Towards Automatic Control for CLP(χ) Programs

Fred Mesnard

Iremia, Université de la Réunion
15, avenue René Cassin - BP 7151 - 97 715 Saint-Denis Messag. Cedex 9
France
fred@univ-reunion.fr

Abstract. We discuss issues of control for constraint logic programs. The problem we try to solve is to find, from the text of a program, a computation rule which ensures finiteness of the computation tree. In a single framework, we address two related areas, namely the generation of control annotations and the local level of control for partial deduction.

1 Introduction

We discuss issues of control for constraint logic programs [9], [3], [10]. It is well known that for the same goal, one computation rule can give rise to a finite computation tree and another one to an infinite computation tree. So the problem we try to solve is how to find, from the text of a program, a computation rule which ensures finiteness of the computation tree. In a single framework, we address two related areas, namely the generation of control annotations and the local level of control for partial deduction (see [13] for a discussion of local versus global level of control).

The paper is organized as follows: Sect. 2 justifies the switch to $\mathcal{Q}^+$. In Sect. 3, we briefly discuss the derivation of interargument relations. Then we focus on the inference of control information in Sect. 4. Section 5 presents a class of computation rules, which can lifted to the original computation domain (Sect. 6). At last, we summarize our approach, compare it with related work and conclude by sketching possible extensions of the proposed method in Sect. 7.

2 From CLP(χ) to CLP($\mathcal{Q}^+$)

The first step in the approach we propose is to switch from χ to $\mathcal{Q}^+$ by using an approximation A which consists in an algebraic morphism associated to a syntactic transformation. (By $\mathcal{Q}^+$, we mean the structure $\langle \mathbb{Q}^+; \{0, 1, +\}; \{=, \geq\}\rangle$ where $\mathbb{Q}^+$ denotes the set of non-negative rational numbers). The approximation A captures a notion of size for the elements of χ. We now define more precisely what we call an approximation.

Let $\chi = \langle D_\chi; F_\chi; \{=_\chi\} \cup C_\chi\rangle$ where D_χ is the domain of χ, F_χ is a set of functions and C_χ a set of relations, be a solution-compact structure without any limit element [10]. Let $\psi = \langle D_\psi; F_\psi; \{=_\psi\} \cup C_\psi \rangle$ be another solution-compact structure.

Definition 1. An *approximation* A consists in a pair of functions $\langle A_{sx}, A_{sm} \rangle$ where:

1. A_{sx} is a mapping from $F_\chi \cup \{=_\chi\} \cup C_\chi$ to $F_\psi \cup \{=_\psi\} \cup C_\psi$ such that:
   (a) for every function or relation symbol s, $arity(s) = arity(A_{sx}(s))$;
   (b) $A_{sx}(F_\chi) \subseteq F_\psi$;
   (c) $A_{sx}(=_\chi) = =_\psi$;
   (d) $A_{sx}(C_\chi) \subseteq C_\psi$.
2. A_{sm} is a mapping from D_χ to D_ψ such that for every $\widetilde{e_\chi} \in \widetilde{D_\chi}$:
   (a) for every function f_χ, $A_{sm}(f_\chi(\widetilde{e_\chi})) =_\psi A_{sx}(f_\chi)(A_{sm}(\widetilde{e_\chi}))$;
   (b) for every relation c_χ, if $\models_\chi c_\chi(\widetilde{e_\chi})$ then $\models_\psi A_{sx}(c_\chi)(A_{sm}(\widetilde{e_\chi}))$.

Example 1. Let $List(\chi) = \langle {D_\chi}^*; \{<>\} \cup \bigcup_{e \in D_\chi} \{< e >\} \cup \{\cdot\}; \{=_{List(\chi)}\} \rangle$ where the constant $<>$ denotes the empty list, the constants $< e >$ denote the lists $< e >$ and $\cdot$ is the concatenation of lists. We define the approximation A from $List(\chi)$ to $\mathcal{Q}^+$:

$$A_{sx}(<>) = 0,\ A_{sx}(< e >) = 1,\ A_{sx}(\cdot) = + \text{ and } A_{sm}(< e_1, \ldots, e_n >) = n.$$

One verifies that $A = \langle A_{sx}, A_{sm} \rangle$ satisfies the definition 1.

Let V be a denumerable set of variables and P be a finite set of predicate symbols. The set of *terms*, *constraints*, *atoms*, *programs*, *goals* are defined as usual. If we extend A_{sx} on $V \cup P$ with $A_{sx}(x) = x$ (resp. $A_{sx}(p) = p$) for every variable x (resp. for every predicate symbol p), then A may be naturally extended on terms, constraints, atoms, programs and goals.

Example 2. Consider the CLP($List(\chi)$) program P:

$$p(< x_1, x_2, x_3, x_4, x_5 >, < x_5, x_4, x_3, x_2, x_1 >, <>) \leftarrow \diamond$$
$$p(< x_1, x_2 > .x, y. < y_1 > .y, < y_1 > .z) \leftarrow \diamond p(y, x.x. < x_1, y_1, x_2 >, z)$$

Here is its approximation $A(P)$ in CLP($\mathcal{Q}^+$):

$$p(5, 5, 0) \leftarrow \diamond$$
$$p(x + 2, 2y + 1, z + 1) \leftarrow \diamond p(y, 2x + 3, z)$$

For a more rigorous treatment and some properties of approximations (e.g. $A(M_P) \subseteq M_{A(P)}$), we refer the reader to [14]. The original idea of such mappings was already present in [21] and developed in [2].

Let us go back to our main subject. The following result gives a first justification for switching to $\mathcal{Q}^+$: termination in $\mathcal{Q}^+$ implies termination in $List(\chi)$.

Theorem 2. *Let A be an approximation from CLP(χ) to CLP(ψ), P a CLP(χ) program and $A(P)$ its approximation, G a goal and $A(G)$ its approximation, R_2 a computation rule for CLP(ψ). If the $(A(P), R_2)$-computation tree for $A(G)$ is finite, then the $(P, R_2 \circ A)$-computation tree for G is finite.*

Note that in Theorem 2, $R_2 \circ A$ means that we first abstract the goal, then choose a literal, denoted by its rank in the goal (cf definition 8) using R_2. So $R_2 \circ A$ is indeed a computation rule for CLP(χ). Moreover, because of condition 2(b) of definition 1, a branch of the $(P, R_2 \circ A)$-computation tree for G may lead to a failure node, though its corresponding branch in CLP(ψ) may be longer and lead to a success node. A proof of Theorem 2 is given in [14].

A second justification for choosing $\mathcal{Q}^+$instead of $\mathcal{N} = \langle \mathbb{N}; \{0, 1, +\}; \{=, \geq\}\rangle$ lies in the fact that many problems are much easier to solve in $\mathcal{Q}^+$ than in $\mathcal{N}$. We believe that the achieved efficiency largely outweighs the loss in accuracy.

3 Inference of Interargument Relations

For termination analysis, it is essential to have information about the relationship between between the sizes of the arguments of a predicate (see for instance the examples 5 and 6 of Sect. 4). In the logic programming framework, the research devoted to the derivation of interargument relations (IR) begun with [20] and is nicely summarized in [5]. We just mention some of the main results obtained so far.

In [22], the authors show how one may compute by abstract interpretation IR's of the form: $\forall(x_1, \ldots, x_n)\ p(x_1, \ldots, x_n) \rightarrow \bigwedge_{1 \leq j \leq m} \left[\sum_{1 \leq i \leq n} c_i^j x_i = k^j\right]$.

In [6], the class of "3-recursive" CLP($\mathcal{Z}$) logic procedures is defined as follows:

$$
\begin{array}{l}
p(\tilde{x}) \leftarrow \xi(\tilde{x}) \diamond \\
p(\tilde{x} + \tilde{a}) \leftarrow \Phi_1(\tilde{x}) \ \diamond\ p(\tilde{x}) \\
p(\tilde{x} + \tilde{b}) \leftarrow \Phi_2(\tilde{x}) \ \diamond\ p(\tilde{x}) \\
p(\tilde{x} + \tilde{c}) \leftarrow \Phi_3(\tilde{x}) \ \diamond\ p(\tilde{x})
\end{array}
$$

where $\tilde{a}$, $\tilde{b}$ and $\tilde{c}$ are vectors of integers, $\xi(\tilde{x}), \Phi_1(\tilde{x}), \Phi_2(\tilde{x})$ and $\Phi_3(\tilde{x})$ are finite linear arithmetic constraints. Such a procedure p can be *exactly* characterized by a finite disjunction of arithmetic constraints. The authors have enumerated all the 512 cases, and for each case, they have computed the characterization. As a consequence, the computational cost of the inference of the meaning of a 3-recursive procedure is almost constant.

In [18], for the class of "linear recursive logic programs satisfying the translativeness property", a technique is presented which enables the derivation of IR's in the form of polyhedral convex sets.

At last, we require that the inferred IR's are added as constraints to the original program P, which leads to a specialized version P_{spec} of P. Note that as the IR's are logical consequences of the least model of P, we preserve the meaning of P (i.e. $M_P = M_{P_{spec}}$).

4 Inference of Control Information

A basic idea for termination relies on associating to every recursive predicate a measure which decreases at each recursive call. For sake of simplicity, we disallow mutually recursive procedures, i.e. we assume that the transitive closure of the dependency graph D_P (see the definition 11 in the appendix) of the program P is antisymmetric.

Definition 3. A *measure* $\mu = (\mu_1, \ldots, \mu_n)$ where the $\mu_i's \in \mathbb{Q}^+$ is a mapping from $(\mathbb{Q}^+)^n$ to $\mathbb{Q}^+$ such that: $\forall (a_1, \ldots, a_n) \in (\mathbb{Q}^+)^n \quad \mu(a_1, \ldots, a_n) = \sum_{i=1}^n \mu_i a_i$.

For a recursive predicate p of P, we are interested in measures which effectively decreases in the least model M_P of P, i.e. we need a link with the semantics of p.

Definition 4. A measure μ_p associated to a recursive predicate p is *valid* if for each clause $p(\tilde{t}) \leftarrow c \diamond \tilde{B}$ defining p in P, for each solution θ of c such that $\tilde{B}\theta \in M_P$, for each atom $p(\tilde{s})$ appearing in $\tilde{B}$, we have: $\mu_p(\tilde{t}\theta) \geq \mu_p(\tilde{s}\theta) + 1$.

Example 3. *$\mu_p^1 = (2, 1, 0)$ and $\mu_p^2 = (0, 0, 1)$ are valid measures for the approximated version of the program defined in example 2 because: $\forall (x, y, z) \in (\mathbb{Q}^+)^3, 2x + 4 + 2y + 1 \geq 2y + 2x + 3 + 1$ and $\forall z \in \mathbb{Q}^+, z + 1 \geq z + 1$.*

However, we have to weaken definition 4, in order to automate the computation of measures.

Definition 5. A measure μ_p associated to a recursive predicate p is *t-valid* (t for textually) if for each clause $p(\tilde{t}) \leftarrow c \diamond \tilde{B}$ defining p in P, for each solution θ of c, for each atom $p(\tilde{s})$ appearing in $\tilde{B}$, we have: $\mu_p(\tilde{t}\theta) \geq \mu_p(\tilde{s}\theta) + 1$.

Example 4. *In example 3, we have in fact shown that μ_p^1 and μ_p^2 are t-valid measures for p, but :*

Proposition 6. *If μ_p is t-valid for p, then μ_p is valid for p.*

Of course, the converse is false.

Example 5. *The measure $\mu_q = (1)$ is valid q:*

$$q(1) \leftarrow \diamond$$
$$q(2x) \leftarrow \diamond q(x)$$

because $q(x) \in M_q$ implies $x \geq 1$ but μ_q is not t-valid. The non-recursive clauses are indirectly useful to termination analysis: they enable the derivation of inter-argument relations which should be added to the program before computing valid measures. In our example, it gives:

$$q(1) \leftarrow \diamond$$
$$q(2x) \leftarrow x \geq 1 \diamond q(x)$$

Now $\mu_q = (1)$ is t-valid.

Automatic discovery of t-valid measures is a consequence of a result of [19]:

Theorem 7. *There exists a complete polynomial procedure for deciding the existence of a t-valid measure for any logic procedure defined using a single predicate symbol.*

This procedure, based on the duality theorem of linear programming (see [19]), can be adapted to compute the coefficients of μ [14]. For a more general logic procedure p where $r_1, \ldots, r_n$ appears in the definition of p, we abstract the meaning of each r_i by a constraint supposed to be the "most precise rational superset" of the meaning of r_i, and then we apply Theorem 7.

Example 6. Consider the following program P:

$$\begin{array}{l} p(0) \leftarrow \diamond \\ p(x) \leftarrow \diamond\, r(x,y), p(y) \\ r(x+u,0) \leftarrow 3 \geq u \geq 1 \diamond \\ r(x+1,y+1) \leftarrow \diamond\, r(x,y) \end{array}$$

Let us assume that we are interested in finding a valid measure for p and that we have inferred : $\forall x \in \mathbb{Q}^+ \, [p(x) \Rightarrow x \geq 0]$ *and* $\forall (x,y) \in (\mathbb{Q}^+)^2 \, [r(x,y) \Rightarrow x \geq y+1]$. *We first specialize P into P_{spec}:*

$$\begin{array}{l} p(0) \leftarrow \diamond \\ p(x) \leftarrow x \geq y+1 \diamond r(x,y), p(y) \\ r(x+u,0) \leftarrow 3 \geq u \geq 1 \diamond \\ r(x+1,y+1) \leftarrow x \geq y+1 \diamond r(x,y) \end{array}$$

Then we forget r, which gives P' :

$$\begin{array}{l} p'(0) \leftarrow \diamond \\ p'(x) \leftarrow x \geq y+1 \diamond p'(y) \end{array}$$

Roughly speaking, the least model of P' is generally a superset of the least model of P_{spec}. Then we compute (Theorem 7) $\mu_{p'} = (1)$, which is t-valid for P' and t-valid for P_{spec}.

We conclude that on the one hand, the notion of validity refers directly to the semantics of the program. On the other hand, the notion of t-validity refers to the text of the program but remains computable.

5 An Extended Resolution for CLP($\mathcal{Q}^+$)

The operational semantics we intend is an extension of the standard top-down execution of CLP. The computation tree can be incomplete by having any goal as a leaf and the computation rule uses and updates a history of the computation. Let us be more precise.

Let c (resp. t) be a constraint (resp. a term) of CLP($\mathcal{Q}^+$). *Min(c;t)* denotes the least (rational) value of t in c. *Bounded(c;t)* is true iff the set of values for t in c is bounded by a (rational) number. Let P be a program which defines a set π of predicate symbols. Let Rec be the subset of π denoting the recursive procedures of P. Let V be a countable set of variables. Let At be the set of atoms defined using π, {0,1,+} and V. A *supervised atom* is a pair $\langle p(\tilde{t}), m\rangle$ where $p(\tilde{t}) \in At$ and $m \in \mathbb{Q}^+$. The set of all supervised atoms is denoted by $SupAt$ and its powerset by 2^{SupAt}. Finally, let $Hist \in 2^{SupAt}$ and RES be the set of all resolvents.

Definition 8. An *extended computation rule* (ecr) $R : RES \times 2^{SupAt} \longrightarrow \mathbb{N} \times 2^{SupAt}$ verifies:

1. $R(\leftarrow c \diamond, Hist) = \langle 0, Hist\rangle$
2. $R(\leftarrow c \diamond A_1, \ldots, A_{n+1}, Hist) = \langle i, Hist'\rangle$ with $0 \leq i \leq n+1$.

Definition 9. Given a goal G and an extended computation rule R, the *extended computation tree* $\tau_{G,R}$ is the smallest tree such that:

1. $\langle G, \phi\rangle$ is the root of $\tau_{G,R}$,
2. if $\langle G', Hist'\rangle$ is a node such that $R(G', Hist') = \langle 0, Hist'\rangle$ then $\langle G', Hist'\rangle$ is a leaf,
3. if $\langle G', H'\rangle$ is a node where $G' = \leftarrow c \diamond A_1, \ldots, A_i, \ldots, A_{n+1}$ and $R(G', H') = \langle i, H''\rangle$ with $i \geq 1$ then the node has a child $\langle \leftarrow c \diamond A_1, \ldots, \tilde{B}, \ldots, A_{n+1}, H''\rangle$ for each clause of P: $Head \leftarrow c' \diamond \tilde{B}$ such that $c, c', A_i = Head$ is satisfiable. Moreover, each atom of $\tilde{B}$ with the same predicate symbol as A_i is marked as `checked`.

We are now in position to define the class $\mathcal{R}$ of ecr's we are interested in, parameterized by a function $SelectAtom : RES \times 2^{SupAt} \longrightarrow \mathbb{N}$ which has to satisfy the postcondition $0 \leq SelectAtom(\leftarrow\ c \diamond A_1, \ldots, A_k) \leq k$ (see Fig. 1).

So $ExtendedComputationRule$ provides a shell which uses and computes histories. The idea is that it divides G into non-recursive and recursive atoms, which in turn is divided into checked, bounded and unbounded atoms. If an unbounded atom is selected, then it adds the atom and its current minimum value for its measure to the history. In any case, it checks that unfolding the selected atom is allowed wrt the updated history. Note that once a function $SelectAtom$ has been chosen, which may embody various heuristics criteria, we hold a particular ecr which satisfies the definition 8. Moreover, any ecr of $\mathcal{R}$ ensures termination.

Theorem 10. *Let P be a CLP($\mathcal{Q}^+$) program such that its dependency graph is antisymmetric. Assume that a valid measure is associated with each recursive predicate. Let R be an extended computation rule $\in \mathcal{R}$. Then for any goal G, the extended computation tree $\tau_{G,R}$ is finite.*

The proof is given in the appendix. We propose two particular ecr's which illustrate the range of $\mathcal{R}$.

If $SelectAtom$ chooses the leftmost atom $\in NR \cup C \cup B$, then we don't need to compute and check the history before we allow this choice (see the proof

$$
\begin{array}{l}
\mathit{ExtendedComputationRule}(G,H) = \\
\textbf{let}\quad i = \mathit{SelectAtom}(G,H) \\
\textbf{in}\quad \textbf{if } i = 0 \textbf{ then } \langle 0, H\rangle \\
\qquad \textbf{else} \\
\qquad\qquad \textbf{let}\quad G = \leftarrow c \diamond p_1(\tilde{t}_1), \ldots, p_{k+1}(\tilde{t}_{k+1}) \\
\qquad\qquad\qquad NR = \{j \mid 1 \le j \le k+1, p_j \notin \mathit{Rec}\} \\
\qquad\qquad\qquad R = \{j \mid 1 \le j \le k+1, p_j \in \mathit{Rec}\} \\
\qquad\qquad\qquad C = \{j \mid j \in R, p_j(\tilde{t}_j) \textit{ is marked as } \texttt{checked}\ \} \\
\qquad\qquad\qquad B = \{j \mid j \in R \setminus C, \mathit{Bounded}(c; \mu_{p_j}(\tilde{t}_j))\} \\
\qquad\qquad\qquad U = \{j \mid j \in R \setminus (B \cup C)\} \\
\qquad\qquad\qquad \left\{\begin{array}{ll} p_i(\tilde{h}_1) \leftarrow c_1 \diamond \ldots & \\ \vdots & \qquad\qquad \textit{the } n \textit{ clauses of } P \textit{ defining } p_i \\ p_i(\tilde{h}_n) \leftarrow c_n \diamond \ldots & \end{array}\right. \\
\qquad\qquad\qquad H' = \textbf{if } i \in U \textbf{ then } H \cup \{\langle p_i(\tilde{t}_i), \mathit{Min}(c, \mu_{p_i}(\tilde{t}_i))\rangle\} \\
\qquad\qquad\qquad\qquad\quad \textbf{else } H \\
\qquad\qquad\qquad \mathit{stop} = \exists j,\ 1 \le j \le n, \exists \langle q(\tilde{s}), m\rangle \in H', \\
\qquad\qquad\qquad\qquad\qquad m < \mathit{Min}(c, c_j, \tilde{t}_i = \tilde{h}_j; \mu_q(\tilde{s})) \\
\qquad\qquad \textbf{in} \\
\qquad\qquad\qquad \textbf{if } \mathit{stop} \textbf{ then } \langle 0, H\rangle \\
\qquad\qquad\qquad \textbf{else } \langle i, H'\rangle
\end{array}
$$

Fig. 1. The function *ExtendedComputationRule*.

of Theorem 10). As a consequence, *such an extended computation rule can be directly wired in a CLP system with a delay primitive.* A complete example is presented in Sect. 6.

On the other end of the scale, we may unfold as much as possible by selecting an atom $\in NR \cup C \cup B$ such that *stop* remains false. We postpone as far as we can the selection of atoms $\in U$ because it implies an increase of the history which potentially refrains the unfolding.

To conclude, it seems reasonable to mix the approaches in a *two-level top-down execution* as follows. Given a goal, we prove it using directly the CLP system with a delay primitive. If there are remaining frozen goals, and if the user agrees, we may unfold them using a meta-interpreter with a smarter ecr as explained above. We call R_{ext} such an ecr.

6 Back to CLP(χ)

We now present a complete example which sketches how one might come back to the original domain of computation. Consider the CLP($\mathbb{Q}$, $List(\mathbb{Q})$) program:

$$\mathit{sort}(<>,<>)\ \leftarrow \diamond$$

$$sort(<x>.y, z) \leftarrow \diamond\, sort(y,t), insert(t,x,z)$$

$$\begin{aligned}
&insert(<>, x, <x>) \leftarrow \diamond \\
&insert(<x>.y, w, <w,x>.y) \leftarrow w \leq x \diamond \\
&insert(<x>.y, w, <x>.z) \leftarrow w > x \diamond insert(y,w,z)
\end{aligned}$$

First, we approximate it in CLP($\mathcal{Q}^+$):

$$\begin{aligned}
&sort(0,0) \leftarrow \diamond \\
&sort(y+1, z) \leftarrow \diamond\, sort(y,t), insert(t, _, z)
\end{aligned}$$

$$\begin{aligned}
&insert(0, _, 1) \leftarrow \diamond \\
&insert(y+1, _, y+2) \leftarrow \diamond \\
&insert(y+1, _, z+1) \leftarrow \diamond\, insert(y, _, z)
\end{aligned}$$

Second, inference of interargument relations may conclude that:

$$\begin{aligned}
&\forall (y,z) \in (\mathbb{Q}^+)^2\; insert(y, _, z) \Rightarrow z = y+1 \\
&\forall (x,y) \in (\mathbb{Q}^+)^2\; sort(x,y) \Rightarrow x = y
\end{aligned}$$

We add the information to the program:

$$\begin{aligned}
&sort(0,0) \leftarrow \diamond \\
&sort(y+1, z) \leftarrow z = t+1, y = t \diamond sort(y,t), insert(t, _, z)
\end{aligned}$$

$$\begin{aligned}
&insert(0, _, 1) \leftarrow \diamond \\
&insert(y+1, _, y+2) \leftarrow \diamond \\
&insert(y+1, _, z+1) \leftarrow z = y+1 \diamond insert(y, _, z)
\end{aligned}$$

Third, we compute the valid measures for *sort* and *insert*:

$$\begin{aligned}
&\mu^1_{insert}(x, _, z) = (1,0,0) \text{ and } \mu^2_{insert}(x, _, z) = (0,0,1) \\
&\mu^1_{sort}(x,y) = (1,0) \text{ and } \mu^2_{sort}(x,y) = (0,1)
\end{aligned}$$

Note that it is useless at this point to try to infer new interargument relations. Fourth, we lift to the original program either by relating lists to their length, e.g. $\mu^1_{insert}(x, \_, z) = l$ where $x :: l$ and using R_{ext} (see the end of Sect. 5) or by compiling the valid measures we found (such a compilation can be easily automated):

$$\begin{aligned}
&sort(x,y) \leftarrow x :: l, y :: l \diamond freeze(l, sort'(x,y)) \\
&sort'(<>, <>) \leftarrow \diamond \\
&sort'(<x>.y, z) \leftarrow y :: l, t :: l, z :: l+1 \diamond sort'(y,t), insert(t,x,z)
\end{aligned}$$

$$\begin{aligned}
&insert(x,y,z) \leftarrow x :: l, z :: l+1 \diamond freeze(l, ins'(x,y,z)) \\
&ins'(<>, x, <x>) \leftarrow \diamond \\
&ins'(<x>.y, w, <w,x>.y) \leftarrow w \leq x \diamond \\
&ins'(<x>.y, w, <x>.z) \leftarrow y :: l, z :: l+1, w > x \diamond ins'(y,w,z)
\end{aligned}$$

The proof of any query about *sort* and *insert* on a CLP system with the *freeze* primitive will stop. For instance:

$$
\begin{aligned}
&> 1 < x < y \diamond sort(n, n), insert(l, x+1, m), \\
&\qquad insert(m, 4y, n), sort(l, < 2y, 3x+1 >); \\
&\{1 < x < y, 2y \leq 3x+1, n =< x+1, 2y, 3x+1, 4y >, \\
&\qquad l =< 2y, 3x+1 >, m =< x+1, 2y, 3x+1 >\} \\
&>
\end{aligned}
$$

7 Discussion

Let us first summarize our approach for the control of a CLP(χ) program P.

1. We abstract P to $A(P)$ in CLP($\mathcal{Q}^+$).
2. We compute interargument relations and add them to $A(P)$
3. We compute the valid measures for $A(P)$.
4. We lift to CLP(χ) either by compiling the measures in order to prove the queries directly with the underlying CLP system or we explicitly run a meta-interpreter based on the extended resolution of Sect. 5.

Before we discuss related work, we point out that our approach can be extended by allowing multiple valid measures for a procedure and mutual recursion.

Termination. Of course we owe a lot to the works on compile-time termination analysis of logic programs. This research begun with [20], [16] and [21]. Once again, we refer the reader to the survey of D. De Schreye and S. Decorte [5]. But we believe that there is no essential difference between termination and local control for partial deduction. Moreover, as most constraint logic programming systems supply a delay primitive, termination analysis should not rely so much on the left-to-right computation rule. We briefly come back to this point at the end of this section.

Automatic control generation. In [15], Naish presents a technique for deriving wait-declarations from the text of the program. However, termination is not guaranteed. Later he notices that non-linearity (when a variable appears more than once in a goal) may lead to non-termination of annotated programs (also true for [11]). Consider the program:

$$
\begin{aligned}
&append(<>, y, y) \leftarrow \diamond \\
&append(< x_1 > .x, y, < x_1 > .z) \leftarrow \diamond append(x, y, z)
\end{aligned}
$$

where a call to $append(x, y, z)$ is delayed until $nonvar(x)$ or $nonvar(z)$. It is for instance the `DELAY` control declaration given for *append* in the module for lists processing of the Gödel programming language [8]. The proof of the goal $G :\leftarrow \diamond append(< x_1 > .x, y, x)$ does not terminate.

Using our technique, we compute the interargument relation for the approximated version of append: $append(x, y, z) \Rightarrow z = x + y$ and the two valid measures: $\mu^1_{append} = (1, 0, 0)$ and $\mu^2_{append} = (0, 0, 1)$. Then we compile this knowledge to obtain:

$$
\begin{array}{l}
append(x,y,z) \leftarrow x :: l_x, y :: l_y, z :: l_z, l_z = l_x + l_y \diamond \\
\qquad freeze(l_x, once(v, append'(x,y,z))), \\
\qquad freeze(l_z, once(v, append'(x,y,z))) \\
append'(<>, y, y) \leftarrow \diamond \\
append'(< x_1 > .x, y, < x_1 > .z) \leftarrow \diamond append'(x,y,z)
\end{array}
$$

$$
\begin{array}{l}
once(v, G) \leftarrow \diamond free(v), !, eq(v, cst), G \\
once(v, G) \leftarrow \diamond
\end{array}
$$

Now the proof of the goal $\leftarrow \ \diamond append(< x_1 > .x, y, x)$ fails because the constraint $< x_1 > .x :: 1 + l_x, y :: l_y, x :: l_x, l_x = 1 + l_x + l_y$ is unsatisfiable.

Local control in partial deduction. Ensuring finite unfolding is one of the problems of partial deduction. Various ad hoc solutions, e.g. imposing an arbitrary depth bound to the derivations, have been proposed, which obviously do not really address the problem. Our technique is firmly based on the criterion established in [1]. The first difference is that we directly move to the CLP($\mathcal{Q}^+$). It allows a smooth integration of interargument relations, and makes the implementation easier and more efficient. The major improvement is that we rely on the concept of valid measures, related to the semantics of the program, to control the unfolding. We illustrate this point. Consider the program given in example 2.
The unfolding of the goal G: $\leftarrow \ \diamond p(7,9,z)$ gives:

$$
\begin{array}{l}
G_1 : \langle \leftarrow \ \diamond p(7,9,z); \phi \rangle \\
G_2 : \langle \leftarrow z = z_1 + 1 \diamond \ p(4,13,z_1); \phi \rangle \\
G_3 : \langle \leftarrow z = z_2 + 2 \diamond \ p(6,7,z_2); \phi \rangle \\
G_4 : \langle \leftarrow z = z_3 + 3 \diamond \ p(3,11,z_3); \phi \rangle \\
G_5 : \langle \leftarrow z = z_4 + 4 \diamond \ p(5,5,z_4); \phi \rangle \\
G_6 : \langle \leftarrow z = 4 \diamond; \phi \rangle
\end{array}
$$

because the valid measure $\mu_p^1(x,y,z) = 2x + y$ is bounded for G. As the first argument increases from G_2 to G_3, the second argument and the sum of the two arguments increase from G_1 to G_2, and the minimum value of the third argument remains 0, the partial deduction process as described in [1] stops at G_3.
The unfolding of the goal $G' :\leftarrow \ \diamond p(x+7, y+9, z)$ under μ_p^1 gives :

$$
\begin{array}{l}
G_1 : \langle \leftarrow \ \diamond p(x+7, y+9, z); \phi \rangle \\
G_2 : \langle \leftarrow z = z_1 + 1 \diamond \ p(y/2+4, 2x+13, z_1); \{\langle p(x+7, y+9, z_1+1), 23 \rangle\} \rangle \\
G_3 : \langle \leftarrow z = z_2 + 2 \diamond \ p(x+6, y+7, z_2); \{\langle p(x+7, y+9, z_2+2), 23 \rangle\} \rangle \\
G_4 : \langle \leftarrow z = z_3 + 3 \diamond \ p(y/2+3, 2x+11, z_3); \{\langle p(x+7, y+9, z_3+3), 23 \rangle\} \rangle \\
G_5 : \langle \leftarrow z = z_4 + 4 \diamond \ p(x+5, y+5, z_4); \{\langle p(x+7, y+9, z_4+4), 23 \rangle\} \rangle \\
\quad G_6 : \langle \leftarrow x = 0, y = 0, z = 4 \diamond; \{\langle p(x+7, y+9, 4), 23 \rangle\} \rangle
\end{array}
$$

or

$$
\begin{array}{l}
G_7 : \langle \leftarrow z = z_5 + 5 \diamond \ p(y/2+2, 2x+9, z_5); \{\langle p(x+7, y+9, z_5+5), 23 \rangle\} \rangle \\
G_8 : \langle \leftarrow z = z_6 + 6 \diamond \ p(x+4, y+3, z_6); \{\langle p(x+7, y+9, z_6+6), 23 \rangle\} \rangle
\end{array}
$$

Once again, the unfolding process described in [1] stops at G_3. In our case, the derivation is stopped at G_8 because otherwise, the history would be violated. From this extended computation tree, we may extract the property: $\forall(x,y,z) \in (\mathbb{Q}^+)^3 p(x+7,y+9,z) \Leftrightarrow (x=y=0 \wedge z=4) \vee (p(x+4,y+3,z) \wedge z \geq 6)$ true in the least $\mathcal{Q}^+$-model of the program. Or, if we only keep the deterministic part of the extended computation tree: $\forall(x,y,z) \in (\mathbb{Q}^+)^3 p(x+7,y+9,z) \Leftrightarrow p(x+5,y+5,z) \wedge z \geq 4$.

To conclude, we believe that the following points deserve further work. It seems feasible to adapt the approach we propose in this paper to other termination criteria [12]. Deterministic goals should be taken into account in order to reduce the size of the computation tree [7], [17]. The switch from χ to $\mathcal{Q}^+$ could be automated [4]. At last, it would nice if we could compute the "maximal" class of goals such that there is no remaining frozen goal in the computation tree. As in [14], a starting point could be to add a second layer of abstraction using CLP($\mathcal{BOOL}$).

Never-ending stories never end ...

Acknowledgments

I would like to thank the referees and the participants of LOPSTR'95 — and especially Alberto Pettorossi — for their helpful comments.

References

1. M. Bruynooghe, D. De Schreye, and B. Martens. A general criterion for avoiding infinite unfolding during partial deduction. *New Generation Computing*, 11(1):47–79, 1992.
2. P. Codognet and G. Filé. Computations, abstractions and contraints in logic programming. *Proc. of ICCL'92*, 1992.
3. A. Colmerauer. An introduction to Prolog III. *CACM*, 33 (7):70–90, July 1990.
4. S. Decorte, F. Stefaan, and D. De Schreye. Automatic inference of norms : a missing link in automation termination analysis. In *Logic Programming - Proceedings of 1993 International Symposium*, pages 420–436, 1993.
5. D. De Schreye and S. Decorte. Termination of logic programs: the never-ending story. *Journal of Logic Programming*, pages 1–199, 1993.
6. L. Fribourg and M. Veloso Peixoto. Bottom-up evaluation of datalog programs with arithmetic constraints: The case of 3 recursive rules. Technical report, L.I.E.N.S, France, 1994.
7. R. Giacobazzi and L. Ricci. Detecting determinate computations by bottom-up abstract interpretation. Technical report, Università di Pisa, 1992.
8. P. Hill and J. Lloyd. *The Gödel Programming Language*. MIT Press, 1994.
9. J. Jaffar and J.L. Lassez. Constraint logic programming. Technical Report 74, Monach University, Australia, 1986.
10. J. Jaffar and M.J. Maher. Constraint logic programming: a survey. *J. Logic Programming*, pages 503–581, 1994.

11. S. LÜETTRINGHAUS-KAPPEL. Control generation for logic programs. *Proc. of 10th ICLP*, pages 478–495, 1993.
12. B. MARTENS and D. DE SCHREYE. Automatic finite unfolding using well-founded measures. *Journal of Logic programming*, To appear.
13. B. MARTENS and J. GALLAGHER. Ensuring global termination of partial deduction while allowing flexible polyvariance. *Proc. of ICLP'95*, 1995.
14. F. MESNARD. *Etude de la terminaison des programmes logiques avec contraintes, au moyen d'approximations.* PhD thesis, Université Paris VI, 1993.
15. L. NAISH. Negation and control in prolog. In *LNCS 238*. Springer-Verlag, 1985.
16. L. PLÜMER. *Termination proofs fo logic programs.* Springer-Verlag, 1989.
17. B. SAGLAM and J.P. GALLAGHER. Approximating constraint logic programs using polymorphic types and regular descriptions. Technical report, University of Bristol, Dpt of Computer Science, 1995.
18. K. SOHN. Constraints among argument sizes in logic programs. *Proc. of PODS'94*, pages 68–74, 1994.
19. K. SOHN and A. VAN GELDER. Termination detection in logic programs using argument sizes. *Proc. of PODS'91*, pages 216–226, 1991.
20. J.D. ULLMAN and A. VAN GELDER. Efficient tests for top-down termination of logical rules. *Journal of the ACM*, 35(2):345–373, 1988.
21. K. VERSCHAETSE and D. DE SCHREYE. Deriving termination proofs for logic programs, using abstract procedures. *Proc. of the 8th ICLP*, pages 301–315, 1991.
22. K. VERSCHAETSE and D. DE SCHREYE. Derivation of linear size relations by abstract interpretation. In *LNCS 631*, pages 296–310. Springer Verlag, 1992.

A Proof of Theorem 10

Let us give a precise meaning of the dependency graph of a program.

Definition 11. Given a program P with π_P as its set of predicate symbols, we define its dependency graph D_P as the subset of $\pi_P \times \pi_P$ such that $(p, q) \in D_P$ iff there is a clause $p(\tilde{s}) \leftarrow \ \ldots, \diamond \ldots, q(\tilde{u}), \ldots$ in P. Let D_P^+ be the transitive closure of D_P.

Let P be a (specialized) CLP($\mathcal{Q}^+$) program. We assume that D_P is antisymmetric and that a valid measure μ_p is associated to each recursive predicate symbol p (i.e. such that $(p, p) \in D_P$). Let $ConstAt_P$ be the set of constraint atoms built from the vocabulary defined in P, i.e. $ConstAt_P = \{c \diamond p(\tilde{t}) | c \ is\ a\ \mathcal{Q}^+ - constraint,\ p(\tilde{t})\ a\ \pi_P - atom\}$. We have the following propositions:

Definition 12. A constraint atom $c \diamond p(\tilde{t})$ is bounded by $m \in \mathbb{Q}^+$ if m is the greatest lower bound of the set $\{\mu_p(\tilde{t}\theta) |\ \theta\ is\ a\ solution\ of\ c\}$.

Proposition 13. *Suppose* $c \diamond p(\tilde{t})$ *is bounded by* m. *Let* $p(\tilde{s}) \leftarrow c' \diamond \ldots, p(\tilde{u}), \ldots$ *be a clause of* P *defining* p. *If* $c, c', \tilde{t} = \tilde{s}$ *is solvable, then* $c, c', \tilde{t} = \tilde{s}' \diamond p(\tilde{u})$ *is bounded by* m' *and* $m \geq m' + 1$.

Definition 14. On $ConstAt_P$ we define a relation $>_P$:
$c \diamond p(\tilde{t}) >_P c' \diamond q(\tilde{s})$ if

$$\begin{cases} p \neq q, (p,q) \in D_P^+ \\ or \\ p = q, c \diamond p(\tilde{t}) \text{ bounded } m_1, c' \diamond p(\tilde{s}) \text{ bounded by } m_2, m_1 \geq m_2 + 1 \end{cases}$$

Proposition 15. *$\langle ConstAt_P, >_P\rangle$ is a partially ordered well-founded set.*

From now on, we stick to the concepts and definitions of [1] to prove the main result of the paper.

Theorem 10 *Let R be an extended computation rule. Then for any goal G, the extended computation tree $\tau_{G,R}$ is finite.*

Proof. Let $p_1, \ldots, p_N$ be the N recursive predicate symbols of P. We construct a *hierarchical prefounding* (see the definition in [1]) for $\tau_{G,R}$ as follows:

$$R_0 = \{A_i | \langle G, H\rangle \equiv \langle \leftarrow c \diamond A_1, \ldots, A_{n+1}, H\rangle \in \tau_{G,R}, \\ R(G,H) = \langle i, H'\rangle, i \geq 1, A_i = p(\tilde{t}), [(p,p) \notin D_p \textit{ or Bounded}(c, p(\tilde{t}))]\}$$

and for $1 \leq k \leq N$:

$$R_k = \{A_i | \langle G, H\rangle \equiv \langle \leftarrow c \diamond A_1, \ldots, A_{n+1}, H\rangle \in \tau_{G,R}, \\ R(G,H) = \langle i, H'\rangle, i \geq 1, A_i = p_k(\tilde{t}), [(p_k,p_k) \in D_p \textit{ and } \neg \textit{Bounded}(c, p_k(\tilde{t}))]\}$$

Note that:

1. We have a finite partition $R_0, \ldots, R_N$ of the set of selected goals in $\tau_{G,R}$. The classes $(C_0, C_1, C_2, \ldots)$ of resolvents from $\tau_{G,R}$ are defined as in the first part of the corresponding definition in [1].
2. Propositions 13 and 15 show that C_0 contains no infinite sequence of direct successors.
3. Let $\langle W_1, >_1\rangle = \ldots \langle W_N, >_N\rangle = \langle \mathbb{Q}^+, \geq +1\rangle$, which is a well-founded set. Let $\langle G, H\rangle \equiv \langle \leftarrow c \diamond A_1, \ldots, A_{n+1}, H \;\; \in \tau_{G,R}$ be an element of C_i such that $R(G,H) = \langle j, H'\rangle, i \geq 1, A_j = p_k(\tilde{t})$. We define $f_i : C_i \rightarrow \mathbb{Q}^+$ as $f_i(G,H) = Min(c; \mu_k(\tilde{t}))$.
4. For any element $\in C_m \cap C_n$ we have $f_m = f_n$.

To finish the proof, it remains to show that each f_i is monotonic. Let $\langle G; H\rangle$ the first element appearing in $\tau_{G,R}$ such that $\langle G; H\rangle \in C_i$. Let $G = \leftarrow c \diamond \tilde{C}, p(\tilde{s}), \tilde{D}$ where R chooses to unfold $p(\tilde{s})$. Let $p(\tilde{t}) \leftarrow c' \diamond \tilde{B}, p(\tilde{u}), \tilde{E}$ be a clause of P defining p. Then the node has as direct descendant: $\langle \leftarrow c, c', \tilde{s} = \tilde{t} \diamond \tilde{C}, \tilde{B}, p(\tilde{u}), \tilde{E}, \tilde{D}; H \cup \{\langle p(\tilde{s}), Min(c, \mu_p(\tilde{s}))\rangle\}\rangle$. Assume that $\langle G''; H''\rangle$ is the first descendant $\in C_i$ of $\langle G; H\rangle$. In general, we have $\langle G''; H''\rangle = \langle \leftarrow c, c', \tilde{s} = \tilde{t}, c'' \diamond \tilde{A}, p(\tilde{u}), \tilde{F}; H \cup \{\langle p(\tilde{s}), Min(c, \mu_p(\tilde{s}))\rangle\} \cup H'''\rangle$.

Let us show $f_i(G,H) \geq f_i(G'', H'') + 1$. On the one hand, we have:

$$Min(c; \mu_p(\tilde{s})) = Min(c, c', \tilde{s} = \tilde{t}; \mu_p(\tilde{s}))$$

$$= Min(c, c', \tilde{s} = \tilde{t}, c''; \mu_p(\tilde{s}))$$
$$= Min(c, c', \tilde{s} = \tilde{t}, c''; \mu_p(\tilde{t}))$$

because the unfolding from $\langle G; H \rangle$ to $\langle G'; H' \rangle$ then to $\langle G''; H'' \rangle$ are allowed.

On the other hand, recall that μ_p is a valid measure for p. Hence:

$$Min(c, c', \tilde{s} = \tilde{t}, c''; \mu_p(\tilde{t})) \geq 1 + Min(c, c', \tilde{s} = \tilde{t}, c''; \mu_p(\tilde{u}))$$

So we obtain:

$$Min(c; \mu_p(\tilde{s})) \geq 1 + Min(c, c', \tilde{s} = \tilde{t}, c''; \mu_p(\tilde{u}))$$

i.e.

$$f_i(G, H) \geq f_i(G'', H'') + 1$$

We conclude the proof by noting that in general, for the descendants $\in C_i$ of $\langle G; H \rangle$ we have, with $n \geq 1$:

$$Min(c; \mu_p(\tilde{s})) \geq 1 + n + Min(c, c', \tilde{s} = \tilde{t}, c'', c'''; \mu_p(\tilde{v}))$$

$\square$