of a constraint and a goal. Conjunction is commutative and, thus, the order of constraints and atoms is immaterial.

We assume that every clause $A_0 \leftarrow c, A_1, \ldots, A_n$ of a CLP program is in *canonical form*, that is: (1) for every $i = 0, \ldots, n$, A_i is of the form $p(X_1, \ldots, X_m)$, where $X_1, \ldots, X_m$ are distinct variables, and (2) all bound variables occurring in c are distinct. Similarly, we may assume that every constrained atom is in canonical form. We assume that, when a clause is renamed apart, each of its bound and free variables does not occur in other clauses.

Notice that any clause γ which is not in canonical form, can be transformed into one which is in canonical form and it is equivalent to γ w.r.t. the least $\mathcal{D}$-model semantics. This can be done by adding extra equality constraints. For example, the clause: $p(X+1) \leftarrow X \leq 0, r(X-1)$ can be transformed into the equivalent clause: $p(Y) \leftarrow Y = X+1,\ X \leq 0,\ Z = X-1,\ r(Z)$, which is in canonical form.

For a clause γ of the form: $H \leftarrow c, G$, we say that H is the head of γ, denoted $hd(\gamma)$, and c, G is the body of γ, denoted $bd(\gamma)$. Given a clause γ of the form: $H \leftarrow K_1, K_2$, for some constrained goals K_1 and K_2, the set of the *linking variables* of K_1 in γ is $FV(K_1) \cap FV(H, K_2)$.

The semantics of a CLP($\mathcal{D}$) program P is the *least $\mathcal{D}$-model* of P, denoted $lm(P, \mathcal{D})$ [12]. We assume that there exists a computable total function *solve*: $\mathcal{C} \times \mathcal{P}_{\mathit{fin}}(\mathit{Vars}) \rightarrow \mathcal{C}$ which is *sound* w.r.t. constraint equivalence, in the sense that, for every constraint c_1 and finite set X of variables, if $\mathit{solve}(c_1, X) = c_2$ then $FV(c_2) \subseteq (FV(c_1) \cap X)$ and $\mathcal{D} \models \forall X((\exists Y c_1) \leftrightarrow c_2)$ where $Y = FV(c_1) - X$.

Moreover, we require that *solve* is *complete* w.r.t. satisfiability in the sense that, for any constraint c and $Y = FV(c)$:
(i) $\mathit{solve}(c, \emptyset) = \mathit{true}$ if c is *satisfiable*, that is, $\mathcal{D} \models \exists Y\, c$, and
(ii) $\mathit{solve}(c, \emptyset) = \mathit{false}$ if c is *unsatisfiable*, that is, $\mathcal{D} \not\models \exists Y\, c$.

Since *solve* is sound w.r.t. constraint equivalence, in (i) and (ii) 'if' can be replaced by 'iff'. The completeness of the *solve* function w.r.t. satisfiability is only required to check the applicability of the Clause Removal rule R4 (see below). Notice that we need not make any assumption about the completeness of the *solve* function w.r.t. non-empty sets of variables.

The process of *specializing* 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*, and (R5) *contextual constraint replacement*. Rule R6 which we will present below, is an instance of rule R5.

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

$$\delta: \ \ newp(X_1, \ldots, X_h) \leftarrow c, A$$

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, A, and (iii) the predicate symbol of A occurs in P_0. From program P_k we derive the new program P_{k+1} which is $P_k \cup \{\delta\}$.
For $i \geq 0$, $Defs_i$ is 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 $\{H_j \leftarrow c_j, G_j \mid j = 1, \ldots, m\}$ be the set of *all* clauses in P_k such that A_i and H_j have the same predicate symbol. For $j = 1, \ldots, m$, let us consider the clause

$$\gamma_j: \ \ A \leftarrow c,\ A_i = H_j,\ c_j,\ A_1, \ldots, A_{i-1}, G_j, A_{i+1}, \ldots, A_n$$

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

R3. Constrained Atomic Folding. Let $\gamma: \ A \leftarrow c, d, A_1, \ldots, A_{i-1}, B,\ A_{i+1}, \ldots, A_n$ be a clause of P_k. Let $\delta: \ newp(X) \leftarrow d, B$, where X is a sequence of distinct variables, be a variant of a clause in $Defs_k$. 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, B by using δ, we derive the new clause

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

and from program P_k we derive the new program P_{k+1} which is $(P_k - \{\gamma\}) \cup \{\gamma_f\}$.

R4. Clause Removal. Let $\gamma: \ A \leftarrow c, G$ be a clause of P_k. If the constraint c is *unsatisfiable*, that is, $solve(c, \emptyset) = false$ then from program P_k we derive the new program P_{k+1} which is $P_k - \{\gamma\}$.

R5. Contextual Constraint Replacement. Let $\mathbf{C}$ be a set of constrained atoms. Let γ be a renamed apart clause in P_k of the form: $p(U) \leftarrow c_1, G$. Suppose that for some constraint c_2, we have that for every constrained atom $c, p(V)$ in $\mathbf{C}$,

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

where: (i) $Y = FV(c_1) - vars(U, G)$, (ii) $Z = FV(c_2) - vars(U, G)$, and (iii) $X = FV(c, U, V, c_1, c_2) - (Y \cup Z)$. Then, we derive program P_{k+1} from program P_k by replacing clause γ by the clause: $p(U) \leftarrow c_2, G$. In this case we say that P_{k+1} has been derived from P_k by contextual constraint replacement w.r.t. $\mathbf{C}$.

An instance of rule R5 for $\mathbf{C} = \{true, p(U)\}$, is the following rule:

R6. Constraint Replacement. Let $\gamma: \ A \leftarrow c_1, G$ be a renamed apart clause of P_k. Assume that $\mathcal{D} \models \forall X (\exists Y\, c_1 \leftrightarrow \exists Z\, c_2)$ where: (i) $Y = FV(c_1) - vars(A, G)$, (ii) $Z = FV(c_2) - vars(A, G)$, and (iii) $X = FV(c_1, c_2) - (Y \cup Z)$. Then from program P_k we derive the new program P_{k+1} which is $(P_k - \{\gamma\}) \cup \{A \leftarrow c_2, G\}$.

A possible value of the constraint $\exists Z\, c_2$ which one may use in this rule, is $solve(c_1, X)$ for any set X of variables which is a subset of $FV(c_1)$ (see the Contextual Specialization Strategy in Section 4). In the Contextual Specialization Strategy below, rule R6 is used either to replace a constraint in the body of a clause or to add a constraint to it.

3 Selection Functions, Well-Quasi-Orders, and Clause Generalization

In this section we introduce the notions of: (i) *unfolding selection function*, (ii) *well-quasi-orders* over constrained goals, and (iii) *clause generalization*, which are used in our program specialization strategy. These notions are extensions to the case of CLP($\mathcal{D}$) programs of similar notions considered in the case of partial evaluation of functional and logic programs (see for instance, [14, 15, 24]).

Definition 1. An *unfolding selection function* is a function U from constrained goals to atoms which is defined for every constrained goal with at least one atom. Given a clause γ, the atom $U(bd(\gamma))$ is said to be the *atom selected for unfolding*.

Definition 2. A *well-quasi-order* (*wqo*, for short) over the set of constrained goals is a reflexive, transitive binary relation $\preceq$ such that for every infinite sequence $\{K_i \mid i = 0, 1, \ldots\}$ of constrained goals there exist two natural numbers i and j such that $i < j$ and $K_i \preceq K_j$ [7]. Given two constrained goals K_1 and K_2, if $K_1 \preceq K_2$ we say that K_1 is *embedded in* K_2.

Various examples of wqo's that are used for ensuring the termination of the unfolding process during the partial evaluation of logic and functional programs, can be found in [14, 24]. For our specialization example of Section 5 we will use a very simple wqo $\preceq_L$ defined as follows: given two constrained goals K_1 and K_2, we have that $K_1 \preceq_L K_2$ iff the leftmost atom (in the textual order) in K_1 and the leftmost atom (in the textual order) in K_2 have the same predicate symbol.

Definition 3. [*Constraint Lattice and Clause Generalization*] Given the set $\mathcal{C}$ of constraints over $\mathcal{D}$, we consider the partial order $\langle \mathcal{C}, \sqsubseteq \rangle$ such that for any two constraints c_1 and c_2 in $\mathcal{C}$, $c_1 \sqsubseteq c_2$ iff $\mathcal{D} \models c_1 \rightarrow c_2$. We assume that $\langle \mathcal{C}, \sqsubseteq \rangle$ is a lattice, where: (i) the least element is *false*, (ii) the greatest element is *true*, (iii) the *least upper bound* of two constraints c_1 and c_2 is denoted by $c_1 \sqcup c_2$, and (iv) the *greatest lower bound* of two constraints c_1 and c_2 is their conjunction c_1, c_2. The *widening operator* ∇ is a binary operator between constraints such that:
(W1) $(c_1 \sqcup c_2) \sqsubseteq (c_1 \nabla c_2)$, and
(W2) for every infinite sequence $\{c_i \mid i = 0, 1, \ldots\}$ of constraints, the infinite sequence $\{d_i \mid i = 0, 1, \ldots\}$ of constraints where $d_0 = c_0$ and $d_{i+1} = d_i \nabla c_{i+1}$, stabilizes, that is, $\exists h \geq 0\ \forall k \geq h\ d_k = d_{k+1}$. Notice that, in general, ∇ is not commutative.
Given two clauses α: $new1(U) \leftarrow c_1, q(X)$ and δ: $new2(V) \leftarrow c_2, q(X)$, where U, $FV(c_1)$, V, and $FV(c_2)$, are subsets of X, we define the *generalization* of α

w.r.t. δ, denoted by $gen(\alpha,\delta)$, to be the clause: $genp(W) \leftarrow c_1\nabla c_2,\ q(X)$, where *genp* is a new predicate symbol and $W = U \cup V$.
We say that the generalization operator *gen preserves* the wqo $\preceq$ iff for every atom $q(X)$ and constraints c, c_1, and c_2, if $c_1, q(X) \preceq c, q(X)$ then $c_1\nabla c_2, q(X) \preceq c, q(X)$.

Example 1. [*Linear equations and inequations over the reals*] Let us consider the domain $\mathcal{R}_{lin}$ of conjunctions of linear equations ($=$) and inequations ($<, \leq, >, \geq$) over real numbers. $\mathcal{R}_{lin}$ is a lattice whose least upper bound operation is defined via the convex hull construction. Several widening operators for $\mathcal{R}_{lin}$ have been defined in the literature [5]. We consider here the following simple operator which is used in our program specialization example of Section 5 below. Given a constraint c, by $ineq(c)$ we denote the constraint obtained by replacing every equation $e_1 = e_2$ in c by the conjunction of inequations $e_1 \geq e_2,\ e_1 \leq e_2$. Given two constraints c and d, such that $ineq(c)$ is the conjunction of the atomic constraints $c_1, \ldots, c_n$, we define $c\nabla d$ to be the conjunction of all c_j, with $0 \leq j \leq n$, such that $d \sqsubseteq c_j$.

This widening operator satisfies Condition W1 because, by construction, $c \sqsubseteq c\nabla d$ and $d \sqsubseteq c\nabla d$, that is, $c\nabla d$ is an upper bound of c, d. Thus, we have that $(c \sqcup d) \sqsubseteq (c\nabla d)$. Moreover, Condition W2 is satisfied because each application of the widening operator starting from the constraint c does not increase the number of atomic constraints in $ineq(c)$.

For instance, given the following two clauses:

α. $mmod_s(I,J,M) \leftarrow I=0,\ J\geq 0,\ mmod(I,J,M)$
δ. $newp(I,J,M) \leftarrow I=1,\ J>0,\ mmod(I,J,M)$

$gen(\alpha,\delta)$ is $genp(I,J,M) \leftarrow I\geq 0,\ J\geq 0,\ mmod(I,J,M)$, because we have that: (i) $ineq(I=0,\ J\geq 0) = (I\geq 0,\ I\leq 0,\ J\geq 0)$ and (ii) $I\leq 0$ is dropped by our widening operator because it is not the case that $(I=1,\ J>0) \sqsubseteq I\leq 0$.

Notice that the widening operator defined in this example preserves the wqo $\preceq_L$ defined above. Indeed, for every atom $q(X)$ and constraints c, c_1, and c_2, we have that $c_1\nabla c_2,\ q(X) \preceq_L c,\ q(X)$. □

4 An Automated Strategy for Contextual Specialization

We now describe a general strategy for deriving efficient CLP($\mathcal{D}$) programs by specialization. Our strategy is a generalization of the strategies for the partial evaluation of definite logic programs using unfold/fold transformation rules presented in [10, 21, 23]. Our strategy is parameterized by: (1) the function *solve* which is used for the application of the constraint replacement rule, (2) an unfolding selection function U for selecting the atom w.r.t. which a clause should be unfolded, (3) a well-quasi-order $\preceq_u$ over constrained goals which tells us when to terminate the unfolding process, (4) a clause generalization operator *gen*, with its associated widening operator ∇, and (5) a well-quasi-order $\preceq_g$ over constrained atoms which tells us when to activate the clause generalization process.

In the Contextual Specialization Strategy presented below, we use the set *Defs* of all clauses introduced by the definition rule during program specialization. This set of clauses is arranged as a tree, called *Defstree*. We also consider a tree $Utree(\delta)$ for each clause δ in *Defs*: the root of $Utree(\delta)$ is δ itself, and the nodes of $Utree(\delta)$ are the clauses derived from δ by applying the unfolding and constraint replacement rules. The usual relation of *ancestor* between nodes in a tree gives us the relation of ancestor between clauses in *Defstree* and also in $Utree(\delta)$ for each clause δ in *Defs*.

In the Input to the Contextual Specialization Strategy below, the condition $FV(c) \subseteq X$ is not actually a restriction, because our constraints are closed w.r.t. existential quantification.

Contextual Specialization Strategy
Input: (i) A CLP($\mathcal{D}$) program P, (ii) a constrained atom $c, p(X)$ such that $FV(c) \subseteq X$, (iii) an unfolding selection function U, (iv) a well-quasi-order $\preceq_u$ over constrained goals, (v) a well-quasi-order $\preceq_g$ over constrained atoms, and (vi) a clause generalization operator *gen* which preserves $\preceq_g$.
Output: A CLP($\mathcal{D}$) program P_s and an atom $p_s(X)$.

Initialization:
Let δ_0 be the clause $p_s(X) \leftarrow c, p(X)$ and let the root of *Defstree* be the clause δ_0.
$P_s := \{\}$; $Defs := \{\delta_0\}$; $NewDefs := \{\delta_0\}$.

while $NewDefs \neq \{\}$ **do**
  for each δ in *NewDefs* **do**
    consider a new tree $Utree(\delta)$ rooted in δ;
    while there exists a leaf clause λ of $Utree(\delta)$ such that:
    (i) λ has at least one atom in its body, and
    (ii) there is no ancestor α of λ in $Utree(\delta)$ such that $bd(\alpha) \preceq_u bd(\delta)$ **do**
    $\Gamma_\lambda := \{\lambda\}$;
    (UNFOLDING) $\Gamma1_\lambda := \{\gamma \mid$ there exists a clause $\eta \in \Gamma_\lambda$ such that γ is derived by unfolding η w.r.t. the atom $U(bd(\eta))\}$;
    (CLAUSE REMOVAL) $\Gamma2_\lambda := \{\gamma \in \Gamma1_\lambda \mid \gamma$ is of the form $H \leftarrow c, G$ and c is a satisfiable constraint$\}$;
    (CONSTRAINT REPLACEMENT) $\Gamma3_\lambda := \{\gamma \mid$ there exists $(H \leftarrow c, G) \in \Gamma2_\lambda$ and $d = solve(c, vars(H, G))$ and γ is $H \leftarrow d, G\}$;
    Make each clause in $\Gamma3_\lambda$ to be a son of λ in $Utree(\delta)$
    end-while
  end-for
  $\Gamma := \{\lambda \mid \lambda$ is a leaf of $Utree(\delta)$ and $\delta \in NewDefs\}$
  (DEFINITION & FOLDING)
  for each clause $\gamma \in \Gamma$ **do**
  Let γ be of the form: $newp(X) \leftarrow d, A_1, \ldots, A_k$, with $k \geq 0$.

Apply the constraint replacement rule to γ as follows: for $i = 1, \ldots, k$, add the constraint d_i, which is $solve(d, vars(A_i))$, thereby obtaining the clause η_1: $newp(X) \leftarrow d, d_1, \ldots, d_k, A_1, \ldots, A_k$.
$NewDefs := \{\}$;
for $i = 1, \ldots, k$ **do**

Let β be the definition clause in *Defs* with head predicate symbol *newp*.
Let δ be the clause $newq(Y) \leftarrow d_i, A_i$, where *newq* is a new predicate symbol and Y are the linking variables of d_i, A_i in η_i. (Since d_i is $solve(d, vars(A_i))$ we also have that Y are the linking variables of A_i in η_i and $FV(d_i) \subseteq vars(A_i)$).

Case 2.1: in *Defs* there exists a clause δ_v which is a variant of δ apart from the head predicate symbol.

Fold η_i w.r.t. the constrained atom d_i, A_i using δ_v, thereby deriving clause η_{i+1}.

Case 2.2: in *Defs* there exists a clause α such that $bd(\alpha) \preceq_g bd(\delta)$, and either (i) α is β or (ii) α is an ancestor of β in *Defstree* and there is no other ancestor $\overline{\alpha}$ of β such that $bd(\overline{\alpha}) \preceq_g bd(\delta)$ and α is an ancestor of $\overline{\alpha}$ in *Defstree*.

Let $gen(\alpha, \delta)$ be a clause of the form: $genq(Y) \leftarrow e_i, A_i$. By using rule R6, replace in η_i the constraint d_i by e_i, thereby deriving clause ρ_i. (The use of rule R6 is justified by the fact that $\mathcal{D} \models \forall X\,(d, d_i \leftrightarrow d, e_i)$ where $X = FV(d, d_i, e_i)$).
If in *Defs* there exists a clause $gen_v(\alpha, \delta)$ which is a variant of $gen(\alpha, \delta)$ apart from the head predicate symbol
then fold ρ_i w.r.t. the constrained atom e_i, A_i using $gen_v(\alpha, \delta)$, thereby deriving clause η_{i+1}
else $NewDefs := NewDefs \cup \{gen(\alpha, \delta)\}$;
Make $gen(\alpha, \delta)$ to be a son of β in *Defstree*;
Fold ρ_i w.r.t. the constrained atom e_i, A_i using $gen(\alpha, \delta)$, thereby deriving clause η_{i+1}.

Case 2.3: otherwise.

$NewDefs := NewDefs \cup \{\delta\}$; Make δ to be a son of β in *Defstree*;
Fold η_i w.r.t. the constrained atom d_i, A_i using δ, thereby deriving clause η_{i+1}.

end-for;
$P_s := P_s \cup \{\eta_{k+1}\}$;
$Defs := Defs \cup NewDefs$

end-for

end-while

(CONTEXTUAL CONSTRAINT REPLACEMENT)
Let $\mathbf{C} = \{(c, p_s(X))\} \cup \{(d, A) \mid H \leftarrow d, A_1, \ldots, A_n$ is a clause of P_s and A is A_i for some $i = 1, \ldots, n\}$. For every predicate symbol, say q, in P_s
- take the set $\{(d_1, q(V_1)), \ldots, (d_k, q(V_k))\}$ of all constrained atoms in $\mathbf{C}$ with predicate symbol q, and

- compute the least upper bound $m = e_1\{V_1/V\} \sqcup \ldots \sqcup e_k\{V_k/V\}$, where V is a tuple of new variables and for $i = 1, \ldots, k$, the constraint e_i is the existential closure of d_i w.r.t. all variables which are not in V_i.
For every renamed apart clause γ: $q(U) \leftarrow f_1, \ldots, f_n, G$ in P_s, where $f_1, \ldots, f_n$ are atomic constraints, replace γ by $q(U) \leftarrow f, G$, where f is the conjunction of all f_j's, with $0 \leq j \leq n$, such that it is not the case that $m\{V/U\} \sqsubseteq f_j$. (This replacement is done by applying the Contextual Constraint Replacement Rule, and the use of this rule is justified by the fact that, for $i = 1, \ldots, k$, $\mathcal{D} \models \forall Z\,((d_i, U\!=\!V_i) \rightarrow (f_1, \ldots, f_n \leftrightarrow f))$, where $Z = FV(d_i, U, V_i, f_1, \ldots, f_n, f)$.) □

Theorem 1. [*Correctness of the Contextual Specialization Strategy*] Let P be a CLP($\mathcal{D}$) program, and $c, p(X)$ be a constrained atom where $FV(c) \subseteq X$. Let P_s and $p_s(X)$ be the CLP($\mathcal{D}$) program and the atom, respectively, obtained by the Contextual Specialization Strategy by using arbitrary well-quasi-orders $\preceq_u$ and $\preceq_g$, an unfolding selection function U, and a clause generalization operator *gen*. Then, for every variable assignment σ, we have that:

$$lm(P, \mathcal{D}) \models_\sigma c, p(X) \qquad \text{iff} \qquad lm(P_s, \mathcal{D}) \models_\sigma c, p_s(X)$$

Theorem 2. [*Termination of the Contextual Specialization Strategy*] For any CLP($\mathcal{D}$) program P, constrained atom $c, p(X)$ with $FV(c) \subseteq X$, well-quasi-orders $\preceq_u$ and $\preceq_g$, unfolding selection function U, and clause generalization operator *gen* which preserves $\preceq_g$, the Contextual Specialization Strategy terminates.

5 An Extended Example

Let us consider the following CLP($\mathcal{R}_{lin}$) program *Mmod*:

1. $mmod(I, J, M) \leftarrow I \geq J,\ M = 0$
2. $mmod(I, J, M) \leftarrow I < J,\ I1 = I+1,\ mod(I, L),\ mmod(I1, J, M1),$
   $M = M1+L$
3. $mod(X, M) \leftarrow X \geq 0,\ M = X$
4. $mod(X, M) \leftarrow X < 0,\ M = -X$

$Mmod(I, J, M)$ holds if and only if $M = |I| + |I{+}1| + \cdots + |I{+}k|$ and k is the largest integer such that $I{+}k$ is smaller than J. Let us assume that we want to specialize the program *Mmod* w.r.t. the constrained atom $I = 0,\ J \geq 0,\ mmod(I, J, M)$.

In this example we instantiate our Contextual Specialization Strategy as follows. (1) The function *solve* is the function which simplifies conjunctions of linear equations and inequations over the reals similarly to what is done by Holzbaur's clp(q,r) solver [3], (2) the unfolding selection function U returns the *leftmost* atom of a constrained goal, for all constrained goals which contain at least one atom, (3) the well-quasi-order $\preceq_u$ is the relation $\preceq_L$ over constrained goals (see Section 3), (4) the operator *gen* for clause generalization is the one introduced in Example 1 (see Section 3), (5) the wqo $\preceq_g$ is the restriction of $\preceq_L$ to constrained atoms, and (6) the least upper bound operation $\sqcup$ is the convex hull construction (see, for instance, [6]).

We apply the Contextual Specialization Strategy and we start off from the set *Defs* which consists of the following clause only:

5. $mmod_s(I,J,M) \leftarrow I=0,\ J\geq 0,\ mmod(I,J,M).$

By unfolding clause 5 w.r.t. the atom $U(bd(\text{clause}\,5))$, which is $mmod(I,J,M)$, we get:

6. $mmod_s(I,J,M) \leftarrow I=0,\ J\geq 0,\ I\geq J,\ M=0$
7. $mmod_s(I,J,M) \leftarrow I=0,\ J\geq 0,\ I<J,\ I1=I+1,\ mod(I,L),$
$\qquad mmod(I1,J,M1),\ M=M1+L$

We apply the constraint replacement rule using *solve*, and we get:

8. $mmod_s(I,J,M) \leftarrow I=0,\ J=0,\ M=0$
9. $mmod_s(I,J,M) \leftarrow I=0,\ J>0,\ I1=1,\ mod(I,L),\ mmod(I1,J,M1),$
$\qquad M=M1+L$

By unfolding clause 9 w.r.t. $U(bd(\text{clause}\,9))$, which is $mod(I,L)$, and using the function *solve*, we transform clause 9 into clause 10:

10. $mmod_s(I,J,M) \leftarrow I=0,\ J>0,\ I1=1,\ mmod(I1,J,M)$

Since clause 10 is the only leaf in $Utree(\text{clause}\,5)$ with at least one atom in its body and $bd(\text{clause}\,5) \preceq_u bd(\text{clause}\,10)$ being clause 5 an ancestor of clause 10, we stop the unfolding process and we enter the DEFINITION & FOLDING phase. We first apply the constraint replacement rule to clause 10 and we get clause η_1 of the form:

10.r $mmod_s(I,J,M) \leftarrow I=0,\ J>0,\ I1=1,\ J>0,\ I1=1,\ mmod(I1,J,M)$

where d is $I=0,\ J>0,\ I1=1$, and d_1 is $J>0,\ I1=1$. Thus, clause δ is:

δ. $newp(I1,J,M) \leftarrow J>0,\ I1=1,\ mmod(I1,J,M)$

We are in Case 2.2 because the clause in *Defs* with head predicate symbol $mmod_s$ is clause 5 and $bd(\text{clause}\,5) \preceq_g bd(\delta)$. Clause $gen(\text{clause}\,5,\delta)$ is (see Example 1 at the end of Section 3):

11. $genp(I1,J,M) \leftarrow I1\geq 0,\ J\geq 0,\ mmod(I1,J,M)$

We fold clause 10.r by using the definition clause 11 and we get:

12. $mmod_s(I,J,M) \leftarrow I=0,\ J>0,\ I1=1,\ genp(I1,J,M)$

Now *NewDefs* $:= \{\text{clause}\,11\}$ and we continue the execution of the outermost while loop of the Contextual Specialization Strategy by unfolding clause 11. We get:

13. $genp(I1,J,M) \leftarrow I1\geq 0,\ J\geq 0,\ I1\geq J,\ M=0$
14. $genp(I1,J,M) \leftarrow I1\geq 0,\ J\geq 0,\ I1<J,\ I2=I1+1,\ mod(I1,L),$
$\qquad mmod(I2,J,M2),\ M=M2+L$

By unfolding clause 14 w.r.t. $U(bd(\text{clause}\,14))$, which is $mod(I1,L)$, and using the function *solve*, we get from clauses 13 and 14:

15. $genp(I1,J,M) \leftarrow I1\geq 0,\ J\geq 0,\ I1=J,\ M=0$
16. $genp(I1,J,M) \leftarrow I1\geq 0,\ J\geq 0,\ I1<J,\ I2=I1+1,$
$\qquad mmod(I2,J,M2),\ M=M2+I1$

Since clause 16 is the only leaf clause in $Utree$(clause 11) with at least one atom in its body and bd(clause 11) $\preceq_u$ bd(clause 16), being clause 11 an ancestor of clause 16, we stop the unfolding process and we enter the DEFINITION & FOLDING phase. We first apply the constraint replacement rule and since $d_1 = solve((I1 \geq 0, J \geq 0, I1 < J, I2 = I1+1, M = M2+I1), \{I2, J, M2\}) = (J \geq 0, I2 \geq 1)$, we get the clause η_1 of the form:

16.r $genp(I1, J, M) \leftarrow I1 \geq 0,\ J \geq 0,\ I1 < J,\ I2 = I1+1,$
$J \geq 0,\ I2 \geq 1,\ mmod(I2, J, M2),\ M = M2+I1$

Thus, we consider the new clause δ of the form:

$\delta 1$. $newq(I2, J, M2) \leftarrow J \geq 0,\ I2 \geq 1,\ mmod(I2, J, M2)$

We are again in Case 2.2 because the clause in $Defs$ with head predicate symbol $genp$ is clause 11 and bd(clause 11) $\preceq_g$ $bd(\delta 1)$. We have that gen(clause 11, $\delta 1$) is clause 11 itself. We fold clause 16.r using clause 11 and we get:

17. $genp(I1, J, M) \leftarrow I1 \geq 0,\ J \geq 0,\ I1 < J,\ I2 = I1+1,\ genp(I2, J, M2),$
$M = M2+I1$

Since during the last phase of definition and folding steps we did not introduce any new definition, the outermost while loop terminates and we get program P_s made out of clauses 8, 12, 15, and 17.

Now we apply the Contextual Constraint Replacement part of the Contextual Specialization Strategy. We have that:

$$\begin{aligned}\mathbf{C} = \{&(I=0,\ J \geq 0,\ mmod_s(I, J, M)),\\ &(I=0,\ J>0,\ I1=1,\ genp(I1, J, M)),\\ &(I1 \geq 0,\ J \geq 0,\ I1 < J,\ I2=I1+1,\ M=M2+I1,\ genp(I2, J, M2))\}\end{aligned}$$

We then compute the least upper bounds of the constraints corresponding to the predicates $mmod_s$ and $genp$. In order to do so we perform a variable renaming and we existentially quantify all variables in a constraint which do not occur in the corresponding atom. We get:

$$\begin{aligned}\mathbf{C}' = \{&(I=0,\ J \geq 0,\ mmod_s(I, J, M)),\\ &(\exists X\ (X=0,\ J>0,\ I=1),\ genp(I, J, M)),\\ &(\exists X, Y\ (X \geq 0,\ J \geq 0,\ X < J,\ I=X+1,\ Y=M+X),\ genp(I, J, M))\}\end{aligned}$$

$\mathbf{C}'$ can be rewritten as:

$$\begin{aligned}\mathbf{C}'' = \{&(I=0,\ J \geq 0,\ mmod_s(I, J, M)),\\ &(I=1,\ J>0,\ genp(I, J, M)),\\ &(I \geq 1,\ J>I-1,\ genp(I, J, M))\}\end{aligned}$$

Thus, the least upper bounds are:
for $mmod_s$: $m_1 = (I=0,\ J \geq 0)$ and for $genp$: $m_2 = (I \geq 1,\ J > I-1)$. Now we process clauses 8, 12, 15, and 17 as indicated in the final part of the Contextual Specialization Strategy.
For clause 8, since $J=0 \leftrightarrow (J \geq 0,\ J \leq 0)$, $m_1 \sqsubseteq I=0$, and $m_1 \sqsubseteq J \geq 0$, we get:

8.c $mmod_s(I, J, M) \leftarrow J \leq 0,\ M=0$

For clause 12, since $m_1 \sqsubseteq I=0$, we get:

12.c $mmod_s(I, J, M) \leftarrow J>0,\ I1=1,\ genp(I1, J, M)$

For clause 15, once renamed, since $m_2 \sqsubseteq I \geq 0$ and $m_2 \sqsubseteq J \geq 0$, we get:

15.c $genp(I, J, M) \leftarrow I = J,\ M = 0$

For clause 17, once renamed, since $m_2 \sqsubseteq I \geq 0$ and $m_2 \sqsubseteq J \geq 0$, we get:

17.c $genp(I, J, M) \leftarrow I < J,\ I2 = I+1,\ genp(I2, J, M2),\ M = M2+I$

The final program P_s we have derived, consists of clauses 8.c, 12.c, 15.c, and 17.c. Experimental results performed on a PC running Linux and SICStus Prolog, confirm the efficiency improvements of this program w.r.t. the initial program *Mmod*. The speedup is of 7.7 times. Program P_s is also more efficient than the program consisting of clauses 8, 12, 15, and 17, that is, the program derived before the application of the Contextual Constraint Replacement part of our Contextual Specialization strategy, and the speedup in this case of 1.1 times.

6 Conclusions and Related Work

Various methods have been presented in the literature for the specialization of programs w.r.t. properties of their input data [2, 4, 19, 20, 22] and contextual specialization can be viewed as one of those methods. Contextual specialization, in fact, improves programs by taking into account some properties of their context of use. In a previous paper [9] we have presented the basic ideas of the contextual specialization of CLP($\mathcal{D}$) programs presenting also a set of transformation rules for performing that task. Now we build upon that work by proposing a fully automated strategy for the contextual specialization of constraint logic programs.

We have assumed the existence of a solver which is represented in our framework as a computable total function *solve* which simplifies conjunctions of constraints in a given domain $\mathcal{D}$. We have also assumed the existence of: (i) an unfolding selection function U which selects the atom w.r.t. which one should unfold a clause, (ii) a well-quasi-order $\preceq_u$ between constrained goals for controlling the unfolding process (also called *local control*, in the literature), (iii) a clause generalization operator *gen* parameterized by a widening operator ∇, and (iv) a well-quasi-order $\preceq_g$ between constrained atoms, which activates clause generalization during the specialization process. We have shown the termination of our specialization strategy. The partial correctness of this strategy is a consequence of the fact that the transformation rules preserve the chosen semantics, which, in our case, is the least $\mathcal{D}$-model.

The hypothesis that the function *solve* is assumed to be complete w.r.t. the satisfiability test, makes our approach different from the one in [2], where the specialization process is based on some undecidable properties and thus, it cannot be easily automated.

The use of our contextual constraint replacement rule R5 allows us to perform optimizations which can be performed neither by using the transformation rules presented in [1, 8, 18] nor by applying the techniques for partial evaluation of logic programs presented in [16, 25]. Indeed, our contextual constraint replacement rule is able to take advantage of the information provided by the context of use of every predicate of the programs to be specialized.

In our work we have used of the well-quasi-ordering techniques [7] for proving termination of rewriting systems. The same techniques were used in [14, 15] for ensuring termination of partial deduction and controlling the introduction of new definitions which are obtained by generalization of old definitions.

Other approaches to the simplification and manipulation of constraints with the aim of improving the efficiency of the derived programs have been described in the literature. Among these we want to mention the approaches based on: (i) multiple specializations and parallelization [22], (ii) optimizations relying on sophisticated forms of interaction with the constraint solver for reducing the workload of the solver itself [19] (these optimizations generate non-monotonic CLP programs), and (iii) abstract interpretation for specifying the context of use for which the programs have to be specialized [4] (this approach has been presented for a strict first-order functional programming language, not a CLP language).

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. Holzbaur C. Ofai clp(q,r) manual, edition 1.3.2. Technical Report TR-95-09, Austrian Research Institute for Artificial Intelligence, Vienna, 1995.
4. C. Consel and S.C. Khoo. Parameterized partial evaluation. *ACM Transactions on Programming Languages and Systems*, 15(3):463–493, 1993.
5. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixpoints. In *Proceedings 4th ACM-SIGPLAN Symposium on Principles of Programming Languages (POPL '77)*, pages 238–252. ACM Press, 1977.
6. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the Fifth ACM Symposium on Principles of Programming Languages (POPL'78)*, pages 84–96. ACM Press, 1978.
7. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leuveen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier, 1990.
8. S. Etalle and M. Gabbrielli. Transformations of CLP modules. *Theoretical Computer Science*, 166:101–146, 1996.
9. F. Fioravanti, A. Pettorossi, and M. Proietti. Rules and strategies for contextual specialization of constraint logic programs. In M. Leuschel, editor, *Proceedings of the ICLP'99 Workshop on Optimization and Implementation of Declarative Programming Languages, WOID '99,* Las Cruces University, New Mexico, USA, pages 1–9, December 2–3, 1999.
10. 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.

11. J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
12. J. Jaffar, M. Maher K. Marriott, and P. Stuckey. The semantics of constraint logic programming. *Journal of Logic Programming*, 37:1–46, 1998.
13. N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.
14. M. Leuschel. Improving homeomorphic embedding for online termination. In P. Flener, editor, *Proceedings of LOPSTR'98, Manchester, UK, June 1998*, Lecture Notes in Computer Science 1559, pages 199–218. Springer-Verlag, 1999.
15. M. Leuschel, B. Martens, and D. De Schreye. Controlling generalization and polyvariance in partial deduction of normal logic programs. *ACM Transactions on Programming Languages and Systems*, 20(1):208–258, 1998.
16. 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.
17. J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *Journal of Logic Programming*, 11:217–242, 1991.
18. M. J. Maher. A transformation system for deductive database modules with perfect model semantics. *Theoretical Computer Science*, 110:377–403, 1993.
19. Kim Marriott and Peter Stuckey. The 3 R's of optimizing constraint logic programs: Refinement, removal and reordering. In *POPL'93: Proceedings ACM SIGPLAN Symposium on Principles of Programming Languages*, 1993.
20. 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.
21. 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.
22. 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.
23. D. Sahlin. Mixtus: An automatic partial evaluator for full Prolog. *New Generation Computing*, 12:7–51, 1993.
24. M. H. Sørensen and R. Glück. An algorithm of generalization in positive supercompilation. In J. W. Lloyd, editor, *Proceedings of the 1995 International Logic Programming Symposium (ILPS '95)*, pages 465–479. MIT Press, 1995.
25. 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.