Towards Automatic Control
for CLP(\chi) Programs

Fred Mesnard

Iremia, Université de la Réunion
15, avenue René Cassin - BP 7151 - 97715 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 \mathbb{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(\chi) to CLP(\mathbb{Q}^+)

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

Let \chi = \langle D_\chi; F_\chi; \{=, \leq \} \cup C_\chi \rangle where \text{\textup{\textit{D}_\chi}} is the domain of \chi, \text{\textup{\textit{F}_\chi}} is a set of functions and \text{\textup{\textit{C}_\chi}} a set of relations, be a solution-compact structure without any
limit element [10]. Let \( \psi = \langle D_\psi; F_\psi; \{\neg_\psi\} \cup C_\psi \rangle \) be another solution-compact structure.

**Definition 1.** An approximation \( A \) consists in a pair of functions \( \langle A_{ax}; A_{am} \rangle \) where:

1. \( A_{ax} \) is a mapping from \( F_\chi \cup \{\neg_\chi\} \cup C_\chi \) to \( F_\psi \cup \{\neg_\psi\} \cup C_\psi \) such that:
   
   (a) for every function or relation symbol \( s \), \( \text{arity}(s) = \text{arity}(A_{ax}(s)) \);
   
   (b) \( A_{ax}(F_\chi) \subseteq F_\psi \);
   
   (c) \( A_{ax}(\neg_\chi) = \neg_\psi \);
   
   (d) \( A_{ax}(C_\chi) \subseteq C_\psi \).

2. \( A_{am} \) is a mapping from \( D_\chi \) to \( D_\psi \) such that for every \( c_\chi \in \tilde{D}_\chi \):
   
   (a) for every function \( f_\chi \), \( A_{am}(f_\chi(c_\chi)) = \psi A_{ax}(f_\chi)(A_{am}(c_\chi)) \);
   
   (b) for every relation \( c_\chi \), if \( \models \chi \ c_\chi(c_\chi) \) then \( \models \psi \ A_{ax}(c_\chi)(A_{am}(c_\chi)) \).

**Example 1.** Let \( \text{List}(\chi) = \langle D_\chi^+; \{<>\} \cup \bigcup \in D_\chi \{<e>\} \cup \{\cdot\}; \{\neg \text{List}(\chi)\} \rangle \) 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 \( \text{List}(\chi) \) to \( Q^+ \):

\[ A_{ax}(<>) = 0, A_{ax}(<e>) = 1, A_{ax}(\cdot) = + \text{ and } A_{am}(<e_1, \ldots, e_n>) = n. \]

One verifies that \( A = \langle A_{ax}, A_{am} \rangle \) 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 \( A_{ax} \) on \( V \cup P \) with \( A_{ax}(x) = x \) (resp. \( A_{ax}(p) = p \)) 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(\( \text{List}(\chi) \)) program \( P \):

\[ \begin{align*}
\text{p}(<x_1, x_2, x_3, x_4, x_5>, <x_5, x_4, x_3, x_2, x_1 >, <>) & \leftarrow \diamond \\
\text{p}(<x_1, x_2 >, x, y, <y_1 >, <z>) & \leftarrow \diamond \text{p}(y, x, x, <x_1, y_1, x_2 >, z)
\end{align*} \]

Here is its approximation \( A(P) \) in CLP(\( Q^+ \)):

\[ \begin{align*}
\text{p}(5, 5, 0) & \leftarrow \diamond \\
\text{p}(x + 2, 2y + 1, z + 1) & \leftarrow \diamond \text{p}(y, 2x + 3, z)
\end{align*} \]

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 \( Q^+ \): termination in \( Q^+ \) implies termination in \( \text{List}(\chi) \).

**Theorem 2.** Let \( A \) be an approximation from CLP(\( \chi \)) to CLP(\( \psi \)), \( P \) a CLP(\( \chi \)) program and \( A(P) \) its approximation, \( G \) a goal and \( A(G) \) its approximation, \( R_2 \) a computation rule for CLP(\( \psi \)). If the \( (A(P), R_2 \circ A) \)-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($\chi$). 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($\phi$) may be longer and lead to a success node. A proof of Theorem 2 is given in [14].

A second justification for choosing $Q^+$ instead of $\mathcal{N} = (\langle \mathbb{N}, \{0, 1, +\}; \{-, \geq\}\rangle$ lies in the fact that many problems are much easier to solve in $Q^+$ 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 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) began 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: $\forall (x_1, \ldots, x_n) p(x_1, \ldots, x_n) \rightarrow \bigwedge_{1 \leq j \leq m} \left[ \sum_{1 \leq i \leq n} \epsilon_i x_i = k_j \right]$.

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

\[
\begin{align*}
p(\bar{x}) &\leftarrow \zeta(\bar{x}) \circ p(\bar{x}) \\
p(\bar{x} + \bar{a}) &\leftarrow \Phi_1(\bar{x}) \circ p(\bar{x}) \\
p(\bar{x} + \bar{b}) &\leftarrow \Phi_2(\bar{x}) \circ p(\bar{x}) \\
p(\bar{x} + \bar{c}) &\leftarrow \Phi_3(\bar{x}) \circ p(\bar{x})
\end{align*}
\]

where $\bar{a}$, $\bar{b}$ and $\bar{c}$ are vectors of integers, $\zeta(\bar{x}), \Phi_1(\bar{x}), \Phi_2(\bar{x})$ and $\Phi_3(\bar{x})$ 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 $P_{\text{spec}}$ 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. $M_P = M_{P_{\text{spec}}}$).

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 $D\rho$ (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 $M\rho$ of $P$, i.e. we need a link with the semantics of $p$.

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

**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: $\forall (x, y, z) \in (\mathbb{Q}^+)^3 \cdot 2x + 4 + 2y + 1 \geq 2y + 2x + 3 + 1$ 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 $\mu_p$ associated to a recursive predicate $p$ is $t$-valid (t for textually) if for each clause $p(\bar{t}) \leftarrow c \circ \bar{B}$ defining $p$ in $P$, for each solution $\theta$ of $c$, for each atom $p(s)$ appearing in $\bar{B}$, we have: $\mu_p(\bar{t}\theta) \geq \mu_p(s\theta) + 1$.

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

**Proposition 6.** If $\mu_p$ is $t$-valid for $p$, then $\mu_p$ is valid for $p$.

Of course, the converse is false.

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

$q(1) \leftarrow \circ \quad q(2x) \leftarrow \circ q(x)$

because $q(x) \in Mq$ implies $x \geq 1$ but $\mu_q$ 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:

$q(1) \leftarrow \circ \quad q(2x) \leftarrow x \geq 1 \circ q(x)$

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 \( \mu \) [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 \( P \):

\[
\begin{align*}
p(0) &\leftarrow \circ \circ r(x, y), p(y) \\
p(x) &\leftarrow x + u, 0 \geq u \geq 1 \circ \\
r(x + 1, y + 1) &\leftarrow \circ r(x, y)
\end{align*}
\]

Let us assume that we are interested in finding a valid measure for \( p \) and that we have inferred : \( \forall x \in \mathbb{Q}^+ \left[ p(x) \Rightarrow x \geq 0 \right] \) and \( \forall (x, y) \in (\mathbb{Q}^+)^2 \left[ r(x, y) \Rightarrow x \geq y + 1 \right] \).

We first specialize \( P \) into \( P_{\text{spec}} \):

\[
\begin{align*}
p(0) &\leftarrow \circ \circ r(x, y), p(y) \\
p(x) &\leftarrow x + y + 1 \circ r(x, y), p(y) \\
r(x + u, 0) &\leftarrow 3 \geq u \geq 1 \circ \\
r(x + 1, y + 1) &\leftarrow x \geq y + 1 \circ r(x, y)
\end{align*}
\]

Then we forget \( r \), which gives \( P' \):

\[
\begin{align*}
p'(0) &\leftarrow \circ \circ r'(y) \\
p'(x) &\leftarrow x \geq y + 1 \circ r'(y)
\end{align*}
\]

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

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(\( \mathbb{Q}^+ \))

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(\( \mathbb{Q}^+ \)). \( \text{Min}(c, t) \) denotes the least (rational) value of \( t \) in \( c \). \( \text{Bounded}(c, t) \) 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 $\pi$ of predicate symbols. Let $Rec$ be the subset of $\pi$ denoting the recursive procedures of $P$. Let $V$ be a countable set of variables. Let $At$ be the set of atoms defined using $\pi$, $\{0,1,+\}$ and $V$. A supervised atom is a pair $\langle \mu(t), m \rangle$ where $\mu(t) \in At$ and $m \in \Phi^+$. 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) $R : RES \times 2^{SupAt} \rightarrow \mathbb{N} \times 2^{SupAt}$ verifies:

1. $R(\neg c \circ Hist) = \langle 0, Hist \rangle$
2. $R(\neg c \circ A_1, \ldots, A_{n+1}, Hist) = \langle i, Hist' \rangle$ 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. $\langle G, \emptyset \rangle$ 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 $G' = \neg c \circ A_1, \ldots, A_i, \ldots, A_{n+1}$ and $R(G', H') = \langle i, H'' \rangle$ with $i \geq 1$ then the node has a child $\langle \neg c \circ A_1, \ldots, \hat{B}, \ldots, A_{n+1}, H'' \rangle$ for each clause of $P$: Head $\rightarrow c' \circ \hat{B}$ such that $c, c', A_i = \text{Head}$ is satisfiable. Moreover, each atom of $\hat{B}$ 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 $SelectAtom : RES \times 2^{SupAt} \rightarrow \mathbb{N}$ which has to satisfy the postcondition $0 \leq SelectAtom(- c \circ A_1, \ldots, A_k) \leq k$ (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($\mathbb{Q}^+$) 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 $\in NrU\cup C\cup B$, then we don't need to compute and check the history before we allow this choice (see the proof
ExtendedComputationRule($G, H$) =
let $i = \text{SelectAtom}(G, H)$
  in if $i = 0$ then $(0, H)$
  else
    let $G' := c \circ p_1(\hat{i}_1), \ldots, p_{k+1}(\hat{i}_{k+1})$
    $NR = \{ j \mid 1 \leq j \leq k+1, p_j \notin \text{Rec} \}$
    $R = \{ j \mid 1 \leq j \leq k+1, p_j \in \text{Rec} \}$
    $C = \{ j \mid j \in R, p_j(\hat{i}_j) \text{ is marked as checked } \}$
    $B = \{ j \mid j \in R \setminus C, \text{Bounded}(c, \mu_{p_j}(\hat{i}_j)) \}$
    $U = \{ j \mid j \in R \setminus (B \cup C) \}$
    $p_i(\hat{i}_n) \leftarrow c_1 \circ \ldots$
    $\vdots$
    $p_i(\hat{i}_m) \leftarrow c_n \circ \ldots$
    $H' = \{ \text{if } i \in U \text{ then } H \cup \{ (p_i(\hat{i}_i), \text{Min}(c, \mu_{p_i}(\hat{i}_i))) \} \}
    \text{else } H$
    $\text{stop} = \exists j, 1 \leq j \leq n, \exists q(\hat{s}), m \in H'$,
    $m < \text{Min}(c, j, \hat{i}_i = \hat{h}_j, \mu_q(\hat{s}))$
  in
    if $\text{stop}$ then $(0, H)$
    else $(i, H')$

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 $\text{stop}$ 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$(\chi)$

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

\[ \text{sort}(<>, <>) \leftarrow \circ \]
\[
\begin{align*}
\text{sort}(\langle x >, y, z \rangle) & \leftarrow \text{sort}(y, t), \text{insert}(t, x, z) \\
\text{insert}(\langle < >, x, < x > \rangle) & \leftarrow \circ \\
\text{insert}(\langle x >, y, w, < w, x >, y \rangle) & \leftarrow \ w \leq x \circ \\
\text{insert}(\langle x >, y, w, < x >, z \rangle) & \leftarrow \ w > x \circ \text{insert}(y, w, z)
\end{align*}
\]

First, we approximate it in CLP(\(Q^+\)):

\[
\begin{align*}
\text{sort}(0, 0) & \leftarrow \circ \\
\text{sort}(y + 1, z) & \leftarrow \text{sort}(y, t), \text{insert}(t, w, z) \\
\text{insert}(0, 1) & \leftarrow \circ \\
\text{insert}(y + 1, z, y + 2) & \leftarrow \circ \\
\text{insert}(y + 1, z, z + 1) & \leftarrow \text{insert}(y, w, z)
\end{align*}
\]

Second, inference of interargument relations may conclude that:

\[
\begin{align*}
\forall (y, z) \in (Q^+)^2 \text{insert}(y, z) & \Rightarrow z = y + 1 \\
\forall (x, y) \in (Q^+)^2 \text{sort}(x, y) & \Rightarrow x = y
\end{align*}
\]

We add the information to the program:

\[
\begin{align*}
\text{sort}(0, 0) & \leftarrow \circ \\
\text{sort}(y + 1, z) & \leftarrow z = t + 1, y = t \circ \text{sort}(y, t), \text{insert}(t, w, z) \\
\text{insert}(0, 1) & \leftarrow \circ \\
\text{insert}(y + 1, z, y + 2) & \leftarrow \circ \\
\text{insert}(y + 1, z, z + 1) & \leftarrow z = y + 1 \circ \text{insert}(y, w, z)
\end{align*}
\]

Third, we compute the valid measures for \text{sort} and \text{insert}:

\[
\begin{align*}
\mu^1_{\text{sort}}(x, y, z) & = (1, 0, 0) \quad \text{and} \quad \mu^2_{\text{sort}}(x, y, z) = (0, 0, 1) \\
\mu^1_{\text{insert}}(x, y, z) & = (1, 0) \quad \text{and} \quad \mu^2_{\text{insert}}(x, y, z) = (0, 1)
\end{align*}
\]

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. \(\mu^1_{\text{insert}}(x, y, z) = l\) where \(x :: l\) and using \(R_{\text{ext}}\) (see the end of Sect. 5) or by compiling the valid measures we found (such a compilation can be easily automated):

\[
\begin{align*}
\text{sort}(x, y) & \leftarrow x :: l, y :: l \circ \text{freeze}(l, \text{sort'}(x, y)) \\
\text{sort'}(\langle < >, < > \rangle) & \leftarrow \circ \\
\text{sort'}(\langle x >, y, z \rangle) & \leftarrow y :: l, l :: l, z :: l + 1 \circ \text{sort'}(y, t), \text{insert}(t, x, z) \\
\text{insert}(x, y, z) & \leftarrow x :: l, z :: l + 1 \circ \text{freeze}(l, \text{ms'}(x, y, z)) \\
\text{ms'}(\langle < >, x, < x > \rangle) & \leftarrow \circ \\
\text{ms'}(\langle x >, y, w, < w, x >, y \rangle) & \leftarrow \ w \leq x \circ \\
\text{ms'}(\langle x >, y, w, < x >, z \rangle) & \leftarrow \ y :: l, z :: l + 1, w > x \circ \text{ms'}(y, w, z)
\end{align*}
\]

The proof of any query about \text{sort} and \text{insert} on a CLP system with the \text{freeze} primitive will stop. For instance:
\[
> 1 \times y \circ \text{sort}(n, n), \text{insert}(l, x + 1, m), \\
\text{insert}(m, 4y, n), \text{sort}(l, < 2y, 3x + 1 >); \\
\{ 1 \times y < 2y, 2y \leq 3x + 1, n = < x + 1, 2y, 3x + 1, 4y >, \\
l = < 2y, 3x + 1 >, m = < x + 1, 2y, 3x + 1 > \}
\]

7 Discussion

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

1. We abstract \(P\) to \(A(P)\) in \(\text{CLP}(\mathbb{Q}^+)\).
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(\(\chi\)) 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 began 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 guard-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:

\[
\text{append}(\langle \rangle, y, y) \leftarrow \phi \\
\text{append}(\langle x_1 > x, y, < x_1 > z) \leftarrow \phi \text{append}(x, y, z)
\]

where a call to \(\text{append}(x, y, z)\) is delayed until \(\text{nonvar}(x)\) or \(\text{nonvar}(z)\). It is for instance the \text{DELAY} control declaration given for \text{append} in the module for lists processing of the Gödel programming language [8]. The proof of the goal \(G := \phi \text{append}(\langle x_1 > x, y, x)\) does not terminate.

Using our technique, we compute the interargument relation for the approximated version of \text{append}: \(\text{append}(x, y, z) \Rightarrow z = x + y\) and the two valid measures: \(\mu_1 \text{append} = (1, 0, 0)\) and \(\mu_2 \text{append} = (0, 0, 1)\). Then we compile this knowledge to obtain:
\begin{align*}
& \text{append}(x, y, z) \leftarrow x : l_x, y : l_y, z : l_z, l_x = l_x + l_y \diamond \\
& \quad \quad \text{free}(l_x, \text{once}(v, \text{append}(x, y, z))), \\
& \quad \quad \text{free}(l_z, \text{once}(v, \text{append}(x, y, z))) \\
& \text{append}(<>, y, y) \leftarrow \diamond \\
& \text{append}(< x_1 >, x, y, < x_1 >) \leftarrow \diamond \text{append}(x, y, z) \\
& \text{once}(v, G) \leftarrow \diamond \text{free}(v) \diamond \text{eq}(v, \text{cst}, G) \\
& \text{once}(v, G) \leftarrow \diamond
\end{align*}

Now the proof of the goal \( \diamond \text{append}(< x_1 >, x, y, x) \) 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(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{align*}
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{align*}

because the valid measure \( \mu_1^f(x, y, z) = 2x + y \) 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 \( \mu_1^f \) gives:

\begin{align*}
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); \{p(x + 7, y + 9, z_1 + 1, 23)\} \rangle \\
G_3: & \langle \leftarrow z = z_2 + 2 \diamond p(x + 6, y + 7, z_2); \{p(x + 7, y + 9, z_2 + 2, 23)\} \rangle \\
G_4: & \langle \leftarrow z = z_3 + 3 \diamond p(y/2 + 3, 2x + 11, z_3); \{p(x + 7, y + 9, z_3 + 3, 23)\} \rangle \\
G_5: & \langle \leftarrow z = z_4 + 4 \diamond p(x + 5, y + 5, z_4); \{p(x + 7, y + 9, z_4 + 4, 23)\} \rangle \\
G_6: & \langle \leftarrow x = 0, y = 0, z = 4 \diamond \{p(x + 7, y + 9, 4, 23)\} \rangle
\end{align*}

or

\begin{align*}
G_7: & \langle \leftarrow z = z_5 + 5 \diamond p(y/2 + 2, 2x + 9, z_5); \{p(x + 7, y + 9, z_5 + 5, 23)\} \rangle \\
G_8: & \langle \leftarrow z = z_6 + 6 \diamond p(x + 4, y + 3, z_6); \{p(x + 7, y + 9, z_6 + 6, 23)\} \rangle
\end{align*}
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: 
\[
\forall (x, y, z) \in (\mathbb{Q}^+)^3 p(x+7, y+9, z) \Leftrightarrow (x = y = 0 \land z = 4) \lor (p(x+4, y+3, z) \land z \geq 6)
\]
true in the least \( \mathbb{Q}^+ \)-model of the program. Or, if we only keep the deterministic part of the extended computation tree: 
\[
\forall (x, y, z) \in (\mathbb{Q}^+)^3 p(x+7, y+9, z) \Leftrightarrow p(x+5, y+5, z) \land z \geq 4.
\]

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 \( \chi \) to \( \mathbb{Q}^+ \) 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(BOOCL).

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


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 $\pi_P$ 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 $p(\bar{s}) \leftarrow \ldots \land c \leftarrow \ldots \land q(\bar{u})\ldots$ in $P$. Let $D_P^+$ be the transitive closure of $D_P$.

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

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

**Proposition 13.** Suppose $c \circ p(\bar{t})$ is bounded by $m$. Let $p(\bar{s}) \leftarrow c_1 \circ \ldots \land p(\bar{u})\ldots$ be a clause of $P$ defining $p$. If $c, c', \bar{t} = \bar{s}$ is solvable, then $c, c', \bar{t} = \bar{s}' \circ p(\bar{u})$ is bounded by $m'$ and $m \geq m' + 1$. 
Definition 14. On $\text{ConstAt}_P$ we define a relation $>_P$: $c \circ p(\tilde{t}) >_P c' \circ q(\tilde{s})$ if
\begin{align*}
&\{ p \neq q, (p, q) \in D_P^+ \\
&\text{or}
\}
\begin{align*}
&\{ p = q, c \circ p(\tilde{t}) \text{ bounded } m_1, c' \circ p(\tilde{s}) \text{ bounded by } m_2, m_1 \geq m_2 + 1
\end{align*}
\]

Proposition 15. $(\text{ConstAt}_P, >_P)$ 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 prefoundating (see the definition in [1]) for $\tau_{G, R}$ as follows:

\begin{align*}
R_0 &= \{ (G, H) \} \\
\text{and for } 1 \leq k \leq N: \hspace{1cm} R_k &= \{ (G, H) \} \\
R(G, H) &= \{ (i, H'), i \geq 1, A_i = p_k(\tilde{t}), ((p, k) \in D_P \text{ or } \neg \text{Bounded}(c, p(\tilde{t}))) \}
\end{align*}

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 $\langle W_1, >_1 \rangle = \ldots (W_N, >_N) = \{(\Phi^+, \geq +1)\}$, which is a well-founded set.
   Let $\langle G, H \rangle = \langle \neg c \circ A_1, \ldots, A_{n+1}, H \rangle \in \tau_{G, R}$ be an element of $C_i$ such that $R(G, H) = \{ (j, H'), i \geq 1, A_j = p_k(\tilde{t}) \}$. We define $f_i : C_i \rightarrow \Phi^+$ as $f_i(G, H) = \text{Min}(c; p_k(\tilde{t}))$.
4. For any element in $C_n \cap C_n$ 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 $G = \neg c \circ C, p(\tilde{s}), \tilde{t}$ where $R$ chooses to unfold $p(\tilde{s})$. Let $\tilde{p}(\tilde{t}) \leftarrow c' \circ \tilde{B}, p(\tilde{u}), \tilde{E}$ be a clause of $P$ defining $p$. Then the node has as direct descendant: $\langle \neg c, c', s = \tilde{t} \circ \tilde{C}, \tilde{B}, p(\tilde{u}), \tilde{E}, \tilde{D}; H \cup \{ p(\tilde{s}), \text{Min}(c, p(\tilde{s})) \} \rangle$. Assume that $\langle G', H' \rangle$ is the first descendant in $C_i$ of $\langle G, H \rangle$. In general, we have $\langle G', H' \rangle = \langle \neg c, c', s = \tilde{t} \circ \tilde{A}, p(\tilde{u}), \tilde{F}; H \cup \{ p(\tilde{s}), \text{Min}(c, p(\tilde{s})) \} \rangle$. Let us show $f_i(G, H) \geq f_i(G', H' \rangle + 1$. On the other hand, we have:

$$\text{Min}(c; p(\tilde{s})) = \text{Min}(c, c', s = \tilde{t} \circ p(\tilde{s}))$$
\[ = \text{Min}(c, c', \tilde{s} = \tilde{l}, c''; \mu_p(\tilde{s})) \]
\[ = \text{Min}(c, c', \tilde{s} = \tilde{l}, c''; \mu_p(\tilde{l})) \]

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 \( \mu_p \) is a valid measure for \( p \). Hence:
\[
\text{Min}(c, c', \tilde{s} = \tilde{l}, c''; \mu_p(\tilde{l})) \geq 1 + \text{Min}(c, c', \tilde{s} = \tilde{l}, c''; \mu_p(\tilde{l}))
\]

So we obtain:
\[
\text{Min}(c; \mu_p(\tilde{s})) \geq 1 + \text{Min}(c, c', \tilde{s} = \tilde{l}, c''; \mu_p(\tilde{l}))
\]

i.e.
\[
f_1(G, H) \geq f_2(G'', H'') + 1
\]

We conclude the proof by noting that in general, for the descendants \( \in C_i \) of \( \langle G; H \rangle \) we have, with \( n \geq 1 \):
\[
\text{Min}(c; \mu_p(\tilde{s})) \geq 1 + n + \text{Min}(c, c', \tilde{s} = \tilde{l}, c''; \mu_p(\tilde{l}))
\]

\[\square\]