Inferring Left-terminating Classes of Queries for Constraint Logic 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

This paper presents an approach for universal left-termination of constraint logic programs, based on approximations. An approximation is basically an algebraic morphism between two constraint structures. By moving from the original domain to natural numbers, we compute inter-argument relations and some control information about a program. By moving from the natural numbers to the booleans, we compute a boolean term called a termination condition such that if the boolean approximation of a goal entails the termination condition, then the Prolog computation tree for that goal is finite.
Keywords: termination, analysis, constraint logic programming.

1 Introduction

Suppose that we want to write a constraint logic procedure $som3(a, b, c)$ where its third argument $c = [a_0 + b_0, \ldots, a_i + b_i, \ldots]$ is the sum of its two first arguments $a = [a_0, \ldots, a_i, \ldots]$ and $b = [b_0, \ldots, b_i, \ldots]$ and then $som4(a, b, c, d)$ where the fourth argument is the sum of its three first arguments.

Example 1.1 Here are a solution for $som3$ and two possibilities for $som4$:

$som3([\,], B, B).$
$som3([A1|A], [\,], [A1|A]).$
$som3([A1|A], [B1|B], [A1 + B1|C]) : -som3(A, B, C).$

$som41(A, B, C, D) : -som3(A, B, E), som3(E, C, D).$
$som42(A, B, C, D) : -som3(E, C, D), som3(A, B, E).$

After compiling this declarative knowledge on our favorite constraint logic programming (CLP) system, we would like to use it. As we know that CLP can prove theorems of the form $\exists \tilde{x}\,[c \wedge p(\tilde{x})]$, the following sample of goals does make sense:

$:-som3([1,2],B,[3,4,5|B])$. $:-som3(A,B,[3,4,5|C])$.
$:-som41(A,B,C,[1,2,3,4])$. $:-som41([0,0],[0,0,0],C,C)$.
$:-som42(A,B,A,[1,2,3,4])$. $:-som42([0,1],[2,3],C,D)$.

Programming with a high level language, we do not want to be involved in low-level details about the computation. Unfortunately, some disappointment will quickly happen ...

We propose an approach which allows the CLP system to compute for each predicate p a boolean term TC_p s.t. if the boolean approximation of the goal G entails TC_p, then universal termination of G under the Prolog left-to-right computation rule is guaranteed (we say that G *left-terminates*).

We organize the paper as follows. After a few preliminaries, section 3 introduces what we called an approximation: it is essentially an algebraic morphism between two constraint structures associated to a corresponding syntactical mapping. Sections 4 and 5 present known techniques to statically infer information from a given program which help to control a proof. Then we compound all the gathered information in section 6 to show how to compute boolean termination conditions. Finally, section 7 discussed the results and related works.

Due to space restriction, we omit the proofs (see [12]) but present instead numerous examples which may help to lay down the intuitions.

2 Preliminaries

We try to stick to the notations and the conventions introduced in [10]. We only consider structures without limit element and ideal CLP systems. $\tilde{x}$ denotes a sequence of distinct variables. Let o be a CLP(χ) object. The set of variables of o is denoted $var(o)$ and $o(\tilde{x})$ means $o \quad where \quad var(o) = \tilde{x}$. We say that θ is a *solution* of the constraint c if θ is a valuation s.t. $\chi \models c\theta$. Let c_1 and c_2 be two constraints. We write $c_1 \rightarrow c_2$ as a shorthand for $\chi \models \forall [c_1 \rightarrow c_2]$.

For sake of simplicity, we disallow mutually recursive procedures (i.e. we assume that the transitive closure of the dependency graph of the program is anti-symmetric). However, the results presented in the paper can be lifted to mutual recursivity. A *recursive* rule is a rule of the form $p(\tilde{x}) \leftarrow \ldots, p(\tilde{y}), \ldots$. A predicate p is *recursive* if there is at least one recursive rule defining p.

3 From CLP(χ) to CLP($\mathcal{B}$) via CLP($\mathcal{N}$)

Let $\mathcal{N}$ denote the structure $\langle \mathbb{N}; \{0,1,+\}; \{=,\geq\} \rangle$ where $\mathbb{N}$ is the set of natural numbers. The first step in the approach we propose is to switch from χ to $\mathcal{N}$ by using an approximation $\mathcal{A}_\chi^{\mathcal{N}}$ which consists in an algebraic morphism A_{sm} associated to a syntactic transformation A_{sx}, capturing a notion of size for the elements of χ.

More formally, 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. Let $\psi = \langle D_\psi; F_\psi; \{=_\psi\} \cup C_\psi \rangle$ be another solution-compact structure.

Definition 3.1 *An* approximation $\mathcal{A}^\psi_\chi$ *from* χ *to* ψ *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* $\chi \models c_\chi(\widetilde{e_\chi})$ *then* $\psi \models A_{sx}(c_\chi)(A_{sm}(\widetilde{e_\chi}))$.

We note that we can naturally generalize the above definition to many-sorted structures, define the identity element and the composition of approximations. We now present three approximations.

Example 3.1 A data structure $\mathcal{S}D$ is often related to $\mathbb{N}$ by means of a mapping $size$ from $D_{\mathcal{S}D}$ to $\mathbb{N}$, which may generate a particular approximation. In the case of lists, let $\mathcal{L}(\chi) = \langle {D_\chi}^*; \{[\,]\} \cup \bigcup_{e \in D_\chi} \{[e]\} \cup \{\cdot\}; \{=_{\mathcal{L}(\chi)}\} \rangle$ where the constant $[\,]$ denotes the empty list, the constants $[e]$ denote the lists $[e]$ and the dot operator "$\cdot$" is the concatenation of lists. We define the approximation $\mathcal{A}^{\mathcal{N}}_{\mathcal{L}(\chi)}$ from $\mathcal{L}(\chi)$ to $\mathcal{N}$: $A_{sx}([]) = 0$, $A_{sx}([e]) = 1$, $A_{sx}(\cdot) = +$ and $A_{sm}([e_1, \ldots, e_n]) = n$.

Example 3.2 For trees, let $\mathcal{T}(\chi) = \langle T_{D_\chi}; \{empty\} \cup \bigcup_{e \in D_\chi} \{t_e\}; \{=_{\mathcal{T}(\chi)}\} \rangle$ where the constant $empty$ denotes the empty tree and the binary function symbols t_e denote the applications $(l, r) \mapsto tree(l, e, r)$. The usual $size$ function defined by $size(empty) = 0$ and $size(tree(l, x, r)) = 1 + size(l) + size(r)$ leads naturally to an approximation from $\mathcal{T}(\chi)$ to $\mathcal{N}$. But more "exotic" norms (where the size of a term is defined by a linear combination of the size of its components) can give rise to approximations, too. From the function $size$ defined by $size(empty) = 0$ and $size(tree(l, x, r)) = 1 + size(r)$, we construct the approximation $\mathcal{A}^{\mathcal{N}}_{\mathcal{T}(\chi)}$ from $\mathcal{T}(\chi)$ to $\mathcal{N}$: $A_{sx}(empty) = 0$, $A_{sx}(t_e) = f$ and $A_{sm}(t) = size(t)$ where f is the function symbol associated to the function $f : (n, p) \mapsto 1 + p$. At first sight, condition 1.b is not verified. We can however simplify any "pseudo-arithmetical" term t containing the symbol function f to an equivalent arithmetical term s (e.g. $f(0, f(1, x)) 1 + (1 + x)$).

Remark 3.1 From now on, we assume that to each structure $\mathcal{SD}$ is associated the structure $\mathcal{N}$ with a function *size* noted $|\,|_{\mathcal{SD}}$, mapping elements of $D_{\mathcal{SD}}$ to their sizes. When we approximate $\mathcal{SD}$ to $\mathcal{N}$ wrt *size*, the function $|\,|_{\mathcal{SD}}$ becomes the identity function $n \mapsto n$.

Example 3.3 Let $\mathcal{B} = \langle\ \{true, false\}\ ;\ \{0, 1, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}\ ;\ \{\Rightarrow, =_{\mathcal{B}}\}\rangle$ where the symbols have their usual interpretation. We define the approximation $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}$ from $\mathcal{N}$ to $\mathcal{B}$: $A_{sx}(0) = 1$, $A_{sx}(1) = 1$, $A_{sx}(+) = \wedge$, $A_{sx}(\geq) = \Rightarrow$ and $A_{sm}(n) = true$.

Now we would like to apply the approximation $\mathcal{A}^{\psi}_{\chi}$ to CLP(χ) entities like terms, programs, goals. Let V be a denumerable set of variables and Π be a finite set of predicate symbols. The set of *terms*, *constraints*, *atoms*, *facts*, *rules*, *programs*, *goals* are defined as usual. If we extend A_{sx} on $V \cup \Pi$ with $A_{sx}(x) = x$ (resp. $A_{sx}(p) = p$) for every variable x (resp. for every predicate symbol p), then $\mathcal{A}^{\psi}_{\chi}$ may be naturally extended on terms, constraints, atoms, sets of atoms, facts, sets of facts, rules, programs and goals.

Example 3.4 For the program $P^{\mathcal{L}(\chi)}_{som3}$ of example 1.1, here are $P^{\mathcal{N}}_{som3} = \mathcal{A}^{\mathcal{N}}_{\mathcal{L}(\chi)}(P^{\mathcal{L}(\chi)}_{som3})$ and $P^{\mathcal{B}}_{som3} = \mathcal{A}^{\mathcal{B}}_{\mathcal{N}}\ (P^{\mathcal{N}}_{som3})$:

$som3(0, B, B).$
$som3(1 + A, 0, 1 + A).$
$som3(1 + A, 1 + B, 1 + C) :-$
$\quad som3(A, B, C).$

$som3(1, B, B).$
$som3(1 \wedge A, 1, 1 \wedge A).$
$som3(1 \wedge A, 1 \wedge B, 1 \wedge C) :-$
$\quad som3(A, B, C).$

We present some properties of approximations. Let $\mathcal{A}^{\psi}_{\chi}$ be an approximation from χ to ψ and P a CLP(χ) program. From a (s-)semantical point of view [8], the image of the semantics of P is included in the semantics of the image of P:

Theorem 3.1 $\mathcal{A}^{\psi}_{\chi}\ (lfp(S^{\chi}_{P})) \subseteq lfp(S^{\psi}_{\mathcal{A}^{\psi}_{\chi}(P)})$

Example 3.5 In $\mathcal{N}$, we have: $\forall[som3(x, y, z) \leftrightarrow z = max(x, y)]$ and for instance $som3(2, 3, 3)$ is true wrt $P^{\mathcal{N}}_{som3}$. In $\mathcal{B}$, we have: $\forall[som3(x, y, z) \leftrightarrow (z \Leftrightarrow x \wedge y) \wedge (x \vee y)]$ and we verify that $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}\ (som3(2, 3, 3)) = som3(1, 1, 1)$ is true wrt $P^{\mathcal{B}}_{som3}$.

But the information we get is richer. Intuitively, we may add ∞ to $\mathbb{N}$ and extend the operations and relations accordingly. Call $\bar{\mathcal{N}}$ the resulting structure. The image of ∞ in $\mathcal{B}$ would be 0 and the reader can verify that we still have an approximation. Now, for every natural number n, $som3(n, \infty, \infty)$ is true wrt $P^{\bar{\mathcal{N}}}_{som3}$ and its approximation $som3(1, 0, 0)$ is true.

Moreover, it is often difficult to compute the semantics of a program P^{χ} in its original structure χ. But if the domain of ψ is finite, then the semantics of $\mathcal{A}^{\psi}_{\chi}(P)$ is computable, which may help to estimate the meaning

of P^χ. Suppose we wonder whether $som3(\infty,\infty,\infty)$ is true wrt $P^{\mathcal{N}}_{som3}$. As $som3(0,0,0)$ is false wrt $P^{\mathcal{B}}_{som3}$, we conclude, with the help of theorem 3.1 that $som3(\infty,\infty,\infty)$ is false wrt $P^{\mathcal{N}}_{som3}$. Indeed the meaning of $P^{\mathcal{N}}_{som3}$ is $\forall(x,y,z)\in(\mathbb{N}\cup\{\infty\})^3\ [som3(x,y,z)\leftrightarrow z=max(x,y)\wedge(x\in\mathbb{N}\vee y\in\mathbb{N})]$.

The approximation $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}$ enjoys the following property: if the boolean image of a numeric constraint c implies that a variable x is set to 1, then the set of possible values for x in c has a least upper bound (we say that x *is bounded in* c).

Proposition 3.1 *Let $c_1^{\mathcal{N}}$ be a numeric constraint with $c_1^{\mathcal{B}}=\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}\ (c_1^{\mathcal{N}})$ and $t=\vee_{j\in J}[\wedge_{i\in I_j}x_i]$ a boolean term. If $c_1^{\mathcal{B}}\rightarrow t$ then $\exists j\in J,\ \forall i\in I_j,\ x_i$ is bounded in $c_1^{\mathcal{N}}$.*

Let us go back to our main subject: termination. The following result justifies the switch to $\mathcal{N}$: left-termination in $\mathcal{N}$ implies left-termination in χ. More generally, let $\mathcal{A}^{\psi}_{\chi}$ be an approximation from χ to ψ, P^χ a CLP(χ) program and G^χ a CLP(χ) goal. Computation trees are constructed using the Prolog left-to-right computation rule.

Theorem 3.2 *If the $\mathcal{A}^{\psi}_{\chi}\ (P^\chi)$-computation tree for $\mathcal{A}^{\psi}_{\chi}\ (G^\chi)$ is finite, then the P^χ-computation tree for G^χ is finite.*

We note that because of condition 2 of definition 3.1, a branch of the P^χ-computation tree for G^χ may lead to a failure node, though its corresponding branch in CLP(ψ) may be longer and lead to a success node. So the length of each branch of P^χ-computation tree for G^χ is "smaller or equal" than its corresponding branch in CLP(ψ).

For the structures $\mathcal{N}$ and $\mathcal{B}$, we have a result which establishes a link between derivations in CLP($\mathcal{N}$) and computations in CLP($\mathcal{B}$).

Remark 3.2 First, we observe that for any program $P^{\mathcal{B}}$ of CLP($\mathcal{B}$) and without loss of generality, we can assume that the least fixpoint $lfp(S^{\mathcal{B}}_P)$ contains exactly one fact $p(\tilde{x})\leftarrow c^{\mathcal{B}}_{p,success}(\tilde{x})$ for each predicate symbol p of $P^{\mathcal{B}}$.

Proposition 3.2 *Let $P^{\mathcal{N}}$ be a CLP($\mathcal{N}$) program and $\leftarrow c_2^{\mathcal{N}}$ the answer constraint of* any *derivation of the goal $\leftarrow c_1^{\mathcal{N}}, p(\tilde{x})$. Let $c_3^{\mathcal{B}}$ be a boolean constraint and $p(\tilde{x})\leftarrow c^{\mathcal{B}}_{p,success}\in lfp(S^{\mathcal{B}}_{\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}(P^{\mathcal{N}})})$. If $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}(c_1^{\mathcal{N}})\rightarrow c_3^{\mathcal{B}}$ then $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}(c_2^{\mathcal{N}})\rightarrow c_3^{\mathcal{B}}\wedge c^{\mathcal{B}}_{p,success}$.*

Before we give an example, we point out that proposition 3.2 holds from CLP(χ) to CLP(ψ) as soon as CLP(ψ) fulfills the condition of remark 3.2.

Example 3.6 Let n be a natural number and consider the classes of goals $G^{\mathcal{N}}_n$ defined by $\leftarrow n\geq z, som3(x,y,z)$. We have: $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}\ (n\geq z)=1\Rightarrow z$.

Moreover, we know (example 3.5) that $[som3(x,y,z) \leftarrow (z \Leftrightarrow x \wedge y) \wedge (x \vee y)]$ characterizes the meaning of $P^{\mathcal{B}}_{som3}$.

So by proposition 3.2, for *any* answer constraint $\leftarrow c^{\mathcal{N}}$ of $G^{\mathcal{N}}_n$, we have: $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}(c^{\mathcal{N}}) \rightarrow (1 \Rightarrow z) \wedge (z \Leftrightarrow x \wedge y) \wedge (x \vee y)$, i.e. $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}(c^{\mathcal{N}}) \rightarrow x \wedge y \wedge z$. Applying proposition 3.1, it comes: for all answer constraint $\leftarrow c^{\mathcal{N}}$ of $G^{\mathcal{N}}_n$, $\exists n'$ s.t. $c^{\mathcal{N}} \rightarrow n' \geq max(x,y,z)$.

Now we note that the computation tree for $G^{\mathcal{N}}_n$ is finite as the third argument of $som3(x,y,z)$, bounded by n, decreases at each recursive calls. Hence: $\exists n''$, s.t. for all answer constraint $\leftarrow c^{\mathcal{N}}$ of $G^{\mathcal{N}}_n$, $c^{\mathcal{N}} \rightarrow n'' \geq max(x,y,z)$.

4 Inference of Interargument Relations

Computing information about the relationship between the sizes of the arguments of a predicate is essential for automatic termination analysis. In the logic programming framework, the research devoted to the derivation of interargument relations (IR) begun with [18] and is summarized in [6].

Remark 4.1 From a theoretical point of view, such IR's should be computed in CLP($\mathcal{N}$) in order to obtain precise results. However, from a practical point of view, CLP($\mathcal{Q}^+$) often seems to be a best choice (where $\mathcal{Q}^+ = \langle \mathbb{Q}^+; \{0,1,+\}; \{=,\geq\} \rangle$ with $\mathbb{Q}^+$ denoting the set of non-negative rational numbers). First, many problems are much easier to solve in $\mathcal{Q}^+$ than in $\mathcal{N}$ (e.g. linear programming). Second, if a property of the form $\forall[p(\tilde{x}) \rightarrow c(\tilde{x})]$ is true in CLP($\mathcal{Q}^+$), then it is true in CLP($\mathcal{N}$).

Let us now mention some of the main results obtained so far. In [20], the authors show how one may compute by abstract interpretation IR's of the form: $\forall \left[p(x_1,\ldots,x_n) \rightarrow \bigwedge_{1 \leq j \leq m} \sum_{1 \leq i \leq n} c^j_i x_i = k^j\right]$.

In [7], the class of "3-recursive" CLP($\mathcal{Z}$) logic procedures is defined as follows:

$$
\begin{array}{l}
p(\tilde{x}) \leftarrow \theta(\tilde{x}) \\
p(\tilde{x}+\tilde{a}) \leftarrow \phi_1(\tilde{x}), p(\tilde{x}) \\
p(\tilde{x}+\tilde{b}) \leftarrow \phi_2(\tilde{x}), p(\tilde{x}) \\
p(\tilde{x}+\tilde{c}) \leftarrow \phi_3(\tilde{x}), p(\tilde{x})
\end{array}
$$

where $\tilde{a}$, $\tilde{b}$ and $\tilde{c}$ are vectors of integers, $\theta(\tilde{x})$ is a linear arithmetical constraint, $\phi_j(\tilde{x})$ for $1 \leq j \leq 3$ are linear arithmetic constraints of the form $\sum_{1 \leq i \leq 3} d^j_i x_i \geq e^j$. Such a procedure p can be *exactly* characterized by a finite arithmetic constraint. The authors have enumerated all the 512 (equivalent) 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. Moreover, if a 3-recursive logic procedure is detected in a CLP($\mathcal{N}$) program, it can be deleted and replaced by its characterization, except for 4 cases where the equivalent arithmetic constraint is *not* linear.

Remark 4.2 From now on, we require that the inferred IR's are added as constraints (cf. remark 3.1) to the original program P, which leads to a specialized version P' of P. But as the IR's are logical consequences of the least model of P, such a specialization preserves the meaning of P.

Example 4.1 Here is the *reverse* predicate, directly translated from Prolog to CLP($\mathcal{L}(\chi)$) on the left, and its CLP($\mathcal{N}$) version on the right.

$$
\begin{array}{ll}
reverse(X,Y) : -rev(X,[\,],Y). & reverse(X,Y) : -rev(X,0,Y). \\
rev([\,],Y,Y). & rev(0,Y,Y). \\
rev([X1|X],Y,Z) : - & rev(1+X,Y,Z) : - \\
\quad rev(X,[X1|Y],Z). & \quad rev(X,1+Y,Z).
\end{array}
$$

Specializing those programs wrt to the IR's: $\forall[rev(x,y,z) \rightarrow z = x + y]$ and $\forall[reverse(x,y) \rightarrow x = y]$, we obtain :

$$
\begin{array}{ll}
reverse(X,Y) : - & reverse(X,Y) : - \\
\quad \mid X \mid = \mid Y \mid, & \quad X = Y, \\
\quad rev(X,[\,],Y). & \quad rev(X,0,Y). \\
rev([\,],Y,Y). & rev(0,Y,Y). \\
rev([X1|X],Y,Z) : - & rev(1+X,Y,Z) : - \\
\quad \mid Z \mid = \mid X \mid +1+ \mid Y \mid, & \quad Z = X + 1 + Y, \\
\quad rev(X,[X1|Y],Z). & \quad rev(X,1+Y,Z).
\end{array}
$$

5 Inference of Control Information

A very basic idea for termination relies on associating to every recursive predicate a weight which decreases at each recursive call. We formalize this concept and adapt it to our approach.

Let $(W, \succ)$ be a (partially) strictly ordered (i.e. $\succ$ is a anti-reflexive, anti-symmetric and transitive binary relation) well-founded (i.e. there is no infinite sequence of elements $w_1, w_2, \ldots$ in W s.t. $\forall i \geq 1, w_i \succ w_{i+1}$) set. Let p be a n-ary recursive predicate symbol of a CLP($\mathcal{N}$) program P and μ be a mapping from $\mathbb{N}^n$ to W (called a *measure*). Let c be a linear arithmetic constraint. We say that $\mu(\tilde{x})$ is *bounded in* c if there exists $w \in W$ s.t. for each solution θ of c, if $\mu(\tilde{x}\theta)$ is defined then $w \succeq \mu(\tilde{x}\theta)$.

In proposition 3.1, we have seen that we can check that some arithmetical terms are bounded: we extend this idea on μ. We also need a link between μ and p. Although possible (see [13] for a more involved discussion about measures, including their use for the local control of partial deduction), it would be too exacting to ask for a purely semantical link. We content ourselves to the text of p. However, note that we are working (cf remark 4.2) with a specialized version of the original program.

Definition 5.1 *A measure μ is* t-valid wrt p *(t for textually) if there exists a non-empty set $I_\mu \subseteq \{1, \ldots, n\}$ s.t.*

1. *for each arithmetical constraint c,*
   $[\forall i \in I_\mu, x_i$ *is bounded in* $c] \rightarrow \mu(\tilde{x})$ *is bounded in* c.

2. *for each recursive rule* $p(\tilde{x}) \leftarrow c, \tilde{B}$ *defining* p *in* P,
for each solution θ *of* c,

   (a) $\mu(\tilde{x}\theta)$ *is defined,*

   (b) *for each atom* $p(\tilde{y})$ *appearing in* $\tilde{B}$, $\mu(\tilde{y}\theta)$ *is defined and* $\mu(\tilde{x}\theta) \succ \mu(\tilde{y}\theta)$.

Example 5.1 Consider the CLP($\mathcal{N}$) version of the Ackerman function (on the left) and on the right its specialized version P'_{ack} wrt the IR: $\forall[ack(x,y,z) \rightarrow z \geq x+y+1]$

$$
\begin{array}{ll}
ack(0,Y,Y+1). & ack(0,Y,Y+1). \\
ack(X+1,0,Z) :- & ack(X+1,0,Z) :- \\
\quad ack(X,1,Z). & \quad Z \geq X+2, ack(X,1,Z). \\
ack(X+1,Y+1,Z) :- & ack(X+1,Y+1,Z) :- \\
\quad ack(X+1,Y,T), & \quad T \geq X+Y+2, Z \geq X+T+1, ack(X+1,Y,T), \\
\quad ack(X,T,Z). & \quad ack(X,T,Z).
\end{array}
$$

If we take $(W,\succ) = (\mathbb{N},>)$ then $\mu^1(x,y,z) = 2z - y$ with $I_{\mu^1} = \{3\}$ is a t-valid measure for P'_{ack}. First, we verify that condition 1 of definition 5.1 is fulfilled. Assume that z is bounded by a in c. Then it is clear that for each solution θ of c, $2a \geq 2z\theta \geq (2z-y)\theta$. So either $0 > (2z-y)\theta$, hence $\mu^1(x,y,z)\theta$ is not defined or $2a \geq (2z-y)\theta \geq 0$. We conclude that $\mu^1(x,y,z)$ is bounded in c. Second, we check condition 2 of definition 5.1. For the second rule of P'_{ack}, let θ be a solution of $z \geq x+2$. We verify that $2z\theta$ and $(2z-1)\theta \geq 3$ are defined and $2z\theta > (2z-1)\theta$. For the third rule of P'_{ack}, let θ be a solution of $t \geq x+y+2 \wedge z \geq x+t+1$. We verify that $(2z-y-1)\theta \geq 3$, $(2t-y)\theta \geq 4$, $(2z-t)\theta \geq 2$ are defined. At last, $(2z-2t)\theta \geq 2$ implies $(2z-(y+1))\theta > (2t-y)\theta$ and $t\theta \geq (y+2)\theta$ implies $(2z-(y+1))\theta > (2z-t)\theta$.

If we take $(W,\succ) = (\mathbb{N}^2, >_{lx})$ where $>_{lx}$ denotes the lexicographic order on $\mathbb{N}^2$, then $\mu^2(x,y,z) = \langle x,y\rangle$ with $I_{\mu^2} = \{1,2\}$ is a t-valid measure for P'_{ack}. First, we check that condition 1 of definition 5.1 is fulfilled. Assume that x (resp. y) is bounded by a (resp. b) in c. It is clear that $\langle a,b\rangle$ bounds $\langle x\theta, y\theta\rangle$ for each solution θ of c. Second, we check condition 2 of definition 5.1. For the second rule of P'_{ack}, let θ be a solution of $z \geq x+2$. We have $\langle (x+1)\theta, 0\rangle >_{lx} \langle x\theta, 1\rangle$. For the third rule, let θ be a solution $t \geq x+y+2 \wedge z \geq x+t+1$. We have $\langle (x+1)\theta, (y+1)\theta\rangle >_{lx} \langle (x+1)\theta, y\theta\rangle$ and $\langle (x+1)\theta, (y+1)\theta\rangle >_{lx} \langle x\theta, t\theta\rangle$.

Example 5.2 For the *rev* predicate of example 4.1, if we take $(W,\succ) = (\mathbb{N},>)$ then $\mu^1(x,y,z) = (1,0,0)$ and $\mu^2(x,y,z) = (0,-1,1)$ with $I_{\mu^1} = \{1\}$ and $I_{\mu^2} = \{3\}$ are two t-valid measures.

There is an important class of measures which can be automatically inferred.

Definition 5.2 *A* p-linear measure *(p for positive)* $\mu = (\mu_1, \ldots, \mu_n)$ *is a mapping from* $I\!N^n$ *to* $I\!N$ *such that: the* μ_i*'s are natural numbers and* $\forall (a_1, \ldots, a_n) \in I\!N^n \ \mu(a_1, \ldots, a_n) = \sum_{i=1}^{n} \mu_i a_i$.

For p-linear measures, we can give an equivalent (wrt definition 5.1) definition for t-validity:

Definition 5.3 *A p-linear measure* μ_p *associated to a recursive predicate* p *is* t-valid *if for each rule* $p(\tilde{t}) \leftarrow c \diamond \tilde{B}$ *defining* p *in* P*, for each solution* θ *of* c*, for each atom* $p(\tilde{s})$ *appearing in* $\tilde{B}$*, we have:* $\mu_p(\tilde{t}\theta) \geq 1 + \mu_p(\tilde{s}\theta)$.

Note that we take $(W, \succ) = (\mathbb{N}, \geq +1)$ and that for a t-valid p-linear measure (tvpl-measure) $\mu = (\mu_1, \ldots, \mu_n)$, the set $\{i \mid \mu_i > 0\}$ defines the associated set I_μ which supports μ.

Example 5.3 Consider the specialized CLP($\mathcal{N}$) version of the example 1.1 using the IR: $\forall[som3(x, y, z) \rightarrow 2x + 2y \geq 2z \geq x + y]$

$som3(0, B, B).$
$som3(A + 1, 0, 0).$
$som3(A + 1, B + 1, C + 1) : -$
$\quad 2A + 2B \geq 2C \geq A + B, som3(A, B, C).$

$som41(A, B, C, D) : -$
$\quad S = \mid A \mid + \mid B \mid, T = \mid C \mid + \mid E \mid, 2S \geq 2E \geq S, 2T \geq 2D \geq T,$
$\quad som3(A, B, E), som3(E, C, D).$
$som42(a, b, c, d) : -$
$\quad S = \mid A \mid + \mid B \mid, T = \mid C \mid + \mid E \mid, 2S \geq 2E \geq S, 2T \geq 2D \geq T,$
$\quad som3(E, C, D), som3(A, B, E).$

Clearly, we have at least three tvpl-measures : $\mu^1 = (1, 0, 0), \mu^2 = (0, 1, 0)$ and $\mu^3 = (0, 0, 1)$. In general, we can limit ourselves to a "finite basis" of measures which covers all others measures (see [12]).

Automatic discovery of tvpl-measures is a consequence of a result of [17] and gives another example of the computation interest in moving from $\mathcal{N}$ to $\mathcal{Q}^+$ (cf. remark 4.1).

Theorem 5.1 *In* $\mathcal{Q}^+$*, there exists a complete polynomial procedure for deciding the existence of a t-valid p-linear measure for any logic procedure defined using a single predicate symbol.*

This procedure, based on the duality theorem of linear programming (hence its theoretical complexity in $\mathcal{Q}^+$), can be adapted to compute the coefficients of μ [12]. The resulting coefficients for μ are of course non-negative rational numbers, but it is routine, starting from a p-linear t-valid measure for a CLP($\mathcal{Q}^+$) logic procedure p to construct a p-linear t-valid measure for p considered as a CLP($\mathcal{N}$) logic procedure.

Example 5.4 For the following CLP($\mathcal{Q}^+$) program P, it is not so easy to detect a tvpl-measure at first sight:

$p(X,Y) :- \ 10 > X, 10 > Y.$
$p(2X+3, Y+5) : -p(Y+7, X+2).$
$p(X+2, Y+3) :- \ Y+1 \geq 5X, p(X+3, Y+2), p(Y+3, 2X+2).$

Using our implementation of the technique of Sohn and Van Gelder, we find that $\mu(x,y) = \frac{5}{2}x + 5y$ is a tvpl-measure. So $\mu'(x,y) = 5x + 10y$ for P considered as a CLP($\mathcal{N}$) program is a tvpl-measure.

For a more general logic procedure p where $r_1, \ldots, r_n$ ($r_i \neq p$) appears in the definition of p, we delete each occurrence of each r_i and we apply theorem 5.1. Once again remember that we are dealing with specialized versions.

We end this section with a definition and two properties which makes our goal (left-termination) appear closer.

Definition 5.4 *Let p be a recursive predicate, associated to a non-empty set Γ_p of t-valid measures. We say that $\leftarrow c, p(\tilde{x})$* is bounded wrt Γ_p *if $\exists \mu \in \Gamma_p$ s.t. $\mu(\tilde{x})$ is bounded in c.*

Once a goal is bounded, its immediate descendants are smaller:

Proposition 5.1 *Let $\leftarrow c, p(\tilde{z})$ be a goal bounded wrt one of its valid measures $\mu : I\!N^{arity(p)} \rightarrow (W, \succ)$. Let $p(\tilde{x}) \leftarrow c', \ldots, p(\tilde{y}), \ldots$ be one of the rules defining p and c'' the conjunction $(\tilde{z} = \tilde{x}) \wedge c \wedge c'$. Then $\leftarrow c'', p(\tilde{y})$ is bounded wrt μ and for each solution θ of c'', $\mu(\tilde{z}\theta) \succ \mu(\tilde{y}\theta)$.*

Using proposition 5.1 and the order defined by the dependency graph of the program (well-founded because we disallow mutual recursive programs), we can show:

Theorem 5.2 *Let $P^{\mathcal{N}}$ be a program s.t. each recursive predicate p is associated to a non-empty set Γ_p of t-valid measures. If all the left most atoms met during the construction of a computation tree are either bounded or non-recursive, then the computation tree is finite.*

Example 5.5 Let us apply theorem 5.2 and proposition 5.1 to P_{ack} (example 5.1). If n is a natural number, then the goals $:- \ n \geq X + Y, ack(X,Y,Z).$ and $:- \ n \geq Z, ack(X,Y,Z).$ terminate.

Example 5.6 The following program (on the left) is taken from [11]. We approximate it (on the right) using the "exotic" size function of the example 3.2. Note that there are no non-trivial IR for *rotate*.

$rotate(empty).$
$rotate(tree(L, X, empty)).$
$rotate(tree(L1, X1, tree(L2, X2, R2))) :-$
$\quad rotate(tree(tree(L1, X1, L2), X2, R2)).$

$rotate(0).$
$rotate(1).$
$rotate(2 + R2) :-$
$\quad rotate(1 + R2).$

The proof of the goal : $-rotate(tree(X1, X2, tree(X3, X4, empty)))$. terminates as $\mu(t) = (1)$ is a tvpl-measure.

Remark 5.1 Indeed, when p is the only predicate symbol appearing in all the rules defining p, then if the top most goal is bounded, the computation tree is finite, for *any* computation rule.

6 Termination Conditions

Some kind of data-flow analysis misses us to take full profit of theorem 5.2. We show how one may rely on CLP($\mathcal{B}$) to fill the gap.

Definition 6.1 *Let $p(\tilde{x})$ be a n-ary CLP($\mathcal{N}$) recursive predicate with $\Gamma_p = \{\mu^1, \ldots, \mu^{q+1}\}$ as its associated set of t-valid measures. We map Γ_p to a boolean term called its* boolean measure *defined by:*

$$\gamma_p(\tilde{x}) = \bigvee_{1 \leq j \leq q+1} \left[\bigwedge_{i \in I_{\mu^j}} x_i \right]$$

If $p(\tilde{x})$ is a non-recursive predicate, we set $\gamma_p(\tilde{x}) = 1$.

Example 6.1 For $ack(x, y, z)$, we found two t-valid measures: $\mu^1 = (0, -1, 1)$ with $I_{\mu^1} = \{3\}$ and $\mu^2(x, y, z) = \langle x, y \rangle$ with $I_{\mu^2} = \{1, 2\}$. Its boolean measure is: $\gamma_{ack}(x, y, z) = z \vee (x \wedge y)$.

Example 6.2 For $som3(x, y, z)$, we found three tvpl-valid measures: $\mu^1 = (1, 0, 0), \mu^2 = (0, 1, 0)$ and $\mu^3 = (0, 0, 1)$. Its boolean measure is: $\gamma_{som3}(x, y, z) = x \vee y \vee z$.

Example 6.3 For $rev(x, y, z)$, $\gamma_{rev}(x, y, z) = x \vee z$.

Our aim is to find, for each predicate p of the CLP(χ) program a sufficient condition expressing left-termination. More precisely:

Definition 6.2 *A boolean term $TC_p(\tilde{x})$ is a* termination condition *for $p(\tilde{x})$ if, for every constraint c, $\forall[\mathcal{A}^{\mathcal{B}}_{\chi}(c) \Rightarrow TC_p(\tilde{x})]$ implies $\leftarrow c, p(\tilde{x})$ left − terminates.*

If we combine proposition 3.1 and remark 5.1, we obtain:

Proposition 6.1 *If p is the only predicate symbol appearing in the rules defining p then γ_p is a termination condition for p.*

Example 6.4 Hence $TC_{ack}(x, y, z) = z \vee (x \wedge y)$, $TC_{som3}(x, y, z) = x \vee y \vee z$ and $TC_{rev}(x, y, z) = x \vee z$.

Now it is time to address the general case. Assume that m rules define the predicate p. Let the kth-rule be:

$$\begin{array}{ll} \text{in CLP}(\mathcal{N})\ r_k^{\mathcal{N}}: & p(\tilde{x}) \leftarrow c_{k,1}^{\mathcal{N}}, p_{k,1}(\widetilde{x_{k,1}}), \ldots, p_{k,j_k}(\widetilde{x_{k,j_k}}) \\ \text{in CLP}(\mathcal{B})\ r_k^{\mathcal{B}}: & p(\tilde{x}) \leftarrow c_{k,1}^{\mathcal{B}}, p_{k,1}(\widetilde{x_{k,1}}), \ldots, p_{k,j_k}(\widetilde{x_{k,j_k}}) \end{array}$$

If $c_{k,1}^{\mathcal{N}}$ is the "empty" constraint, we rewrite it as $true$ and we put $c_{k,1}^{\mathcal{B}} = true$. From the kth-rule $r_k^{\mathcal{B}}$, we define a sequence $S_k^{\mathcal{B}}$ of j_k boolean constraints as follows: if the body of r_k is empty, then $S_k^{\mathcal{B}} = \langle\ \rangle$ else $S_k^{\mathcal{B}} = \langle c_{k,1}^{\mathcal{B}}, \ldots, c_{k,j_k}^{\mathcal{B}} \rangle$ with

$$c_{k,i+1}^{\mathcal{B}} \equiv c_{k,i}^{\mathcal{B}} \wedge c_{p_{k,i},success}^{\mathcal{B}}(\widetilde{x_{k,i}})$$

for $1 \leq i \leq j_k - 1$ (cf remark 3.2 for the definition of $c_{p_{k,i},success}^{\mathcal{B}}$).

Intuitively, each $c_{k,i}^{\mathcal{B}}$ is the boolean approximation of the "calling context" of $p_{k,i}(\widetilde{x_{k,i}})$. Now if for each predicate q other, we have a termination condition TC_q and we want to check TC_p, it remains to show that the calling context (depending on the calling context of the top goal p, i.e. TC_p) is strong enough to ensure that the sub-proof starting with q will left-terminate, i.e. implies TC_q. Note that if the rule is recursive, then there is a q which is p. More formally, with the help of propositions 3.1, 3.2, 6.1 and theorems 3.2, 5.2, we can prove:

Theorem 6.1 *Assume that for each predicate $q \neq p$ appearing in the rules defining p, we have a termination condition TC_q. If $TC_p(\tilde{x})$ verifies :*

$$\begin{cases} \forall [TC_p(\tilde{x}) \Rightarrow \gamma_p(\tilde{x})] \\ \textit{and} \\ \forall\ 1 \leq k \leq m, \forall\ 1 \leq i \leq j_k, \forall \left[TC_p(\tilde{x}) \wedge c_{k,i}^{\mathcal{B}} \Rightarrow TC_{p_{k,i}}(\widetilde{x_{k,i}})\right] \end{cases}$$

then $TC_p(\tilde{x})$ is a termination condition for $p(\tilde{x})$.

Example 6.5 We apply theorem 6.1 to prove that $TC(x, y, z, t) = (x \wedge y)$ is a termination condition for $som41(x, y, z, t)$ from example 1.1. We obtain the system:

$$\begin{cases} \forall [x \wedge y \Rightarrow 1] \\ \forall [x \wedge y \wedge 1 \Rightarrow x \vee y \vee u] \\ \forall [x \wedge y \wedge 1 \wedge (x \vee y) \wedge (u \Leftrightarrow (x \wedge y)) \Rightarrow u \vee z \vee t] \end{cases}$$

First, we check TC implies γ_{som41}. Second, TC together with the empty constraint implies $TC_{som3}(x, y, u)$. Third, TC and the returned constraint corresponding to the success of the proof of $som3(x, y, u)$ implies $TC_{som3}(u, z, t)$. All the three implications are true, hence $TC(x, y, z, t) = (x \wedge y)$ is a termination condition for $som41(x, y, z, t)$.

Theorem 6.1 also defines a method for inferring a termination condition: compute the most general (wrt $\Rightarrow$) boolean term $T(\tilde{x})$ which verifies the system. A correct algorithm is given in [12]. With our implementation, we compute the following termination conditions:

- for the specialized *reverse* predicate of the example 4.1,
  $TC_{reverse}(x,y) = x \vee y$
- for the non-specialized version of the example 1.1,
  $TC_{som41}(x,y,z,t) = (x \vee y) \wedge (x \vee z \vee t) \wedge (y \vee z \vee t)$
  $TC_{som42}(x,y,z,t) = (x \vee y \vee t) \wedge (z \vee t)$
- for the specialized (see example 5.3) version of the example 1.1,
  $TC_{som'41}(x,y,z,t) = (y \vee z \vee t) \wedge (x \vee y \vee t) \wedge (x \vee z \vee t)$
  $TC_{som'42}(x,y,z,t) = TC_{som'41}(x,y,z,t)$

So without specialization, TC_{som41} and TC_{som42} are not comparable. For $(x,y,z,t) = (1,1,0,0)$, $TC_{som41} = 1$ and $TC_{som42} = 0$ and for $(x,y,z,t) = (0,0,0,1)$, $TC_{som41} = 0$ and $TC_{som42} = 1$. But once specialized, the two predicates have the same better operational behavior: $TC_{som'4_1} = TC_{som'4_2}$, $TC_{som'4_1} \Rightarrow TC_{som4_1}$ and $TC_{som'4_1} \Rightarrow TC_{som4_2}$.

This is a general rule which we verify on many examples: if we can generate IR's in the form of conjunction of inequalities and if we can compute t-valid measures, then in most cases the computation rule has a rather low influence on termination, although it may have an impact on efficiency.

We see two possible ways to lift a termination condition from $\mathcal{B}$ to χ. Either the CLP system warns the user that for the goal $\leftarrow c, p(\tilde{x})$ to be proven, if $\forall \left[\mathcal{A}^{\mathcal{B}}_{\chi}(c) \Rightarrow TC_p(\tilde{x})\right]$ is false, left-termination is not insured. Or the result of the termination analysis can be an explicit text (e.g. "for $som'41(a,b,c,d)$, left termination is guaranteed if d is a list of bounded length or two lists among a,b,c are of bounded length"). In the latter case, we assume that the main words ("list of bounded length") are given with the definition of $\mathcal{A}^{\mathcal{N}}_{\chi}$.

We end this section by pointing out that the system defined in theorem 6.1 can be seen as an *effective definition* of TC_p in the propositional μ-calculus for which there is an efficient interpreter: *Toupie*, described in [5].

7 Conclusion

Let us first summarize our approach for left-termination of a CLP(χ) program P^{χ}, given $\mathcal{A}^{\mathcal{N}}_{\chi}$:

1. Abstract P^{χ} to $P^{\mathcal{N}}$,
2. Find interargument relations for $P^{\mathcal{N}}$ and add them to $P^{\mathcal{N}}$ and P^{χ}, now specialized to $P'^{\mathcal{N}}$ and P'^{χ}
3. Compute the t-valid measures for $P'^{\mathcal{N}}$,
4. Abstract $P'^{\mathcal{N}}$ to $P'^{\mathcal{B}}$ and compute the least fixpoint of $P'^{\mathcal{B}}$,
5. Compute the termination conditions in $\mathcal{B}$. Lifting them to χ, they become sufficient left-termination conditions for P'^{χ}.

So the main question is: to what extent can we automate $\mathcal{A}_\chi^\mathcal{N}$? We think that there are two possible answers to the problem. On the one hand, the more ambitious answer relies on adding another layer of approximation to infer $\mathcal{A}_\chi^\mathcal{N}$. Some works have already begun concerning that problem (e.g. [16]). On the other hand, a more reasonable approach might be to insist on a type discipline for CLP and to predefine an approximation based on the "usual" size for each data-structure. For instance, one can compute the termination conditions of most predicates defined in the library of SICStus Prolog using the usual associated sizes.

Concerning approximations, the original idea of such mappings was already present for resolution theorem proving in [14]. The concept has been adapted and developed for CLP in [3, 9]. We find it useful both from the theoretical side (clarity of the algebraic reasoning) and from the practical side (a CLP system freely provides the needed solvers).

Of course we owe a lot to the works on compile-time termination analysis of logic programs. This research started with [18, 15, 19]. Again, we refer the reader to the survey of D. De Schreye and S. Decorte [6]. In our approach, there is an obvious relation with the notions of *level mappings, rigidity, recurrency, boundeness* and *norms* introduced in [1, 2]. The two important improvements are the following. On the one hand, we allow, via the notion of measure, more flexiblity to choose the well-founded set. On the other hand, we generalize to CLP, where a particular data-structure may be defined with some "non-syntactical" function (e.g. the concatenation function for lists).

From the best of our knowledge, there is only one paper [4] dealing with termination for CLP. The authors characterize termination wrt to a precondition by means of the notion of "termination triple". The approach may take into account non-ideal CLP system and negation. Then they introduce a methodology for finding such triples and a sufficient criterion for termination. Although our proposal seems less powerful, it is directly oriented towards automation: we believe that programmers are both lazy and in a hurry. Hence we feel that it is up to the CLP system to provide, as far as it can, termination proofs.

We think that the main contribution of our work is twofold. First, we propose a clean generalization of concepts about termination from LP to CLP using approximations. Second, the extra-layer using CLP($\mathcal{B}$) allows us to derive at compile-time whether arguments are bounded in CLP($\mathcal{N}$), and to compute a set of classes of left-terminating queries.

To go one step farther, one may study the following problem. Given a class of left-terminating goals for P, is it possible, while preserving left-termination, to reorder the literals in the rules of P in order to reduce the size of the computation tree ?

References

[1] M. Bezem. Characterizing termination of logic programs with level mappings.

J. Logic Programming, 15(1-2):79–98, 1992.

[2] A. Bossi, N. Cocco, and M. Fabris. Norms on terms and their use in proving universal termination of a logic program. *Theoretical Computer Science*, 124:297–328, 1994.

[3] P. Codognet and G. Filé. Computations, abstractions and contraints in logic programs. In *Proc. of ICCL'92*. IEEE, 1992.

[4] L. Colussi, E. Marchiori, and M. Marchiori. On termination of constraint logic programs. In *Proc. of PPCP'95*, LNCS 976. Springer-Verlag, 1995.

[5] M-M. Corsini, B. Le Charlier, K. Musumbu, and A. Rauzy. Efficient abstract interpretation of prolog programs by means of constraint solving over finite domains. In *Proc. of PLILP'93*, LNCS 714. Springer-Verlag, 1993.

[6] D. De Schreye and S. Decorte. Termination of logic programs: the never-ending story. *J. Logic Programming*, 19:199–260, 1993.

[7] L. Fribourg and H. Olsén. Datalog programs with arithmetical constraints: Hierarchic, periodic and spiralling least fixpoints. Technical report, L.I.E.N.S, France, 1995.

[8] M. Gabbrielli and G. Levi. Modelling answer constraints in constraint logic programs. In *Proc. of ICLP'91*, pages 238–252. MIT Press, 1991.

[9] R. Giacobazzi, S.K. Debray, and G. Levi. A generalized semantics for constraint logic programs. In *Proc. of FGCS'92*, pages 581–591, 1992.

[10] J. Jaffar and M.J. Maher. Constraint logic programming: a survey. *J. Logic Programming*, 19:503–581, 1994.

[11] S. Lüettringhaus-Kappel. Control generation for logic programs. In *Proc. of 10th ICLP*, pages 478–495. MIT Press, 1993.

[12] F. Mesnard. *Etude de la terminaison des programmes logiques avec contraintes, au moyen d'approximations*. PhD thesis, Université Paris VI, 1993.

[13] F. Mesnard. Towards automatic control for clp(χ) programs. In *Proc. of LOPSTR'95*, LNCS 1048, pages 106–119. Springer Verlag, 1995.

[14] D. Plaisted. Abstraction mappings in mechanical theorem proving. In *Proc. of CADE'80*, LNCS 87, pages 264–280. Springer-Verlag, 1980.

[15] L. Plümer. *Termination Proofs For Logic Programs*. Springer-Verlag, 1989.

[16] B. Saglam and J.P. Gallagher. Approximating constraint logic programs using polymorphic types and regular descriptions. In *Proc. of PLILP'95*, LNCS 982, pages 461–462. Springer-Verlag, 1995.

[17] K. Sohn and A. Van Gelder. Termination detection in logic programs using argument sizes. In *Proc. of PODS'91*, pages 216–226. ACM Press, 1991.

[18] 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.

[19] K. Verschaetse and D. De Schreye. Deriving termination proofs for logic programs, using abstract procedures. In *Proc. of ICLP'91*, pages 301–315. MIT Press, 1991.

[20] K. Verschaetse and D. De Schreye. Derivation of linear size relations by abstract interpretation. In *Proc. of PLILP'92*, pages 296–310. LNCS 631, Springer Verlag, 1992.