CLP(χ) FOR AUTOMATICALLY PROVING PROGRAM PROPERTIES [1]

F. MESNARD, S. HOARAU and A. MAILLARD

▷ Various proof methods have been proposed to solve the implication problem, i.e. proving that properties of the form : $\forall(P \rightarrow Q)$ - where P and Q denote conjunctions of atoms - are logical consequences of logic programs. Nonetheless, it is a commonplace to say that it is still quite a difficult problem. Besides, the advent of the constraint logic programming scheme constitutes not only a major step towards the achievement of efficient declarative logic programming systems but also a new field to explore. By recasting and simplifying the implication problem in the constraint logic programming framework, we define a generic proof method for the implication problem, which we prove sound from the algebraic point of view. We present four examples using CLP($\mathbb{N}$), CLP($\mathcal{RT}$), CLP(Σ^*) and RISC-CLP($\mathbb{R}$). The logical point of view of the constraint logic programming scheme enables the automation of the proof method. At last, we prove the unsolvability of the implication problem, we point out the origins of the incompleteness of the proposed proof method and we identify two classes of programs for which we give a decision procedure for the implication problem. ◁

1. INTRODUCTION

Various proof methods, e.g. [5], [22], [23], [8] and [11], have been proposed to solve the implication problem, i.e. proving that properties of the form: $\forall(P \rightarrow Q)$ - where P and Q denote conjunction of atoms - are logical consequences of logic programs [24]. Nonetheless, it is a commonplace to say that it is still quite a difficult problem.

[1] An earlier version of this paper was presented at the workshop "Frontiers of Combining Systems", Munich, March 1996.

Address correspondence to Iremia - Université de La Réunion, 15 av. René Cassin, B.P. 7151 97715 Saint-Denis Messag. Cedex 9 France (e-mail: fred@univ-reunion.fr)

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/94/$7.00

Besides, the advent of the constraint logic programming (CLP) scheme (cf. [18], [19]) whose main instances are described in [7], [12] and [20], constitutes not only a major step towards the achievement of efficient declarative logic programming systems, but also a new field to explore.
By simplifying and recasting the implication problem in the CLP framework, i.e. proving that properties of the form $\forall(d_1 \wedge p \rightarrow d_2)$, where p is an atom and d_1 and d_2 disjunctions of constraints, are logical consequences of definite constraint logic programs, we propose a *generic* proof procedure for the implication problem which may be easily implemented.
Let us give an intuitive idea of the implementation. The standard goal of a constraint solver is to check the solvability of a conjunction of constraints. A non-standard use is to test whether a constraint is a logical consequence of a set of constraints. It relies on the following trick: an atomic constraint c is a logical consequence of a conjunction of constraints C_s iff $C_s \cup \neg c$ is unsolvable. So, in order to prove the property Prop: $\forall(p \rightarrow c)$, we first prove that Prop is true for the non-recursive clauses defining p. Then, assuming it holds for p in the body of a recursive clause, we thus prove that Prop is true for the head of the clause.

The paper is organized as follows: section 2 recalls basic notations, definitions and results of the CLP scheme. Section 3 presents the proof method from the algebraic point of view. We propose four examples of its use in section 4. In section 5, we show that the logical point of view of the CLP framework enables the automation of the proof method. We prove the unsolvability of the implication problem and we discuss the completeness of the method in section 6. We conclude by comparing our work with other techniques and sketching possible extensions of the proposed proof method.

2. PRELIMINARIES

Let us *briefly* recall some basic concepts of the CLP framework (see [18] and [19] for more details). In order to be concise, we only consider the mono-sorted case.

Let V be a denumerable set of variables, F a finite set of function symbols, C a finite set of constraint symbols and P a finite set of predicate symbols. A finite arity is assigned to each function, constraint and predicate symbol. We assume that C contains the binary symbol $=$. The set of *terms*, *atomic constraints* and *atoms*, defined as usually done, are denoted by $T(F,V)$, $A(C,F,V)$ and $A(P,F,V)$. $var(o)$ denotes the set of variables of the syntactic object o, and we write $o(\tilde{x})$ as a shorthand for o where $var(o) \subseteq \{\tilde{x}\}$.

Definition 2.1. A constraint is a (possibly empty) conjunction of atomic constraints.

Definition 2.2. A generalized constraint is a (possibly empty) disjunction of constraints.

Note that at this point of the statement we do not impose any finiteness condition on a constraint or a generalized constraint.

When we switch from syntax to semantics, the key notion of the CLP(χ) scheme lies in the introduction of a *structure* which embodies the meaning of the specific

intended constraint domain χ. More precisely, a structure χ over $< F, C >$ consists in a non-empty domain D_χ, an assignment to each n-ary function symbol of a function $(D_\chi)^n \rightarrow D_\chi$ and an assignment to each n-ary constraint symbol of a subset of $(D_\chi)^n$ (apart from the binary constraint symbol $=$, which is interpreted as equality). A χ-*valuation* is a mapping $V \rightarrow D_\chi$, extended in the obvious way to formulae. An atomic constraint c is χ-*solvable* iff there is a χ-valuation θ such that χ models $c\theta$ (which means that $c\theta$ evaluates to true wrt χ); θ is a χ-*solution* to c and we write: $\models_\chi c\theta$. Otherwise c is χ- *unsolvable*. χ-solvability and χ-unsolvability extend naturally to constraints (the empty conjunction of atomic constraints is trivially χ-solvable) and disjunctions of constraints (the empty disjunction of constraints is trivially χ-unsolvable). Let $\neg c$ denote the complement[1]of the solution space of c. This notation extends naturally to constraints and generalized constraints.
The structure χ has to be *solution-compact* wrt $< F, C >$, i.e. it should satisfy:

(SC_1) every element of D_χ can be defined by a constraint (i.e. $\forall d \in D_\chi$, there exists a constraint c with $\{x\} \subseteq var(c)$ s.t. for every solution θ of c we have: $x\theta = d$)

and

(SC_2) the complement of each atomic constraint can be described by a generalized constraint (i.e. for every atomic constraint c, there exists a generalized constraint c' s.t. $\models_\chi \forall(\neg c \leftrightarrow c')$).

Let Pgm be a definite CLP(χ) program, P be the set of predicate symbols in Pgm and ${D_\chi}^*$ be the set of finite sequences of elements of D_χ. The χ-*base* is the cross product $P \times {D_\chi}^*$ respecting the arities of the predicate symbols. A χ-*interpretation* is any subset of the χ-base. A χ-*model* of Pgm is a χ-interpretation in which all the clauses of Pgm are true. We have a mapping $T_{Pgm,\chi}$ from and into the χ-base:

$T_{Pgm,\chi}(S) = \{$ $d \in \chi$-base: there is a clause in Pgm: $A \leftarrow c \diamond Body$ where A is an atom, $Body$ a conjunction of atoms, c a finite constraint and a χ-valuation θ such that $\models_\chi c\theta$, $\models_\chi$ $(A\theta = d)$ and $\{Body\theta\} \subseteq S\}$

whose powers are defined as usually done:

$$\begin{cases} T_{Pgm,\chi} \uparrow 0 = \emptyset \ ; \\ T_{Pgm,\chi} \uparrow n+1 = T_{Pgm,\chi}(T_{Pgm,\chi} \uparrow n) \ ; \\ T_{Pgm,\chi} \uparrow \omega = \bigcup_{n \in \mathbb{N}} (T_{Pgm,\chi} \uparrow n). \end{cases}$$

At last, the following fundamental property establishes the equivalence between the algebraic and the fixpoint semantics:

Theorem 2.1. (semantics of constraint logic programs [18]) The least χ-model of Pgm is the least fixpoint of $T_{Pgm,\chi}$ i.e. we have $M_{\chi,Pgm} = T_{Pgm,\chi} \uparrow \omega$

[1]If we consider the structure $< \mathbb{N}, \{0, 1, +\}, \{=\} >$ and the constraint $c(x, y) \equiv x = y + 1$ then $\neg c(x, y)$ denotes for instance the following generalized constraint: $x = y \vee x = y + 2 \vee x = y+3 \vee \ldots \vee x+1 = y \vee x+2 = y \vee \ldots$ But $\neg c(x, y)$ denotes the generalized constraint $x \geq y+2 \vee y \geq x$ if the structure is $< \mathbb{N}, \{0, 1, +\}, \{=, \geq\} >$.

3. THE PROOF METHOD (ALGEBRAIC POINT OF VIEW)

Let B_1 and B_2 be two generalized constraints. We write $B_1 \models_\chi B_2$ iff every χ-solution of B_1 is a χ-solution of B_2. We have the following property:

Observation 3.1. $B_1 \wedge \neg B_2$ *is* $\chi-$*unsolvable iff* $B_1 \models_\chi B_2$.

We now give the format of the properties we want to prove.

Definition 3.1. (system of implications) Let $\tilde{x}$ be a vector of distinct variables, $P = \{p_1, \cdots, p_n\}$ a set of predicate symbols, and $B_1, \cdots, B_n$ n finite generalized constraints. We call *a system of implications* the following conjunction of implications:

$$\bigwedge_{i=1}^{n} \Big[\forall \tilde{x}\big(p_i(\tilde{x}) \rightarrow B_i(\tilde{x})\big)\Big]$$

Before formulating the main result of this section, we give a syntax of constraint logic procedures. Let Pgm be a definite CLP(χ) program and P the set of all predicate symbols that appear in at least one clause of Pgm. If $p \in P$, let n_p be the total number of clauses from Pgm which define p. The ith clause defining p is denoted $Cl_{p,i}$ and can be written as:

$$(Cl_{p,i}) \qquad p(\tilde{h}_i) \leftarrow C_{p,i} \diamond p_{i,1}(\tilde{t}_{i,1}) \ldots p_{i,k_i}(\tilde{t}_{i,k_i})$$

where $p_{i,1}, \ldots, p_{i,k_i}$ are predicates from P, C's denote finite constraints, $\tilde{h}$'s and $\tilde{t}$'s are vectors of terms from $T(F,V)$, well formed wrt the corresponding predicate arity. Notice that we can have $k_i = 0$, which means $Cl_{p,i}$ is a unitary clause and, that in the body of a clause of a predicate p, we can have $p_{i,j} = p$, which means $Cl_{p,i}$ is a recursive clause.
Now, let (SI) be the following system of implications:

$$\bigwedge_{p \in P} \Big[\forall \tilde{x}\big(p(\tilde{x}) \rightarrow B_p(\tilde{x})\big)\Big]$$

Let τ_{SI} be the mapping which turns each clause $Cl_{p,i}$ into the generalized constraint[2]:

$$\tau_{SI}(Cl_{p,i}) \equiv \Big(\bigwedge_{j=1}^{k_i} B_{p_{i,j}}(\tilde{t}_{i,j})\Big) \wedge C_{p,i} \wedge (\tilde{h}_i = \tilde{x}) \wedge \neg B_p(\tilde{x})$$

We have the following result:

Theorem 3.1. (the proof method and its correctness) If

$$\Big[\bigvee_{\substack{p \in P \\ 1 \leq i \leq n_p}} \tau_{SI}(Cl_{p,i})\Big] \text{ is } \chi - \text{unsolvable} \qquad (H)$$

then the least χ*-model of* Pgm *is a* χ*-model of* (SI).

[2]One may write $\neg B_p(\tilde{h}_i)$ instead of $(\tilde{h}_i = \tilde{x}) \wedge \neg B_p(\tilde{x})$. But this simplification hides a crucial point of the proof, namely that $\tilde{x}$ is a vector of fresh distinct variables.

PROOF. Let I be the set $\{p(\tilde{s}) \mid p \in P, \models_\chi B_p(\tilde{s})\}$. I is a χ-interpretation which is a χ-model of (SI) (by construction). We first prove that I is a χ-model of Pgm. Let $Cl_{p,i}$ be a clause of Pgm, say $p(\tilde{h}_i) \leftarrow C_{p,i} \diamond p_{i,1}(\tilde{t}_{i,1}) \ldots p_{i,k_i}(\tilde{t}_{i,k_i})$ and let θ be a χ-valuation verifying the following properties:

$$\begin{cases} p_{i,1}(\tilde{t}_{i,1}\theta) \in I, \ldots p_{i,k_i}(\tilde{t}_{i,k_i}\theta) \in I & (3.1) \\ \models_\chi C_{p,i}\theta & (3.2) \end{cases}$$

We must prove that $p(\tilde{h}_i\theta) \in I$. By hypothesis (H) we know that:

$$\Big[\bigvee_{\substack{p \in P \\ 1 \le i \le n_p}} \tau_{SI}(Cl_{p,i}) \Big] \; is \; \chi - unsolvable$$

and so, we know that for the predicate p and the considered clause:

$$\Big[\big(\bigwedge_{j=1}^{k_i} B_{p_{i,j}}(\tilde{t}_{i,j}) \big) \wedge C_{p,i} \wedge (\tilde{h}_i = \tilde{x}) \wedge \neg B_p(\tilde{x}) \Big] \; is \; \chi - unsolvable$$

The previous formula is universally quantified and hence, if we apply θ:

$$\Big[\big(\bigwedge_{j=1}^{k_i} B_{p_{i,j}}(\tilde{t}_{i,j}\theta) \big) \wedge C_{p,i}\theta \wedge (\tilde{h}_i\theta = \tilde{x}\theta) \wedge \neg B_p(\tilde{x}\theta) \Big] \; is \; \chi - unsolvable$$

But by (3.1) we have: $\models_\chi \bigwedge_{j=1}^{k_i} B_{p_{i,j}}(\tilde{t}_{i,j}\theta)$. So by (3.2) and because $\tilde{h}_i\theta = \tilde{x}\theta$ is always solvable, we conclude that $\neg B_p(\tilde{h}_i\theta)$ is χ-unsolvable i.e. $\models_\chi B_p(\tilde{h}_i\theta)$ (by observation 3.1). Consequently, $p(\tilde{h}_i\theta) \in I$ which implies that I is a χ-model of $Cl_{p,i}$, hence a χ-model of Pgm.
To conclude the proof, it remains to show that the least χ-model $M_{\chi,Pgm}$ of Pgm is a χ-model of (SI). Let $p \in P$ and $Imp_p : \forall \tilde{x} \; \big(p(\tilde{x}) \rightarrow B_p(\tilde{x})\big)$ be the implication corresponding to p in (SI). If there is no $p(\tilde{s}) \in M_{\chi,Pgm}$ then Imp_p is trivially true. Else, since $M_{\chi,Pgm} \subseteq I$ (by definition of the least χ-model) and I is a χ-model of (SI), we have $\models_\chi B_p(\tilde{s})$. □

Remark 3.1. With our definition of the mapping τ, we must check that for each predicate p and for each clause $Cl_{p,i}$ defining p, $[\exists\tau(Cl_{p,i})]$ is χ-unsolvable. Let us restate the approach in an implicative form that one may find easier to understand. First, we define a new mapping τ':

$$\tau'_{SI}(Cl_{p,i}) \;\equiv\; \left[\left(\bigwedge_{j=1}^{k_i} B_{p_{i,j}}(\tilde{t}_{i,j}) \right) \wedge C_{p,i} \wedge (\tilde{h}_i = \tilde{x}) \right] \Rightarrow B_p(\tilde{x})$$

Then we check that $\models_\chi \forall[\tau'(Cl_{p,i})]$ for each clause $Cl_{p,i}$. Note that from a practical point of view, the usual way to show $\models_\chi \forall\phi$ using a constraint solver of a CLP system is to prove that $\neg\phi$ is χ-unsolvable.

Corollary 3.1. The proof method can be applied to the following system of extended implications:

$$\bigwedge_{p\in P}\Big[\forall\tilde{x}\ \big(A_p(\tilde{x})\wedge p(\tilde{x})\big)\rightarrow B_p(\tilde{x})\Big]$$

where A_p and B_p are generalized constraints.

PROOF. This result is due to two reasons. We have the logical equivalence:

$$\forall\tilde{x}\ \big(A_p(\tilde{x})\wedge p(\tilde{x})\big)\rightarrow B_p(\tilde{x})\ \equiv\ \forall\tilde{x}\ p(\tilde{x})\rightarrow\big(\neg A_p(\tilde{x})\vee B_p(\tilde{x})\big)$$

and we notice that the negation of a generalized constraint can be rewritten (SC_2) as a generalized constraint. □

4. EXAMPLES

Let us give four applications of theorem 3.1. We would like to point out that the simplification of constraints relies on algebraic reasoning in the involved structures.

Example 4.1. This example emphasizes the use of a generalized constraint in an implication. Let Pgm be the following CLP($\mathbf{N}$) definite program (McCarthy's 91 function), where $\mathbf{N}$ denotes the set of natural numbers, with $F=\{0,1,+,-\}$ and $C=\{=,\neq,>,\leq\}$ [9]:

$$\begin{array}{lll}(Cl_{p,1}) & p(u,u-10)\leftarrow u>100\ \diamond & \\ (Cl_{p,2}) & p(u,w)\leftarrow u\leq 100\ \diamond\ p(u+11,v),\ p(v,w) & \end{array}$$

and the implication:

$$(SI)\qquad \forall(x,y)\quad p(x,y)\rightarrow\big[(x>100\wedge y=x-10)\ \vee\ (x\leq 100\wedge y=91)\big]$$

We apply the mapping τ_{SI}:

$\tau_{SI}(Cl_{p,1})\equiv\big[u>100\big]\wedge\big[(u,u-10)=(x,y)\big]\wedge\neg\big[(x>100\wedge y=x-10)\vee(x\leq 100\wedge y=91)\big]$

$\tau_{SI}(Cl_{p,2})\equiv\big[(u+11>100\wedge v=u+11-10)\vee(u+11\leq 100\wedge v=91)\big]\wedge\big[(v>100\wedge w=v-10)\vee(v\leq 100\wedge w=91)\big]\wedge\big[u\leq 100\big]\wedge\big[(u,w)=(x,y)\big]\wedge\neg\big[(x>100\wedge y=x-10)\vee(x\leq 100\wedge y=91)\big]$

We must prove $\big(\tau_{SI}(Cl_{p,1})\vee\tau_{SI}(Cl_{p,2})\big)$ is $\mathbf{N}$-unsolvable.

- $\tau_{SI}(Cl_{p,1})\rightarrow\big[u>100\big]\wedge\neg\big[(u>100\wedge u-10=u-10)\big]\wedge\neg\big[(u\leq 100\wedge u-10=91)\big]$

$\rightarrow\big[u>100\big]\wedge\big[(u\leq 100\vee u-10\neq u-10)\big]\wedge\big[(u>100\vee u-10\neq 91)\big]$

$\rightarrow\big[(u>100)\wedge(u\leq 100)\big]\wedge\big[(u>100\vee u-10\neq 91)\big]$

$\tau_{SI}(Cl_{p,1})$ is $\mathbf{N}$-unsolvable.

- $\tau_{SI}(Cl_{p,2})\rightarrow\big[(u>89\wedge v=u+1)\vee(u\leq 89\wedge v=91)\big]\wedge\big[(v>100\wedge w=v-10)\vee(v\leq 100\wedge w=91)\big]\wedge\big[u\leq 100\big]\wedge\big[u\leq 100\vee w\neq u-10\big]\wedge\big[u>100\vee w\neq 91\big]$

By simplifications and the application of $(\vee/\wedge)$-distributivity we obtain:

$$\tau_{SI}(Cl_{p,2}) \rightarrow \big[(w=91)\wedge(w\neq 91)\wedge[\ldots]\big]\vee\big[(v=91)\wedge(v>100)\wedge[\ldots]\big]\vee \big[(w=91)\wedge(w\neq 91)\wedge[\ldots]\big]\vee\big[(w=91)\wedge(w\neq 91)\wedge[\ldots]\big]$$

Once again, $\tau_{SI}(Cl_{p,2})$ is $\mathbb{N}$-unsolvable. So we conclude to the unsolvability of $\big(\tau_{SI}(Cl_{p,1})\vee\tau_{SI}(Cl_{p,2})\big)$, i.e. the least $\mathbb{N}$-model of Pgm is a $\mathbb{N}$-model of (SI). Notice that the converse:

$$\forall(x,y)\qquad \big[(x>100\wedge y=x-10)\vee(x\leq 100\wedge y=91)\big]\rightarrow p(x,y)$$

is true and can be proved directly by any system including CLP($\mathbb{Q}$). $\triangleleft$

Example 4.2. This example shows that it might be necessary to prove a 'strong enough' system of implications (see section 6). Let Pgm be the following CLP($\mathcal{RT}$) definite program, where $\mathcal{RT}$ denotes the set of rational trees, with the two constraints $=$ and $\neq$ [6].

$$\begin{array}{lll}(Cl_{q,1}) & q(u) & \leftarrow u=g(v),\ v=h(w),\ w=f(u)\ \diamond\\ (Cl_{q,2}) & q(u) & \leftarrow u=g(v)\ \diamond\ p(f(u),v)\\ (Cl_{p,1}) & p(u,v) & \leftarrow v=h(u)\ \diamond\ q(g(v))\end{array}$$

And let (SI) be the system of implications we want to prove (we write $fgh(x)$ as a shorthand for $f(g(h(x)))$):

$$\begin{cases}\forall x\quad q(x)\ \rightarrow x=ghf(x) & (4.1)\\ \forall(x,y)\quad p(x,y)\ \rightarrow x=fgh(x)\ \wedge\ y=hfg(y) & (4.2)\end{cases}$$

We apply the mapping τ_{SI}:

$\tau_{SI}(Cl_{q,1}) \equiv \big[\big(u=g(v)\big)\wedge\big(v=h(w)\big)\wedge\big(w=f(u)\big)\wedge\big(u=x\big)\wedge\big(x\neq ghf(x)\big)\big]$

$\tau_{SI}(Cl_{q,2}) \equiv \big[\big(f(u)=fghf(u)\big)\wedge\big(v=hfg(v)\big)\wedge\big(u=x\big)\wedge\big(x\neq ghf(x)\big)\big]$

$\tau_{SI}(Cl_{p,1}) \equiv \big[\big(g(v)=ghfg(v)\big)\wedge\big(v=h(u)\big)\wedge\big((u,v)=(x,y)\big)\wedge\neg\big(x=fgh(x)\wedge y=hfg(y)\big)\big]$

We must prove $\big(\tau_{SI}(Cl_{q,1})\vee\tau_{SI}(Cl_{q,2})\vee\tau_{SI}(Cl_{p,1})\big)$ is $(\mathcal{RT})$-unsolvable.

- $\tau_{SI}(Cl_{q,1})\rightarrow\big(u=ghf(u)\big)\wedge\big(u=x\big)\wedge\big(x\neq ghf(x)\big)$
  $\rightarrow\big(x=ghf(x)\big)\wedge\big(x\neq ghf(x)\big)$ and is $(\mathcal{RT})$-unsolvable
- $\tau_{SI}(Cl_{q,2})\rightarrow\big(u=ghf(u)\big)\wedge\big(v=hfg(v)\big)\wedge\big(u=g(v)\big)\wedge\big(u\neq ghf(u)\big)$, is $(\mathcal{RT})$-unsolvable

Note that the implication (4.2) is used to achieve the demonstration of the $(\mathcal{RT})$-unsolvability of $\tau_{SI}(Cl_{q,2})$.
Similarly, we can prove that $\tau_{SI}(Cl_{p,1})$ is $(\mathcal{RT})$-unsolvable. Hence the least $(\mathcal{RT})$-model of Pgm is a $(\mathcal{RT})$-model of (SI). $\triangleleft$

Example 4.3. We study a formal system using CLP(Σ^*). Consider the set S of

words (or strings) over $\Sigma = \{\mathbf{m,i,u}\}$ defined in [16] as follows: S is the least set containing **mi** and verifying the four rules (α and β are any words over Σ):

if $\alpha\mathbf{i} \in S$ then $\alpha\mathbf{iu} \in S$;
if $\mathbf{m}\alpha \in S$ then $\mathbf{m}\alpha\alpha \in S$;
if $\alpha\mathbf{iii}\beta \in S$ then $\alpha\mathbf{u}\beta \in S$;
if $\alpha\mathbf{uu}\beta \in S$ then $\alpha\beta \in S$.

We now briefly propose a possible definition of CLP(Σ^*) (see also [30]). Let $F_0 = \{\mathbf{m,i,u,.}\}$ where **m**, **i** and **u** are constant symbols and **.** is a binary function symbol, $C = \{=, \in\}$, where $=$ and $\in$ are two binary constraint symbols. Let t and s be two elements of $T(F_0, V)$, $F = F_0 \cup \{\emptyset, \Lambda, *, +\}$, and R be an element of $T(F, V)$, which is a regular expression over $\{\mathbf{m,i,u,.}\}$. An atomic constraint is either of the form $t = s$ or $t \in R$. The associated domain D_{Σ^*} is $\{\mathbf{m,i,u}\}^*$, the constant symbols **m**, **i** and **u** are interpreted as the strings **m**, **i** and **u**, and **.** as concatenation of strings. The solution space of the constraint $x \in R$ is the set of words over $\{\mathbf{m,i,u}\}$ which the regular expression R denotes. This structure is clearly solution-compact wrt $< F, C >$. Moreover, it is well known that there is a regular expression R' such that $\neg(x \in R) \equiv x \in R'$; we write $x \notin R$ as a shorthand for $\neg(x \in R)$.

We can now easily construct a CLP(Σ^*) program Pgm (with a slight abuse of notation) that exactly recognizes the words of S:

$dh(\mathbf{mi}) \leftarrow \diamond$
$dh(x.\mathbf{iu}) \leftarrow \diamond\ dh(x.\mathbf{i})$
$dh(\mathbf{m}.x.x) \leftarrow \diamond\ dh(\mathbf{m}.x)$
$dh(x.\mathbf{u}.y) \leftarrow \diamond\ dh(x.\mathbf{iii}.y)$
$dh(x.y) \leftarrow \diamond\ dh(x.\mathbf{uu}.y)$

Let us prove that each element of S begins with **m** and, after the **m**, contains only **i**'s and **u**'s. Formally, we have:

$$\forall x \quad dh(x) \rightarrow x \in \mathbf{m.(i+u)}^* \tag{4.3}$$

We try to apply theorem 3.1:
- $x = \mathbf{mi} \wedge x \notin \mathbf{m.(i+u)}^*$ is clearly Σ^*-unsolvable;
- $x.\mathbf{i} \in \mathbf{m.(i+u)}^*$ implies $x.\mathbf{iu} \in \mathbf{m.(i+u)}^*$ hence $x.\mathbf{i} \in \mathbf{m.(i+u)}^* \wedge x.\mathbf{iu} \notin \mathbf{m.(i+u)}^*$ is Σ^*-unsolvable;
- likewise, $\mathbf{m}.x \in \mathbf{m.(i+u)}^* \wedge \mathbf{m}.x.x \notin \mathbf{m.(i+u)}^*$ is Σ^*-unsolvable;
- if $x.\mathbf{iii}.y \in \mathbf{m.(i+u)}^*$ then $x = \mathbf{m}.x'$ and $x'.\mathbf{iii}.y \in \mathbf{(i+u)}^*$ so $x'.\mathbf{u}.y \in \mathbf{(i+u)}^*$ therefore $\mathbf{m}.x'.\mathbf{u}.y \in \mathbf{m.(i+u)}^*$. Consequently, $x.\mathbf{iii}.y \in \mathbf{m.(i+u)}^* \wedge x.\mathbf{u}.y \notin \mathbf{m.(i+u)}^*$ is Σ^*-unsolvable;
- All the same, $x.\mathbf{uu}.y \in \mathbf{m.(i+u)}^* \wedge x.y \notin \mathbf{m.(i+u)}^*$ is Σ^*-unsolvable.

From theorem 3.1, we conclude that (4.3) is true in the least Σ^*-model of Pgm. In the same way, we could prove:

$$\forall x \quad dh(x) \rightarrow x \in \mathbf{m.(u^*.i.u^* + u^*.i.u^*.i.u^*).(i.u^*.i.u^*.i.u^*)}^* \tag{4.4}$$

which shows that the number n of **i**'s in a word x of S verifies: $n \equiv 1$ (mod 3) or $n \equiv 2$ (mod 3). Consequently, (4.4) gives a negative answer to the main question asked by D. Hofstadter [16] concerning S: 'does **mu** belong to S ?' $\triangleleft$

Example 4.4. Let us consider the homographic series defined as follows:

$$\begin{cases} u_0 \neq -\frac{d}{c} \\ u_{n+1} = \frac{au_n + b}{cu_n + d},\ n \geq 0 \end{cases}$$

where a, b, c and d are real numbers with $c \neq 0$. We want to prove that the series is constant equal to $\frac{a}{c}$ from $n = 1$ if $ad - bc = 0$.

First, we define the following definite program on RISC-CLP($\mathbb{R}$), where the domain is the set of real numbers, $F = \{0, 1, +, -, *\}$ and $C = \{=, \neq, \geq, >\}$ (see [17] for more details):

$$\begin{array}{ll} (Cl_1)\ s(0, u_0, a, b, c, d) & \leftarrow -cu_0 \neq d,\ c \neq 0 \diamond \\ (Cl_2)\ s(n+1, x, a, b, c, d) & \leftarrow -cy \neq d,\ x(cy + d) = ay + b,\ c \neq 0 \\ & \quad \diamond\ s(n, y, a, b, c, d) \end{array}$$

So, we want to prove the following implication:

$$\forall (n, x, a, b, c, d) \in \mathbb{R}^6, [(n > 0\ \wedge\ ad - bc = 0) \wedge s(n, x, a, b, c, d)] \rightarrow cx = a$$

We recognize the format of extended implications described in corollary 3.1. So, we must prove (SI):

$$\forall (n, x, a, b, c, d) \in \mathbb{R}^6, s(n, x, a, b, c, d) \rightarrow (0 \geq n\ \vee\ ad - bc \neq 0\ \vee\ cx = a)$$

$\tau_{SI}(Cl_1) \equiv n = 0\ \wedge\ -cu_0 \neq d\ \wedge\ c \neq 0\ \wedge\ n > 0\ \wedge\ ad - bc = 0\ \wedge\ cu_0 \neq a$, is clearly $\mathbb{R}$-unsolvable.

$\tau_{SI}(Cl_2) \equiv (0 \geq n\ \vee\ ad - bc \neq 0\ \vee\ cy = a)\ \wedge\ -cy \neq d\ \wedge\ c \neq 0\ \wedge x(cy + d) = ay + b\ \wedge\ n > 0\ \wedge\ ad - bc = 0\ \wedge\ cx \neq a$

If $0 \geq n$ then $\tau_{SI}(Cl_2)$ is trivially $\mathbb{R}$-unsolvable. And the same if $ad - bc \neq 0$. If $cy = a$ then:

$$\begin{aligned} x(cy + d) = ay + b &\equiv cx(cy + d) = acy + bc \text{ (multiply by } c \neq 0) \\ &\equiv cx(a + d) = a^2 + bc \text{ (because } cy = a) \\ &\equiv cx(a + d) = a^2 + ad \text{ (because } ad - bc = 0) \\ &\equiv cx(a + d) = a(a + d) \end{aligned} \tag{4.5}$$

If $a + d = 0$ then $(cy = a) \equiv (-cy = d)$ and then $\tau_{SI}(Cl_2)$ is $\mathbb{R}$-unsolvable. Else, we simplify (4.5) by $a + d$ which gives $cx = a$ and $\tau_{SI}(Cl_2)$ is $\mathbb{R}$-unsolvable. So the property holds. $\triangleleft$

5. AUTOMATING THE PROOF METHOD (LOGICAL POINT OF VIEW)

First, in order to automate the proof method described in section 3, we take advantage of the complete correspondence between the algebraic and the logical semantics for definite constraint logic programs, by considering a *satisfaction-complete* theory Th which *corresponds* to χ:

- χ is a model of Th, i.e. $\models_\chi Th$
- for every constraint c, $\models_\chi \exists c$ iff $Th \models \exists c$

- for every constraint c, either $Th \models \exists c$ or $Th \models \neg\exists c$

Next, we impose that the negation of the generalized constraints appearing in a system of (extended) implications are *finite* generalized constraints themselves. A sufficient condition for this requirement is that for each atomic constraint c of each generalized constraint, there are n atomic constraints $c_1, \ldots, c_n$ such that:

$$\neg c \equiv c_1 \vee \ldots \vee c_n$$

It implies the signature of the constraint system contains only a finite number of constant symbols. But many theoretical results about logic programming assume an infinite number of constant symbols (see [2] for example). Hence if we want to automatically prove properties of pure logic programs, we need a constraint solver for $\mathcal{FT}$ with equality and disequality constraints.

So, by replacing χ-unsolvability by Th-unsatisfiability, theorem 3.1 directly provides us with a correct procedure to prove that a system SI of (extended) implications is true in the least model of a program Pgm and Th. The finiteness condition concerning SI, as stated above, ensures its termination. However, its computational complexity heavily depends on the complexity of the constraint solver, for which Th gives a theoretical lower bound [15].

Example 5.1. We explain why one might use CLP($\mathbb{Q}$) for proving properties of CLP($\mathbb{N}$) programs. Let $F = \{0, 1, +\}$, where 0 and 1 are constant symbols, and $+$ is a binary function symbol. Let $C = \{=, \geq\}$, where $=$ and $\geq$ are two binary constraint symbols. If t is a term from $T(F, V)$ and n a natural number, we denote by nt the term defined by $0t = 0$ and $(n+1)t = nt + t$. Furthermore, we write n to abbreviate $n1$. One can easily show that $\mathbb{N}$ and $\mathbb{Q}$ - wrt the obvious interpretation of F and C - are two solution-compact structures for $< F, C >$. We define the '$\mathbb{N}-$*complement*' of $t = s$ and $t \geq s$ as:

$$\neg(t = s) \equiv (t \geq s + 1) \vee (s \geq t + 1) \quad \textit{and} \quad \neg(t \geq s) \equiv (s \geq t + 1)$$

We have the following proposition:

Observation 5.1. $(c \wedge \neg c')$ *is* $\mathbb{Q}-$*unsolvable implies* $c \models_{\mathbb{N}} c'$

that enables us to use a symbolic simplex-like algorithm as a constraint solver to implement theorem 3.1, and provides us with a correct procedure for proving properties of CLP($\mathbb{N}$) programs. The complexity of a solver for CLP($\mathbb{N}$) [9] and observation 5.1 justify the switch from $\mathbb{N}$ to $\mathbb{Q}$. $\triangleleft$

An implementation of theorem 3.1 based on observation 5.1 has been written in Prolog III. The idea is that for all predicate p of a program and for each clause defining p, we use the Prolog III interpreter to check the unsolvability of the formulae: $\left(\bigwedge_{j=1}^{k_i} B_{p_{i,j}}(\tilde{t}_{i,j})\right) \wedge C_{p,i} \wedge (\tilde{h}_i = \tilde{x}) \wedge \neg B_p(\tilde{x})$. If we succeed for all clauses of p then we conclude that the hypothesis of theorem 3.1 is true and so we have the

implication for p.

Example 5.2. Consider the following program which defines the Ackermann's function:

$$\begin{array}{ll} ack(0,y,y+1) & \leftarrow y \geq 0 \diamond \\ ack(x+1,0,z) & \leftarrow x \geq 0,\ z \geq 0 \diamond\ ack(x,1,z) \\ ack(x+1,y+1,z) & \leftarrow x \geq 0,\ y \geq 0,\ z \geq 0,\ t \geq 0 \diamond \\ & \quad ack(x+1,y,t), ack(x,t,z) \end{array}$$

Using our implementation we can show that $\forall(x,y,z)\ \ [ack(x,y,z) \rightarrow\ z \geq y+1]$, and thus specialize the third clause:

$$\begin{array}{ll} ack(x+1,y+1,z) & \leftarrow x \geq 0,\ y \geq 0,\ t \geq y+1,\ z \geq t+1 \diamond \\ & \quad ack(x+1,y,t), ack(x,t,z) \end{array}$$

in order to be sure to find all the solutions to the goal:

$$\begin{array}{l} >\ \ z \leq 61 \diamond\ ack(3,y,z) \\ \{y=0\ ,\ z=5\} \\ \{y=1\ ,\ z=13\} \\ \{y=2\ ,\ z=29\} \\ \{y=3\ ,\ z=61\} \\ >\ \ \triangleleft \end{array}$$

6. INHERENT INCOMPLETNESS OF THE PROOF METHOD

First of all, let us point out how difficult is the problem we address:

Theorem 6.1. The implication problem, i.e. proving that properties of the form $\forall(d_1 \wedge p \rightarrow d_2)$ *- where* p *is an atom and* d_1 *and* d_2 *disjunctions of constraints - are* χ*-logical consequences of definite constraint logic programs, is unsolvable.*

PROOF. The proof we present relies on CLP($\mathbf{N}$) and can be generalized to most domains under reasonable assumptions.

A 2-register machine (see [26], [21] and [29]) has a pair of registers R_1 and R_2 which may hold an arbitrary natural number. A program for the machine is defined by specifying a finite number of states $S_0, (S_1, \ldots, S_n)$, together with, for each i, $1 \leq i \leq n$, an instruction to be carried out whenever the machine is in the state S_i. S_1 is the initial state and S_0 is the terminal state. Suppose we are in state S_i $(1 \leq i \leq n)$, there are the two kinds of instruction:

(1) add 1 to register R_j ($j = 1$ or 2) and move to state S_k, $(0 \leq k \leq n)$

(2) test if R_j holds 0: if it does, move to state S_l, otherwise, subtract 1 from it and move to state S_k, $(0 \leq k, l \leq n)$.

We can describe such a machine by a program in CLP($\mathbf{N}$). These following clauses represent the instructions (1) and (2):

$$\begin{array}{l} p(R_1,R_2,i) \leftarrow p(R_1+1,R_2,k) \\ p(R_1,R_2,i) \leftarrow R_2 = 0 \diamond\ p(R_1,R_2,l) \\ p(R_1,R_2,i) \leftarrow R_2 \geq 1 \diamond\ p(R_1,R_2-1,k) \end{array}$$

Consider a 2-register machine and Pgm its CLP($\mathbb{N}$) associated program. The proposition "$Pgm \models_{\mathbb{N}} p(R_1, R_2, q)$", where R_1, R_2 and q are integers, means that starting from the state S_q with registers R_1 and R_2, the machine halts in state S_0.

Example 6.1. Here is a program which computes $n_1 - n_2$ if $n_1 \geq n_2$ and fails to terminate if $n_1 < n_2$.

$S_0 \leftrightarrow$ Stop
$S_1 \leftrightarrow$ If $R_2 = 0$ then move to S_0
else subtract 1 from R_2 and move to S_2
$S_2 \leftrightarrow$ If $R_1 = 0$ then move to S_3
else subtract 1 from R_1 and move to S_1
$S_3 \leftrightarrow$ add 1 to $R_1$and move to S_3.

And its associated CLP($\mathbb{N}$) program:

$p(R_1, R_2, 0) \leftarrow \diamond$
$p(R_1, 0, 1) \leftarrow \diamond\ p(R_1, 0, 0)$
$p(R_1, R_2, 1) \leftarrow R_2 \geq 1\ \diamond\ p(R_1, R_2 - 1, 2)$
$p(0, R_2, 2) \leftarrow \diamond\ p(0, R_2, 3)$
$p(R_1, R_2, 2) \leftarrow R_1 \geq 1\ \diamond\ p(R_1 - 1, R_2, 1)$
$p(R_1, R_2, 3) \leftarrow \diamond\ p(R_1 + 1, R_2, 3)$

◁

2-register machines are useful for coding any unary recursive function. Let f be an unary recursive function, $Dom\, f$ be the definition domain of f and Pgm the program in CLP($\mathbb{N}$) associated with the 2-register machine coding f. We have the equivalence:

$$n \in Dom\, f \iff Pgm \models_{\mathbb{N}} p(n, R_2, 1) \tag{6.1}$$

Now, consider the following proposition:

$$(I) \qquad \forall R_1, R_2, q \in \mathbb{N} \quad p(R_1, R_2, q) \implies q \neq 1$$

If $Pgm \models_{\mathbb{N}} I$ then $p(R_1, R_2, 1)$ is not in the $\mathbb{N}$-model of Pgm. By (6.1), we conclude that $Dom\, f = \emptyset$. Conversely, if $Pgm \not\models_{\mathbb{N}} I$ then $\exists R_1, R_2, q \in \mathbb{N}$ such as $p(R_1, R_2, 1)$. Thus by (6.1) we have $R_1 \in Dom\, f$. Finally $(Pgm \models_{\mathbb{N}} I) \iff (Dom\, f = \emptyset)$.
Since for recursive functions, the problem "$Dom\, f = \emptyset$" is unsolvable and as I is a particular case of the implications we consider, we conclude to the unsolvability of the implication problem. □

Now, let us try to identify three weakness of the proof method we propose. The first one is a drawback of the implementation (but not of theorem 3.1) we present in example 5.1 lies in the fact that we make use of CLP($\mathbb{Q}$) to prove CLP($\mathbb{N}$) properties: clearly, there is a lack of precision. Consider for instance the following program:

$p(0) \leftarrow \diamond$
$p(x) \leftarrow 2x = 1\ \diamond$

The implication $\forall x,\ p(x) \rightarrow x = 0$ is true in CLP($\mathbb{N}$) but false in CLP($\mathbb{Q}$).

The second disadvantage of the proof method lies in the fact that we must have a 'strong enough' system of implications. For instance, in example 4.2, the proof method cannot work if we consider the following system of implications (let us call it SI'):

$$\begin{cases} \forall x & q(x) & \rightarrow x = x \\ \forall (x,y) & p(x,y) & \rightarrow x = fgh(x) \ \wedge \ y = hfg(y) \end{cases}$$

Yet this system is clearly true. A solution can be to first prove the system of implication SI defined in example 4.2. Then we prove $\models_\chi \forall(SI \rightarrow SI')$. But it seems difficult to compute the right 'strong enough' system of implications.

The third and main drawback of our proof method lies deeper. If the clauses defining a logic procedure p are unitary clauses, the method is obviously complete. However, as soon as the definition of p contains one recursive clause, problems may arise. Assume our proof method fails to prove the property $\forall \tilde{x}(p(\tilde{x}) \rightarrow c(\tilde{x}))$. Hence there exists one clause, say $Cl_{p,i}$, defining p such that the χ-solvability of a $\tau_{SI}(Cl_{p,i})$ produces a non-empty set S of solutions. If the clause $Cl_{p,i}$ is unitary, the property does not hold. But if the clause is recursive, then roughly speaking, the question is: $(Q) \quad M_{\chi,p} \cap S = \emptyset$? In other words, is there any solution $s \in S$ which also belongs to the semantics of p ? If the intersection is empty, then the property is true wrt the ith clause of p. If the intersection is not empty, we have at the same time a set of counter-examples and a guilty clause that show that the expected property is false. *However, we do not dispose of any finite means to answer the question Q.* In general, we can not finitely compute the set $M_{\chi,p} \cap S$. A possible enhancement could be to prove that p terminates on S then to execute the query $\leftarrow \tilde{x} \in S \diamond p(\tilde{x})$ on a CLP system. Anyway, it is well known that termination is undecidable. So we believe that here stands the heart of the incompleteness of the proof method shown by the theorem 6.1.

Example 6.2. Consider the CLP($\mathbb{N}$) program:

$$\begin{array}{ll} p(1,1) & \leftarrow \diamond \\ p(x+1, y+2x+1) & \leftarrow \diamond \ p(x,y) \end{array}$$

Assume we want to prove that $\forall x,y \ (p(x,y) \rightarrow y + 2 \leq 3x)$. The property is true for the unitary clause but for the recursive clause we obtain: $S(x,y) \equiv x + 3 \leq y + 2 \leq 3x$. So the question Q now becomes: can we find two natural numbers x and y s.t. $x + 3 \leq y + 2 \leq 3x$ and $p(x,y)$ hold ? The answer is yes: $x = 2$ and $y = 4$ (in fact, it is the unique solution and a graphic representation of p and S may help). Hence the property is false. The recursive clause generates the atom $p(3,9)$ true wrt the program, which invalidates the property. $\triangleleft$

This last remark leads us to recall a well-known technique for proving implications such as I: $\forall \tilde{x}[p(\tilde{x}) \rightarrow c(\tilde{x})]$, which resides in giving the goal $p(\tilde{x}) \wedge \neg c(\tilde{x})$ to the corresponding CLP system. If the answer is negative, then I is true; if the system computes some values for $\tilde{x}$, then I is false. However, it may also loop: this is the case for most examples in this article. This fact constitutes the main difference with our proof procedure which always terminates.

Finally, for some classes of programs, there is at least a decision procedure[3] for proving systems of implications. Let us present two classes of programs and a possible proof method.

Definition 6.1. [24] A level mapping of a program is a mapping from its set of predicate symbols to the non-negative integers. We refer to the value of a predicate symbol p under this mapping as the level of that predicate symbol, denoted by $level(p)$.

Definition 6.2. [24] A program is hierarchical if it has a level mapping such that, in every program statement $p(\tilde{h}) \leftarrow Body$, the level of every predicate symbol in $Body$ is less than the level of p.

Proposition 6.1. If a program Pgm is hierarchical, then, for each predicate p of Pgm, there is a finite generalized constraint A_p which characterize p i.e.:

$$\forall \tilde{x} \ \left(p(\tilde{x}) \leftrightarrow A_p\right)$$

PROOF. By induction on $level(p)$. □

Definition 6.3. [14] A program is *3-recursive* if it is a definite CLP($\mathcal{Z}$) program such that all its predicates are defined as follows:

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

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.

Theorem 6.2. [14] If a program Pgm is 3-recursive then each predicate of Pgm can be characterized by a finite generalized arithmetic constraint A_p.

Let Pgm be a definite program. Consider the following system of implications:

$$\bigwedge_{p \in P} \left[\forall \tilde{x}(p(\tilde{x}) \rightarrow B_p(\tilde{x}))\right].$$

Proposition 6.2. If Pgm is hierarchical or 3-recursive, then the following proof method is complete:
for each p of Pgm:
- compute the finite generalized constraint A_p characterizing p,
- check if $A_p \models_\chi B_p$.

PROOF. Obvious by propositions 3.1, 6.1 and theorem 6.2. □

Automation of the above proof method follows the lines of section 5. In contrast, the proof method we propose in section 3 does not constitute a decision procedure

[3] i.e. a correct, complete and terminating procedure.

but our procedure can be applied to every definite program.

7. CONCLUSION

Let us first compare our technique with other works.

The proof method we propose can be seen as an instance of computational induction [25] (see also [24], chapter 2) specially adapted to constraint logic programming. Extended execution [22], [27] may prove a larger class of systems of implications although such systems usually require the assistance a human user to guide the search.

A whole chapter of [2] is devoted to partial correctness of pure logic programs. It summarizes the work of [13], [4], [1] and [3] about the subject in a single framework. Two notions of partial correctness are introduced.

The first one aims at determining the form of computed instances of a query and can be proved as follows. For every logic procedure, a specification in the form of a precondition and a postcondition is given. Then one has to check manually that the program to be verified is well-asserted, i.e. it satisfies the specifications. Now if an atom verifies its precondition, then all its computed instances (if any) are included in the intersection of the instances of the atom with its associated postcondition. The technique is applied in [2] to a sample of programs. It is the closest method to our work. If we can express the specifications as constraints then we can automatically prove a kind of well-assertedness for programs.

The second notion of partial correctness aims at computing the strongest postcondition of a query wrt a program, i.e. the set of all computed instances of a query. In [10], it has been applied to concurrent constraint programs (ccp for short) and in particular to CLP programs with dynamic scheduling. The properties to be proven are defined using constraints, user defined predicates and logical connectives. The basic ideas are first to map any CLP program with delay conditions to a semantically equivalent ccp where the delay declarations are embodied in "ask" actions. Then for such specific ccp programs, a proof system for determining the strongest postcondition is proved sound and "relatively" complete using the denotational semantics. "Relative" completeness means that completeness is guaranted only when it is possible to express the strongest postcondition in the language of properties. On the one hand, this approach is obviously more powerful than ours. It allows an extended format for the properties and takes into account the delay declarations associated to a CLP program. Moreover, one can express our proof method as a derived rule of the proof system presented in [10]. On the other hand, our method seems easier to understand, to apply and to implement. Furthermore, in section 6, we also investigate some *syntactic* conditions about CLP programs under which the strongest postcondition can be expessed as a finite generalized constraint.

Some positive results about decidability of various semantics of logic programs can be found in [28], where the author studies a related problem, namely the *testing problem*. It consists in checking whether or not the semantics of a Prolog program includes a given finite set of atoms. First the class of *bounded* programs is introduced. Bounded programs are operationally characterized as the programs s.t. for every ground query Q, the number of LD-refutations starting with Q is finite (note that the LD-tree for Q may contain infinite derivations). Then, the main result of the paper shows that the testing problem for bounded programs is decidable.

The author codes his decision procedure in Prolog as a variation of the well-known `VANILLA` meta-interpreter. We believe that it does not seem clear whether the above result can be easily lifted to CLP because it requires the introduction of new symbolic constants into the constraint solver. Moreover, our problem is clearly closer to the partial correctness problem than to the testing problem since we check that every atom of the semantics of a program verifies its associated property.

We now give some research directions in order to pursue this work. An obvious extension of the proof technique we present is the ability to process many-sorted structures. Negation inside the body of the clauses could be considered. The introduction of existential quantifiers in the consequent part of an implication seems to raise many interesting problems. Improving our approach with techniques based on Hoare logic, with suitable restrictions to ensure decidability, could also be investigated.

Finally let us summarize our work. We have presented a correct method for proving definite constraint logic program properties which can easily be implemented. Unfortunately any proof method is incomplete, as explained in section 6, but we would like to emphasize on the *generality* of our work: theorem 3.1 relies on properties of the structure χ which all domains that satisfy the framework described in [18] possess. The only requirement (which strengthens SC_2) concerns the atomic constraints c's appearing in a system of implications, namely: $\neg\, c \equiv\ c_1 \vee\ \ldots \vee\ c_n$. To our knowledge, for all constraint logic programming languages currently available, it does not seem to be too severe a restriction.

ACKNOWLEDGMENTS

We would like to thank Philippe Devienne who gave us the sketch of the proof of theorem 6.1 and anonymous referees for their helpful comments.

REFERENCES

1. K.R. Apt. Program verification and prolog. In E. Börger, editor, *Specification and Validation Methods for Programming Languages and Systems*, pages 55–95. Oxford University Press, Oxford, 1995.
2. K.R. Apt. *From Logic Programming to Prolog.* Prentice Hall, 1997.
3. K.R. Apt, M. Gabbrielli, and D. Pedreschi. A closer look at declarative interpretations. *Journal of Logic Programming*, 28(2):147–180, 1996.
4. A. Bossi and N. Cocco. Verifying correctness of logic programs. In *Proc. of TAPSOFT'89*, LNCS 352, pages 96–110. Springer-Verlarg, 1989.
5. K.L. Clark. Predicate logic as a computational formalism. Technical Report Doc 79/59, Logic Programming Group, Imperial College, London, 1979.
6. A. Colmerauer. Equations and inequations on finite and infinite trees. *Proc. of FGCS'84*, pages 85–99, 1984.
7. A. Colmerauer. An introduction to Prolog III. *CACM*, 33 (7):70–90, July 1990.
8. L. Colussi and E. Marchiori. Proving correctness of logic programs using axiomatic semantics. In *Proc. of ICLP'91*, pages 629–642. MIT Press, 1991.
9. E. Contejean. Solving linear diophantine constraints incrementally. In *Proc. of ICLP'93*. MIT Press, 1993.

10. F. DE BOER, M. GABBRIELLI, E. MARCHIORI and C. PALAMIDESSI. Proving concurrent constraint programs correct. To appear in *ACM-TOPLAS*, 1998.

11. P. DERANSART. Proof methods of declarative properties of definite programs. *Theoretical Computer Science*, pages 99–166, 1993.

12. M. DINCBAS, P. VAN HENTENRICK, H. SIMONIS, A. AGGOUN, T. GRAF, and F. BERTHIER. The constraint logic programming language CHIP. *Proc. of FGCS'88*, pages 693–702, 1988.

13. W. DRABENT and J. MALUSZYNSKI. Inductive assertion method for logic programs. *Theoretical Computer Science*, 59(1):133–155, 1988.

14. 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.

15. S. GRIGORIEFF. Décidabilité et complexité des théories logiques. *Logique et Informatique : une introduction, Collection Didactique, INRIA*, pages 7–97, 1989.

16. D. HOFSTADTER. *Gödel, Escher, Bach : an Eternal Golden Braid.* Basic Books, Inc., 1979.

17. H. HONG. Non-linear real constraints in constraint logic programming. In *LNCS 632*, pages 201–212. Springer Verlag, 1992.

18. J. JAFFAR and J.L. LASSEZ. Constraint logic programming. Technical Report 74, Monach University, Australia, 1986.

19. J. JAFFAR and M.J. MAHER. Constraint logic programming: a survey. *Journal of Logic Programming*, pages 503–581, 1994.

20. J. JAFFAR, S. MICHAYLOV, P.J. STUCKEY, and R.H.C. YAP. The CLP($\mathbb{R}$) language and system. In *Proc. of the ICLP'87.* MIT Press, 1987.

21. P.T. JOHNSTONE. *Notes on logic and set theory.* Cambridge Mathematical Textbooks. Cambridge University Press, 1986.

22. T. KANAMORI and H. FUJITA. Formulation of induction formulas in verification of prolog programs. In *Proc. of the 8th CADE*, LNCS, pages 281–299. Springer-Verlag, 1986.

23. J.M. LEVER. Proving program properties by means of SLS-resolution. In *Proc. of the ICLP'91*, pages 614–628. MIT Press, 1991.

24. J.W. LLOYD. *Foundations of Logic Programming.* Springer-Verlag, 1987.

25. Z. MANNA. *Mathematical theory of computation.* McGraw-Hill, 1974.

26. M.L. MINSKY. Recursive unsolvability of post's problem of 'tag' and other topics in the theory of Turing machines. *Ann. of Math.*, 74:437–455, 1961.

27. S. RENAULT. Generalized extended execution for normal programs. In *Proc. of Lopstr'94*, LNCS. Springer-Verlag, 1994.

28. S. RUGGIERI. Decidability of logic program semantics and application to testing. In *Proc. of Plilp'96*, LNCS 1140. Springer-Verlag, 1996.

29. J.C. SHEPHERDSON. Unsolvable problems for SLDNF resolution. *Journal of Logic Programming*, pages 19–22, 1991.

30. C. WALINSKY. CLP(Σ^*): Constraint logic programming with regular sets. In *Proc. of ICLP'89*, pages 181–196. MIT Press, 1989.