Debugging Systems for Constraint Programming

ESPRIT 22532

Task T.WP2.1: **Declarative Debugging in Constraint Programming**

Delivrable D.WP2.1.M2.1-2

Correctness and Completeness of CLP Semantics revisited with (Co-)Induction

Gérard Ferrand and Alexandre Tessier

LIFO, rue Léonard de Vinci, BP 6759, 45067 Orléans Cedex 2, France
Gerard.Ferrand@lifo.univ-orleans.fr, Alexandre.Tessier@inria.fr
http://www.univ-orleans.fr/SCIENCES/LIFO/Members/tessier/

`http://discipl.inria.fr/`

Abstract

We propose a reformulation of the constraint logic program semantics in terms of positive and negative semantics, using a uniform inductive framework. It is a natural and elegant way to express and study correctness and completeness results. In particular we state a completeness for negative semantics by using some infinite sets of constraints. This theoretical framework is an original extension of the "Grammatical View of Logic Programming".

1 Introduction

Because of the non determinism of Constraint Logic Programming (CLP), the sequence of answers to a goal may be read in two way. Two levels of computation may be considered:

- First level is the SLD derivation. A finite SLD derivation for a goal $\leftarrow g$ computes an answer constraint c.

  For example if the program P is the three clauses (pure Prolog[1])

  $$p(X) \leftarrow q(X)$$
  $$q(a) \leftarrow$$
  $$q(b) \leftarrow$$

  then one of the computed answer constraints to the goal $\leftarrow p(X)$ is $X = a$.

  At this level, the *correctness of the computation* is expressed by the fact that is c is an answer to $\leftarrow g$ then $c \rightarrow g$ is true in the program semantics.

  In the example it is the formula $X = a \rightarrow p(X)$, that is $p(a)$. $p(a)$ is true in the *least Herbrand model* of P, $p(a)$ is a logical consequence of P.

  At this level, the *completeness of the computation* is expressed by the fact that if $c \rightarrow g$ is true in the program semantics then $c \rightarrow \bigvee c_i$ is true in the constraint semantics, where the c_i are the answers to the goal $\leftarrow g$.

[1] Note that Prolog is a particular case of CLP. Here Prolog is sufficient to understand.

In the example, we can consider that the constraint c is a conjunction of equalities and the formula $c \rightarrow \bigvee c_i$ is the formula $c \rightarrow X = a \vee X = b$.

The constraint semantics may be formalized by the Herbrand domain equiped with the equality, or by the *complete Clark equality theory* (CET), giving the underlying language ([4]).

For example, if the constants of the underlying language are a and b the formula $\mathit{true} \rightarrow p(X)$ (here c is the empty conjunction), that is $\forall X p(X)$, is true in the least Herbrand model of P and the formula $\mathit{true} \rightarrow X = a \vee X = b$ is true in the Herbrand domain.

If there exists other constants then there less constraints c such that $c \rightarrow p(X)$ is true in the least Herbrand model of P. Moreover, because of some particular properties of this domain, for each of these constraints c, one of the formulae $c \rightarrow X = a$ or $c \rightarrow X = b$ is true in the Herbrand domain.

In terms of substitutions we see that the terms such that $p(t)$ is a consequence of P are the terms a and b.

- Second level is the SLD tree (or search tree). A finite SLD tree for the goal $\leftarrow g$ computes an answer C which is the disjunction of the answers obtained at the first level.

  In the example, C is $X = a \vee X = b$. This way to read the sequence of answers obtained at the first level as one answer at the second level is similar to what is obtained in ISO Prolog using the built-in `findall/3`.

  At this level, *the correctness of the computation* is expressed by the fact that if C is an answer to $\leftarrow g$ then $g \rightarrow C$ is true in the program semantics.

  In the example, the formula $p(X) \rightarrow X = a \vee X = b$ is true in the least Herbrand model of P. However this formula is not a consequence of P. It is a consequence of the set of formulas $\{p(X) \rightarrow q(X), q(X) \rightarrow X = a \vee X = b\}$ and also of the set of formulas $\{p(X) \rightarrow q(X), q(X) \rightarrow X = a \vee X = b\}$ usualy denoted by P^*.

  In general P^* is not sufficient, we have to add a *constraint theory* (for example CET) to obtained the completion of P.

  At this second level, a particular case is $C = \mathit{false}$, $g \rightarrow C$ is $\neg g$, it is *finite failure*.

In the example, with the new goal $\leftarrow p(f(X))$, C is the empty disjunction, that is *false*. $\forall X \neg p(f(X))$ is a consequence of $\{p(X) \rightarrow q(X), q(X) \rightarrow X = a \vee X = b\}$ increased with CET because $\neg(f(X) = a \vee f(X) = b)$ is a consequence of CET.

In some sense, the answer C at the second level means that there is no more answers at the first level except those in C.

In the example, with the goal $\leftarrow p(X)$, the answer at the second level $(X = a \vee X = b)$ means that there is no other answers at the first level that $X = a$ and $X = b$.

It is the reason why the first level is called *positive level* and the second level is called *negative level*. Unfortunately, there is no *completeness* when there is no finite SLD tree.

The more trivial example is obtained with $C = true$ and the program reduced to $p \leftarrow p$.

There is just a result, known as completeness of negation by failure when $\neg g$ is in the program semantics.

This distinction between positive and negative computation is a useful clarification motivated by validation purposes, in particular in debugging [16] where two kinds of error symptoms are considered: wrong answer (positive symptom) and missing answer (negative symptom) [7, 6].

In this paper, we propose a uniform framework to formalize these two semantics[2] and to clearly express correctness and completeness results. In particular we state a completeness for negative semantics by using some infinite sets of constraints. From an operational point of view these infinite sets are naturally introduced by infinite computations. From a denotational point of view they can be defined by *co-induction*.

The usefulness of induction (and least fixpoint) for semantics is well-known especially for CLP (basically a program can be seen as an inductive definition of relations [15]). Co-induction (and greatest fixpoint) in general is used to deal with some non-well-founded objects (for example [13, 2] uses this principle in another context). [14] used a "greatest fixpoint" semantics in

[2]The *positive semantics* (first level) which provides a positive knowledge on the program and the *negative semantics* (second level) which provides a negative knowledge on the program.

logic programming for infinite computations. [10] also deals with infinite computations.

In this paper we show how positive and negative semantics have natural and elegant definitions and properties using induction and co-induction. Both are studied at the same time and in the same way. We found again, easily, the classical results [11, 9] of correctness and completeness of the positive level. Moreover we clearly give the necessary hypothesis on the constraint solver. The results of negative correctness and completeness we add using infinite sets of contraints are also easily obtained and are very natural in this framework.

Another contribution of this paper is an extension of the *grammatical view of logic programming* [3] not only to deal with CLP, taking into account incompleteness of real solvers, but also to deal with negative semantics using infinite skeletons. Finite skeletons are a notion coming from syntactic analysis useful to describe computations and to obtain confluence for free. From a theoretical viewpoint we can consider that the "grammatical view" consist in the systematic use of (finite or infinite) skeletons in relation with induction (or co-induction).

2 Preliminaries

Let us consider once and for all four sets which define the program language:

- an infinite set of *variables* V;
- a set of *function symbols* Σ;
- a set of *constraint predicate symbols* Ξ;
- a set of *program predicate symbols* Π.

Each symbol is equipped with its arity.

A *pure atom* (in short *atom*) is an atomic formula $p(\overline{X})$ built over the first order language $\mathcal{L}(V, \emptyset, \Pi)$, where $\overline{X}$ is a sequence of distinct variables.

The set of *basic constraints* is a subset of the first order language $\mathcal{L}(V, \Sigma, \Xi)$ closed under variable renaming. A *constraint* is a formula of the least set

which contains the set of basic constraints and is closed under conjunction. Here it is sufficient to identify a conjunction of basic constraints with the set of basic constraints of the conjunction. So a constraint is viewed as a finite set of basic constraints.

Among V we distinguish the set of program variables $V_P = \{X_{i,j} \mid i \geq 0, j \geq 1\} \cup \{Y_j \mid j \geq 1\}$.

A *clause* is a formula $p_0(\overline{X_0}) \leftarrow C \wedge p_1(\overline{X_1}) \wedge \cdots \wedge p_n(\overline{X_n})$, where

- for each $i = 0, \ldots, n$: n_i is the arity of $p_i \in \Pi$, $\overline{X_i} = X_{i,1}, \ldots, X_{i,n_i}$;
- C is a constraint such that the set of the n_C free variables of C which are not in $\bigcup_{i=0,\ldots,n}\{X_{i,j} \mid 1 \leq j \leq n_i\}$ is $\{Y_j \mid 1 \leq j \leq n_C\}$.

The fact that we suppose that all the variables of the atoms are distinct is not a limitation because we assume that all the links between them are made into the constraint (for example, using equalities). V_P is just a way to formalize the name of variables in the clauses. Each program can be written using like that.

A *program* is a family of clauses. In this paper P is a program. The set of indexes of P is denoted by $\delta(P)$.

A *name of clause* is a member of $\delta(P)$. The *definition* of $p \in \Pi$ in P is the sub-family of clauses of P whose head predicate symbol is p; it is indexed by the subset $\delta_p(P) \subseteq \delta(P)$. We assume $\delta_p(P)$ is finite for each $p \in \Pi$. The clause whose name (i.e. index) is f is denoted by $clause(P, f)$.

We use the following notations, where F is a first order formula built over the language $\mathcal{L}(V, \Sigma, \Pi \cup \Xi)$ and $\tilde{X} = \{X_1, \ldots, X_n\} \subseteq V$:

- $var(F)$ denotes the set of free variables of F;
- $\forall F$ denotes the universal closure of F;
- $\exists F$ denotes the existential closure of F;
- $\exists_{\tilde{X}} F$ denotes $\exists X_1 \cdots \exists X_n F$;
- $\exists_{-\tilde{X}} F$ denotes $\exists_{var(F)\setminus\tilde{X}} F$.

For each $p \in \Pi$, we define

- $\mathrm{IF}(P,p) = \forall(p(\overline{X_0}) \leftarrow \bigvee_{f \in \delta_p(P)} \exists_{-\overline{X_0}} B_f)$;
- $\mathrm{FI}(P,p) = \forall(p(\overline{X_0}) \rightarrow \bigvee_{f \in \delta_p(P)} \exists_{-\overline{X_0}} B_f)$;
- $\mathrm{IFF}(P,p) = \mathrm{IF}(P,p) \wedge \mathrm{FI}(P,p)$.

where for each $f \in \delta_p(P)$: $\mathit{clause}(P,f) = p(\overline{X_0}) \leftarrow B_f$.

Note that when the definition of p is empty: $\mathrm{FI}(P,p) = \forall(\neg p(\overline{X_0}))$.

We define the *if-part* of P, the *only-if-part* of P and finally the *if-and-only-if-part* of P (the *completion* without constraint theory) as follows:

- $\mathrm{IF}(P) = \{\mathrm{IF}(P,p) \mid p \in \Pi\}$;
- $\mathrm{FI}(P) = \{\mathrm{FI}(P,p) \mid p \in \Pi\}$;
- $\mathrm{IFF}(P) = \{\mathrm{IFF}(P,p) \mid p \in \Pi\}$.

A constraint logic programming system use, in general, an adaptation of the SLD-resolution to compute the set of answers to a goal as described in several papers [8, 9]. Our aim is to give a logical meaning to the set of answers (or to each answer alone).

The set of proof variables[3] is the subset of V: $V_S = \{X^\nu_{i,j} \mid i \geq 0, j \geq 1, \nu \in \mathbb{N}^*_+\} \cup \{Y^\nu_j \mid j \geq 1, \nu \in \mathbb{N}^*_+\}$. This set is used to normalize variable renaming during a computation.

A goal is an atom $p(X^\varepsilon_{0,1}, \ldots, X^\varepsilon_{0,n})$. Again, this is not a limitation, in comparison with real systems which allow more general goal, because we can always add a new predicate symbol to Π and a clause to P (whose body is the general goal and head is built with the new predicate symbol and the free variables of the body).

A *store* is a (finite or infinite[4]) set of basic constraints whose free variables are in V_S. A store S is satisfiable if there exists a valuation v such that for each $c \in S$: $v(c) = \mathit{true}$. Then we say v is a solution of S.

[3] $\mathbb{N}^*_+$ is the set of finite sequences of positive integers.

[4] We will consider "infinite answers" later.

A *pre-interpretation* $\mathcal{D}$ is composed of:

- a non empty set $\mathbb{D}$ (the domain);
- for each function symbol $\varphi \in \Sigma$ with arity n, a mapping $\varphi_{\mathcal{D}}$ from $\mathbb{D}^n$ to $\mathbb{D}$;
- for each constraint predicate symbol $\rho \in \Xi$ with arity n, a relation $\rho_{\mathcal{D}}$ over $\mathbb{D}^n$.

A $\mathcal{D}$-*atom* is a "pseudo-atom" denoted $p(d_1, \ldots, d_n)$, where $p \in \Pi$ with arity n and each $d_i \in \mathbb{D}$. The $\mathcal{D}$-*base* is the set of all $\mathcal{D}$-atoms. A $\mathcal{D}$-*interpretation* I is a subset of the $\mathcal{D}$-base. It is identified with an interpretation which expands[5] $\mathcal{D}$.

A *valuation* v is a mapping from V to $\mathbb{D}$ which is extended, as usual, to mappings also denoted by v:

- from the set of terms to $\mathbb{D}$;
- from the set of constraints to $\{true, false\}$;
- from the set of atoms to the $\mathcal{D}$-base.

We use the classical notions of $\mathcal{D}$-model and $\mathcal{D}$-consequence (denoted with $\models_{\mathcal{D}}$).

Let $T_P^{\mathcal{D}}$ be the *immediate consequence operator* over the set of $\mathcal{D}$-interpretations defined by:

$$T_P^{\mathcal{D}}(I) = \left\{ v(a) \;\middle|\; \begin{array}{l} v \text{ is a valuation,} \\ \text{there exists } a \leftarrow C \wedge a_1 \wedge \cdots \wedge a_n \in P, \\ v(C) = true \text{ and each } v(a_i) \in I \end{array} \right\}$$

We recall that, where I is a $\mathcal{D}$-interpretation:

- $T_P^{\mathcal{D}}(I) \subseteq I$ iff I is a $\mathcal{D}$-model of $\mathrm{IF}(P)$;

[5] An interpretation of the whole language which is an expansion of $\mathcal{D}$, that is, which add to $\mathcal{D}$, for each program predicate symbol $p \in \Pi$ with arity n, a relation p_I over $\mathbb{D}^n$.

- $T_P^{\mathcal{D}}(I) \supseteq I$ iff I is a $\mathcal{D}$-model of $\mathrm{FI}(P)$;
- $T_P^{\mathcal{D}}(I) = I$ iff I is a $\mathcal{D}$-model of $\mathrm{IFF}(P)$.

$T_P^{\mathcal{D}}$ is monotonic (and continuous) so it has a least and a greatest fixpoint.

- $lfp(T_P^{\mathcal{D}})$ is the least $\mathcal{D}$-model of $\mathrm{IF}(P)$ (and $\mathrm{IFF}(P)$);
- $gfp(T_P^{\mathcal{D}})$ is the greatest $\mathcal{D}$-model of $\mathrm{FI}(P)$ (and $\mathrm{IFF}(P)$).

The aim of this paper is to formalize the logical relations between a goal and the answer constraints to this goal. Thus, we study the logical consequences of $\mathrm{IFF}(P)$ of the form $a \leftrightarrow F$ where a is an atom and F is a formula built over the constraint language $\mathcal{L}(V, \Sigma, \Xi)$. Obviously $\models \forall(a \leftrightarrow F) \leftrightarrow \forall((a \leftarrow F) \wedge (a \rightarrow F))$. Note also that $\models \forall(a \leftarrow \bigvee_{1 \leq i \leq n} F_i) \leftrightarrow \bigwedge_{1 \leq i \leq n} \forall(a \leftarrow F_i)$.

Using $lfp(T_P^{\mathcal{D}})$ and $gfp(T_P^{\mathcal{D}})$ we obtain the following equivalences, where F is a formula over the constraint language[6] $\mathcal{L}(V, \Sigma, \Xi)$, a is an atom, $\mathcal{D}$ is a pre-interpretation and $\mathcal{T}$ is a constraint theory[7]:

- $\mathrm{IFF}(P) \models_{\mathcal{D}} \forall(a \leftarrow F)$ iff $\mathrm{IF}(P) \models_{\mathcal{D}} \forall(a \leftarrow F)$,
  thus $\mathrm{IFF}(P) \models \forall(a \leftarrow F)$ iff $\mathrm{IF}(P) \models \forall(a \leftarrow F)$,
  and $\mathcal{T}, \mathrm{IFF}(P) \models \forall(a \leftarrow F)$ iff $\mathcal{T}, \mathrm{IF}(P) \models \forall(a \leftarrow F)$;
- $\mathrm{IFF}(P) \models_{\mathcal{D}} \forall(a \rightarrow F)$ iff $\mathrm{FI}(P) \models_{\mathcal{D}} \forall(a \rightarrow F)$,
  thus $\mathrm{IFF}(P) \models \forall(a \rightarrow F)$ iff $\mathrm{FI}(P) \models \forall(a \rightarrow F)$,
  and $\mathcal{T}, \mathrm{IFF}(P) \models \forall(a \rightarrow F)$ iff $\mathcal{T}, \mathrm{FI}(P) \models \forall(a \rightarrow F)$.

3 Two Inductive Definitions

This section give the preliminaries definitions used in section 5 to prove correctness and completeness of the operational semantics (positive and negative) of CLP systems. There is no notion of computation here.

In a first time, let us consider the following set of rules over the set of pair $(t : p(\overline{d}))$, where t is a tree[8] labeled by $\delta(P)$ and $p(\overline{d})$ is a $\mathcal{D}$-atom:

[6] The results are wrong, in general, when F is a formula over $\mathcal{L}(V, \Sigma, \Pi \cup \Xi)$.
[7] A constraint theory is a theory over the constraint language.
[8] A tree rooted by f whose sub-trees are $t_1, \ldots, t_n$ is denoted by $f(t_1, \ldots, t_n)$.

For each *name of clause* $f \in \delta(P)$, let $clause(P,f) = p_0(\overline{X_0}) \leftarrow C \wedge p_1(\overline{X_1}) \wedge \cdots \wedge p_n(\overline{X_n})$, for each valuation v solution of C, for each trees $t_1, \ldots, t_n$, we consider the rule:

$$\frac{t_1 : v(p_1(\overline{X_1})) \quad \cdots \quad t_n : v(p_n(\overline{X_n}))}{f(t_1, \ldots, t_n) : v(p_0(\overline{X_0}))}$$

The operator associated with the previous set of rules is denoted by $\mathcal{T}_P^{\mathcal{D}}$. It is monotonic (and continuous) so has a least and a greatest fixpoint.

A member $t : p(\overline{d})$ of the least fixpoint of $\mathcal{T}_P^{\mathcal{D}}$ is such that t is finite ($\mathcal{T}_P^{\mathcal{D}}$ is continuous and the "fact rules" have a tree reduced to one node). t corresponds to the notion of skeletons [3] and corresponds exactly to the notion of complete non rejected skeletons defined by terms over $\delta(P)$ in [6].

A member $t : p(\overline{d})$ of the greatest fixpoint of $\mathcal{T}_P^{\mathcal{D}}$ is such that t may be infinite. t corresponds to the more general notion of infinite skeletons.

The operator $\mathcal{T}_P^{\mathcal{D}}$ is strongly linked to the operator $T_P^{\mathcal{D}}$:

Lemma 1 Links between $\mathcal{T}_P^{\mathcal{D}}$ and $T_P^{\mathcal{D}}$.

- $\{p(\overline{d}) \mid$ *there exists* $t, t : p(\overline{d}) \in lfp(\mathcal{T}_P^{\mathcal{D}})\} = lfp(T_P^{\mathcal{D}})$ *(the least* $\mathcal{D}$*-model of* $\mathrm{IF}(P)$*);*
- $\{p(\overline{d}) \mid$ *there exists* $t, t : p(\overline{d}) \in gfp(\mathcal{T}_P^{\mathcal{D}})\} = gfp(T_P^{\mathcal{D}})$ *(the greatest* $\mathcal{D}$*-model of* $\mathrm{FI}(P)$*).*

Proof. Straightforward application of inductive definitions. We have internalized[9] an encoding of proofs: in $(t : p(\overline{d}))$, t codes a proof of $p(\overline{d})$. □

$\mathcal{D}$-atoms are not syntactic notions. In fact, computations only deal with syntactic notions which are constraints and atoms. The previous inductive definition was given for educational purpose and because it is used in the

[9] This is similar to the constructive theory of types where λ-terms codes proofs of formulae, but only based on modus ponens here.

proofs of the results of next sections. So, now we introduce another inductive definition to prove "pseudo-formulae" of the language. But a more elaborated formalism will be useful to take into account "infinite answers":

A *positive sequent* is a pair $S \vdash p$ where S is a store and $p \in \Pi$.

A positive sequent $S \vdash p$ is *valid* in a $\mathcal{D}$-interpretation I if for each solution v of S: $v(p(X^{\varepsilon}_{0,1}, \ldots, X^{\varepsilon}_{0,n})) \in I$ (n is the arity of p). When S is finite this is equivalent to $I \models \bigwedge_{c \in S} c \rightarrow p(X^{\varepsilon}_{0,1}, \ldots, X^{\varepsilon}_{0,n})$.

A *negative sequent* is a pair $p \vdash \mathcal{Z}$ where $\mathcal{Z}$ is a *set* of stores and $p \in \Pi$.

A negative sequent $p \vdash \mathcal{Z}$ is *valid* in a $\mathcal{D}$-interpretation I if for each valuation v such that $v(p(X^{\varepsilon}_{0,1}, \ldots, X^{\varepsilon}_{0,n})) \in I$: there exists $S \in \mathcal{Z}$ and there exists a solution v' of S such that for each $1 \leq i \leq n$: $v(X^{\varepsilon}_{0,i}) = v'(X^{\varepsilon}_{0,i})$. When $\mathcal{Z}$ is a finite set of finite stores this is equivalent to $I \models p(X^{\varepsilon}_{0,1}, \ldots, X^{\varepsilon}_{0,n}) \rightarrow \bigvee_{S \in \mathcal{Z}} \exists_{-\{X^{\varepsilon}_{0,1}, \ldots, X^{\varepsilon}_{0,n}\}} \bigwedge_{c \in S} c$.

Let F be a formula such that $var(F) \subseteq V_P$ (the set of program variables) and let $\nu \in \mathbb{N}^*_+$. F^{ν} denotes the variant of F where $X^{\nu}_{i,j}$ takes the place of $X_{i,j}$ and Y^{ν}_j takes the place of Y_j in F.

Let F be a formula such that $var(F) \subseteq V_S$ (the set of proof variables) and let $k \in \mathbb{N}_+$. F^{+k} denotes the variant of F where[10] $X^{\nu k}_{i,j}$ takes the place of $X^{\nu}_{i,j}$ and $Y^{\nu k}_j$ takes the place of Y^{ν}_j in F. If S is a set of formulae $S^{+k} = \{F^{+k} \mid F \in S\}$.

Now we give the second inductive definition suitable to prove positive (and negative) sequents.

Let us consider the following set of rules over the set of pair $(t : S \vdash p)$, where t is a tree labeled by $\delta(P)$ and $S \vdash p$ is a positive sequent. For each *name of clause* $f \in \delta(P)$, let $clause(P, f) = p_0(\overline{X_0}) \leftarrow C \wedge p_1(\overline{X_1}) \wedge \cdots \wedge p_n(\overline{X_n})$, for each trees $t_1, \ldots, t_n$ and for each stores $S_1, \ldots, S_n$, we consider the rule:

$$\frac{t_1 : S_1 \vdash p_1 \quad \cdots \quad t_n : S_n \vdash p_n}{f(t_1, \ldots, t_n) : S_0 \vdash p_0}$$

where $S_0 = C^{\varepsilon} \cup S_1^{+1} \cup \cdots \cup S_n^{+n} \cup S'$, with $S' = \bigcup_{1 \leq i \leq n} \bigcup_{1 \leq j \leq n_i} \{X^{\varepsilon}_{i,j} = X^{i}_{0,j}\}$, where each n_i is the arity of p_i.

[10] νk is the concatenation of ν and k.

The operator associated with this set of rules is denoted by $\mathcal{T}_P$. It is monotonic (and continuous) so has a least and a greatest fixpoint.

A member $t : S \vdash p$ of the least fixpoint of $\mathcal{T}_P$ is such that t and S are finite. t codes a proof of $S \vdash p$ (note that $S \vdash p$ only depends on t), S is equivalent to the usual notion of answer constraint [9] without taking into account the problem of unsatisfiability.

A member $t : S \vdash p$ of the greatest fixpoint of $\mathcal{T}_P$ is such that t and S may be infinite. Again, $S \vdash p$ only depends on t. $S \vdash p$ is the sequent whose skeleton is t.

The operators $\mathcal{T}_P^{\mathcal{D}}$ and $\mathcal{T}_P$ are linked by the fact that if $t : S \vdash p \in \mathit{lfp}(\mathcal{T}_P)$ (resp. $\mathit{gfp}(\mathcal{T}_P)$) and if there exists a solution v of S then $t : v(p(\overline{X_0^\varepsilon})) \in \mathit{lfp}(\mathcal{T}_P^{\mathcal{D}})$ (resp. $\mathit{gfp}(\mathcal{T}_P^{\mathcal{D}})$). Vice versa, if $t : v(p(\overline{X_0^\varepsilon})) \in \mathit{lfp}(\mathcal{T}_P^{\mathcal{D}})$ (resp. $\mathit{gfp}(\mathcal{T}_P^{\mathcal{D}})$) then there exists a store S and a solution v' of S, $v(p(\overline{X_0^\varepsilon})) = v'(p(\overline{X_0^\varepsilon}))$, such that $t : S \vdash p \in \mathit{lfp}(\mathcal{T}_P)$ (resp. $\mathit{gfp}(\mathcal{T}_P)$).

4 Logical View of the Inductive Definition

In this section, we give a logical meaning (correctness/completeness) to the inductive definition of the previous section. We start by its correctness, first from the positive viewpoint and next from the negative viewpoint.

Lemma 2 Positive correctness.
Let I be a $\mathcal{D}$-model of $\mathrm{IF}(P)$. *If $t : S \vdash p \in \mathit{lfp}(\mathcal{T}_P)$ then $S \vdash p$ is valid in I (i.e* $\mathrm{IF}(P) \models_{\mathcal{D}} \bigwedge_{c \in S} c \rightarrow p(\overline{X_0^\varepsilon})$ *because S is finite).*

Proof. Inductively. □

Corollary 3 Positive correctness.

- *If $t : S \vdash p \in \mathit{lfp}(\mathcal{T}_P)$ then $\mathcal{T}, \mathrm{IF}(P) \models \bigwedge_{c \in S} c \rightarrow p(\overline{X_0^\varepsilon})$, where $\mathcal{T}$ is a constraint theory.*
- *If $t : S \vdash p \in \mathit{lfp}(\mathcal{T}_P)$ then* $\mathrm{IF}(P) \models \bigwedge_{c \in S} c \rightarrow p(\overline{X_0^\varepsilon})$.

We denote by $\mathcal{Z}^-(p)$ the set $\{S \mid \text{there exists } t \text{ such that } t : S \vdash p \in gfp(\mathcal{T}_P)\}$.

The negative sequent associated with $p \in \Pi$ is $p \vdash \mathcal{Z}^-(p)$.

Lemma 4 Negative correctness.
Let I be a $\mathcal{D}$-model of $\mathrm{FI}(P)$. *$p \vdash \mathcal{Z}^-(p)$ is valid in I.*

Proof. Use lemma 1. □

Corollary 5 Negative correctness.
If $\mathcal{Z}^-(p)$ is a finite set of finite stores then:

- $\mathrm{FI}(P) \models_{\mathcal{D}} p(\overline{X_0^\varepsilon}) \rightarrow \bigvee_{S \in \mathcal{Z}^-(p)} \exists_{-\overline{X_0^\varepsilon}} \bigwedge_{c \in S} c$;
- $\mathcal{T}, \mathrm{FI}(P) \models p(\overline{X_0^\varepsilon}) \rightarrow \bigvee_{S \in \mathcal{Z}^-(p)} \exists_{-\overline{X_0^\varepsilon}} \bigwedge_{c \in S} c$, *where $\mathcal{T}$ is a constraint theory;*
- $\mathrm{FI}(P) \models p(\overline{X_0^\varepsilon}) \rightarrow \bigvee_{S \in \mathcal{Z}^-(p)} \exists_{-\overline{X_0^\varepsilon}} \bigwedge_{c \in S} c$.

In order to generalize the previous corollary, we define the notion of negative sequent *consequence* of a set of formulae. A negative sequent $p \vdash \mathcal{Z}$ is a consequence of Γ when it is valid in all models of Γ.

Corollary 6 Negative correctness.
$p \vdash \mathcal{Z}^-(p)$ is consequence of $\mathcal{T} \cup \mathrm{FI}(P)$, where $\mathcal{T}$ is a constraint theory (a particular case is $p \vdash \mathcal{Z}^-(p)$ is consequence of $\mathrm{FI}(P)$*).*

In order to express the completeness of the inductive definition, we introduce a notion similar to the notion of sequents, but which only refers to formulae over the constraint language.

A *negative cover* is a pair $F \vdash \mathcal{Z}$, where F is a formula over the constraint language and $\mathcal{Z}$ is a set of stores.

A negative cover $F \vdash \mathcal{Z}$ is *valid* in the pre-interpretation $\mathcal{D}$ if for each solution v of F there exists a store $S \in \mathcal{Z}$ such that v is a solution of S. The finite case corresponds to $F \rightarrow \bigvee_{S \in \mathcal{Z}} \bigwedge_{c \in S} c$ is valid in $\mathcal{D}$. A negative cover

is consequence of a constraint theory $\mathcal{T}$ if it is valid in each model $\mathcal{D}$ of $\mathcal{T}$. The notion of negative cover, when each store of $\mathcal{Z}$ is finite, corresponds to the usual notion of cover in the classical completeness theorems [11, 9, 18].

A *positive cover* is a pair $S \vdash F$, where F is a formula over the constraint language and S is a store.

A positive cover $S \vdash F$ is *valid* in the pre-interpretation $\mathcal{D}$ if for each solution v of S: v is a solution of F. The finite case corresponds to $\bigwedge_{c \in S} c \rightarrow F$ is valid in $\mathcal{D}$. A positive cover is consequence of a constraint theory $\mathcal{T}$ if it is valid in each model $\mathcal{D}$ of $\mathcal{T}$.

We denote by $\mathcal{Z}^+(p)$ the set $\{S \mid \text{there exists } t \text{ such that } t : S \vdash p \in \mathit{lfp}(\mathcal{T}_P)\}$.

Lemma 7 Positive completeness (positive cover, positive interpolation).
Let F be a formula built over the constraint language.
If $\mathrm{IF}(P) \models_{\mathcal{D}} F \rightarrow p(\overline{X_0^\varepsilon})$ *then the negative cover* $\exists_{-\overline{X_0^\varepsilon}} F \vdash \mathcal{Z}^+(p)$ *is valid in* $\mathcal{D}$.

Proof. Use the least $\mathcal{D}$-model of $\mathrm{IF}(P)$ and lemma 1. □

From a logical point of view, this property may be seen as an interpolation. $\mathcal{Z}^+(p)$ is halfway between $\exists_{-\overline{X_0^\varepsilon}} F$ and $p(\overline{X_0^\varepsilon})$.

Corollary 8 Positive completeness and compactness.
Let F be a formula built over the constraint language.
If $\mathcal{T}, \mathrm{IF}(P) \models F \rightarrow p(\overline{X_0^\varepsilon})$ *then the negative cover* $\exists_{-\overline{X_0^\varepsilon}} F \vdash \mathcal{Z}^+(p)$ *is consequence of* $\mathcal{T}$, *and there exists a finite part*[11] $\mathcal{Z} \subseteq \mathcal{Z}^+(p)$ *such that* $\exists_{-\overline{X_0^\varepsilon}} F \vdash \mathcal{Z}$ *is consequence of* $\mathcal{T}$, *i.e.* $\mathcal{T} \models \exists_{-\overline{X_0^\varepsilon}} F \rightarrow \bigvee_{S \in \mathcal{Z}} \bigwedge_{c \in S} c$ *(compactness of the first order logic and each* $S \in \mathcal{Z}^+(p)$ *is finite).*

Lemma 9 Negative completeness (negative cover, negative interpolation).
Let F be a formula built over the constraint language.
If $\mathrm{FI}(P) \models_{\mathcal{D}} p(\overline{X_0^\varepsilon}) \rightarrow F$ *then for each* $S \in \mathcal{Z}^-(p)$*: the positive cover* $S \vdash \exists_{-\overline{X_0^\varepsilon}} F$ *is valid in* $\mathcal{D}$.

[11] Reduced to a singleton when the domain has the Independence of Negated Constraints [12] (see the example of the Introduction).

Proof. Use the greatest $\mathcal{D}$-model of FI(P) and lemma 1. □

Again, $\mathcal{Z}^-(p)$ is halfway (interpolation formula) between $p(\overline{X_0^\varepsilon})$ and $\exists_{-\overline{X_0^\varepsilon}}F$.

Corollary 10 Negative completeness and compactness.
Let F be a formula built over the constraint language.
If $\mathcal{T}$, FI(P) $\models p(\overline{X_0^\varepsilon}) \rightarrow F$ then for each $S \in \mathcal{Z}^-(p)$: the positive cover $S \vdash \exists_{-\overline{X_0^\varepsilon}}F$ is consequence of $\mathcal{T}$ and there exists a finite part S' of S such that $S' \vdash \exists_{-\overline{X_0^\varepsilon}}F$ is consequence of $\mathcal{T}$, i.e. $\mathcal{T} \models \bigwedge_{c \in S'} c \rightarrow \exists_{-\overline{X_0^\varepsilon}}F$.

Through the correctness and completeness lemmas we have given a logical meaning to positive and negative sequents. The point is that real constraint logic programming systems do not compute all this sequents. Systems end the computation when they detect that the current store is unsatisfiable, thanks to the "constraint solver". Thus a lot of positive sequents are never computed by the system. Indeed, if $S \vdash p \in \mathit{lfp}(\mathcal{T}_P)$ and S is unsatisfiable then $S \rightarrow p(\overline{X_0^\varepsilon})$ is always true and not interesting because independent of P. In order to formalize the constraint solver we introduce, in the next section, the *reject criterion*.

5 Correctness and Completeness

In this section, we study correctness and completeness of the classical theoretical operational semantics of CLP systems.

A *reject criterion* is a set of stores RC. A store S is *rejected* if $S \in$ RC. Real systems only handle finite stores, so we first define the relation RC over finite stores.

The reject criterion verifies the following properties, where S is a finite store:

- if $S \in$ RC then for each variable renaming θ: $S\theta \in$ RC;
- if $S \in$ RC then for each finite store S': $S \cup S' \in$ RC;
- $\emptyset \notin$ RC (the empty store $\emptyset$ is the logic constant *true*).

A reject criterion RC is *correct* wrt the pre-interpretation $\mathcal{D}$ (resp. the constraint theory $\mathcal{T}$) when $S \in \text{RC}$ implies that there exists no solution of S in $\mathcal{D}$ (resp. there exists no solution of S in any model of $\mathcal{T}$).

A reject criterion RC is *complete* wrt the pre-interpretation $\mathcal{D}$ (resp. the constraint theory $\mathcal{T}$) when there exists no solution of S in $\mathcal{D}$ (resp. there exists no solution of S in any models of $\mathcal{T}$) implies that $S \in \text{RC}$.

For example, the reject criterion may be defined by a relation *consistent* and a function *infer* (see [9]) as follows: a store S is rejected if $\mathit{infer}(\emptyset, S) = (S_1, S_2)$ and $S_1 \notin \mathit{consistent}$.

We assume the reader is familiar with the notions of SLD-derivation and SLD-tree wrt a "constraint solver"[9], formalized here by a reject criterion. A SLD derivation may be seen as the construction of a skeleton, then it is defined as a sequence of finite partial skeletons[12] [17]: a SLD tree regroups together all the derivations for a goal according to a computation rule [19].

5.1 Positive Computation

A *positive computation* is a success SLD-derivation. To each success SLD-derivation for the goal $p(\overline{X_0^\varepsilon})$ corresponds a member $t : S \vdash p$ of $\mathit{lfp}(\mathcal{T}_P)$ such that $S \notin \text{RC}$. Then the positive sequent $S \vdash p$ is called a RC-*computed positive sequent*. The finite skeleton t characterizes the clauses used by the derivation and S is the store computed by the derivation. This permits to easily justify the property called *(positive) independence of the computation rule*: the success branches of two SLD-trees for a goal $p(\overline{X_0^\varepsilon})$ are bijectively linked because each success branch is characterized by a finite skeleton t (and vice versa).

As a particular case of lemma 2 we have:

Theorem 11 Positive correctness.
Let RC *be a reject criterion and* I *be a* $\mathcal{D}$*-model of* $\text{IF}(P)$.
If $S \vdash p$ *is a computed positive sequent then* $S \vdash p$ *is valid in* I.

We can state a corollary as for corollary 3.

[12] A partial skeleton is a skeleton where on some nodes an undefined leaf takes the place of the subtree.

Let $\mathcal{Z}^+(p, \mathrm{RC}) = \{S \mid$ there exists t such that $t : S \vdash p \in \mathit{lfp}(\mathcal{T}_P), S \notin \mathrm{RC}\}$, that is $\{S \mid S \vdash p$ is a RC-computed positive sequent$\}$.

Theorem 12 Positive completeness.
Let RC *be a reject criterion correct wrt* $\mathcal{D}$ *and let* F *be a formula built over the constraint language.*
If $\mathrm{IF}(P) \models_{\mathcal{D}} F \rightarrow p(\overline{X_0^\varepsilon})$ *then the negative cover* $\exists_{-\overline{X_0^\varepsilon}} F \vdash \mathcal{Z}^+(p, \mathrm{RC})$ *is valid in* $\mathcal{D}$.

> **Proof.** From lemma 7 because the members of $\mathcal{Z}^+(p) \setminus \mathcal{Z}^+(p, \mathrm{RC})$ have no solution in $\mathcal{D}$. □

Similarly, we can state a version of corollary 8 with a reject criterion correct wrt $\mathcal{T}$.

5.2 Negative Computation

A *negative computation* corresponds to the computation of a SLD-tree.

In order to formalize infinite (*fair*) derivation, the reject criterion is extended to infinite stores in the following way: a store S is rejected if and only if there exists a finite part of S which is rejected. This is a form of compactness of the reject criterion to express that when a SLD-derivation is infinite the computation has never met a rejected store.

To each (finite or infinite) non failure fair SLD-derivation for the goal $p(\overline{X_0^\varepsilon})$ corresponds a member $t : S \vdash p$ of $\mathit{gfp}(\mathcal{T}_P)$ such that $S \notin \mathrm{RC}$. The skeleton t characterizes the clauses used by the derivation and S is the set of constraints accumulated by the derivation. This permits to justify a property that we call *negative independence of the fair computation rule*: the non failure branches of two fair SLD-trees[13] for a goal $p(\overline{X_0^\varepsilon})$ are bijectively linked because each branch is characterized by a skeleton t (and vice versa).

Let $\mathcal{Z}^-(p, \mathrm{RC}) = \{S \mid$ there exists t such that $t : S \vdash p \in \mathit{gfp}(\mathcal{T}_P), S \notin \mathrm{RC}\}$.

The negative sequent associated with p wrt RC is $p \vdash \mathcal{Z}^-(p, \mathrm{RC})$. The negative sequent is RC-*computed* when $\mathcal{Z}^-(p, \mathrm{RC})$ is a finite set of finite stores (i.e. when there exists a finite SLD-tree).

[13] Note that each finite SLD-tree is fair (independently of the computation rule).

Theorem 13 Negative correctness.
Let RC *be a reject criterion correct wrt* $\mathcal{D}$ *and* I *be a* $\mathcal{D}$*-model of* FI(P).
$p \vdash \mathcal{Z}^-(p, \mathrm{RC})$ *is valid in* I.

> **Proof.** From lemma 4 because the members of $\mathcal{Z}^-(p) \setminus \mathcal{Z}^-(p, \mathrm{RC})$ have no solution in $\mathcal{D}$. □

> *Remark.* Note that when there is finite failure for the goal $p(\overline{X_0^\varepsilon})$, we have $\mathcal{Z}^-(p, \mathrm{RC}) = \emptyset$ so $p \vdash \emptyset$, i.e. $\neg p(\overline{X_0^\varepsilon})$ is valid in I. ◇

Similarly, we can state a version of corollary 5 when RC is correct wrt $\mathcal{T}$.

Theorem 14 Negative completeness.
Let RC *be a reject criterion, let* $\mathcal{D}$ *be a pre-interpretation and let* F *be a formula built over the constraint language.*
If FI(P) $\models_{\mathcal{D}} p(\overline{X_0^\varepsilon}) \rightarrow F$ *then for each* $S \in \mathcal{Z}^-(p, \mathrm{RC})$*: the positive cover* $S \vdash \exists_{-\overline{X_0^\varepsilon}} F$ *is valid in* $\mathcal{D}$.

> **Proof.** From lemma 9 because $\mathcal{Z}^-(p, \mathrm{RC})$ is a subset of $\mathcal{Z}^-(p)$. □

Again, we can state a version of corollary 10 and next the well-known results of *completeness of negation by failure*:

Lemma 15 Completeness of negation by failure.
Let RC *be a reject criterion complete wrt a constraint theory* $\mathcal{T}$.
If $\mathcal{T}$, FI(P) $\models_{\mathcal{D}} \neg p(\overline{X_0^\varepsilon})$ *then each fair SLD-tree (according to* RC*) for the goal* $p(\overline{X_0^\varepsilon})$ *is a finite failure SLD-tree.*

> **Proof.** We have to show that $\mathcal{Z}^-(p, \mathrm{RC})$ is empty (i.e. there is no non failure branches in the fair SLD-trees). But for each $S \in \mathcal{Z}^-(p, \mathrm{RC})$: $S \vdash$ *false* from the corollary of theorem 14 and if $\mathcal{Z}^-(p, \mathrm{RC})$ contains a store S then S is not rejected and, because the reject criterion is complete wrt $\mathcal{T}$, we have a contradiction (because of the compactness of the first order logic). □

One would like to have a result of finite negative completeness (which extends the result of the completeness of negation by failure), for example: if $\mathcal{T}, \mathrm{FI}(P) \models p(\overline{X_0^\varepsilon}) \rightarrow F$ then there exists a finite SLD tree for the goal $p(\overline{X_0^\varepsilon})$ such that, for each $S \in \mathcal{Z}^-(p, \mathrm{RC})$, $\bigwedge_{c \in S} c \rightarrow \exists_{-\overline{X_0^\varepsilon}} F$ is valid. But we see the reason why there is no more general finite negative completeness. Let us assume $\mathcal{T}, \mathrm{FI}(P) \models p(\overline{X_0^\varepsilon}) \rightarrow F$. To show that there exists a finite SLD tree for the goal $p(\overline{X_0^\varepsilon})$ such that each success implies F amounts to show that $\mathcal{Z}^-(p, \mathrm{RC})$ is a finite set of finite stores. Even if RC is complete wrt $\mathcal{T}$ the single case where it is guaranteed is when F is *false*, i.e. $\mathcal{Z}^-(p, \mathrm{RC}) = \emptyset$.

6 Conclusion

We have sketched a reformulation of the semantics of CLP which is based on inductive definitions. By using only principles of induction and co-induction, the basic results are reconstructed in a natural and elegant way. For example, in this approach, we see directly where the hypotheses on the constraint solver (correctness, completeness) occur: in theorem 11 (positive correctness) and theorem 14 (negative completeness) we have no hypothesis on the reject criterion while in theorem 12 (positive completeness) and theorem 13 (negative correctness) it is assumed to be correct. Note also that the reject criterion is assumed to be complete in lemma 15 (completeness of negation by failure). Few real constraint solvers are completes, thus the interest of this result in the CLP framework is very limited. So we see the wide interest of research on other kinds of negation (for example constructive negation [5]).

In our formalism, it is possible to give an inductive definition which corresponds exactly to what is computed: we keep only the rules whose conclusions have a non rejected store. In that case, the least fixpoint of the associated operator corresponds to the $\mathcal{S}$-semantics of the program [1].

Some additional inductive notions can be used to formalise *finite negative* RC-*computation*: in [7, 6] an inductive definition is used as the basis for some notion of error (*non complete cover*) associated with a symptom of missing answer.

The theoretical framework sketched in this paper is an extension to Constraint Logic Programming of the Grammatical View of Logic Programming

[3]. The novelty is not only the parameterisation by a constraint domain and a reject criterion, it is also in the use of finite and infinite skeletons. For example this notion gives a straightforward characterisation of the non-failed (finite or infinite) branches of a fair SLD-tree (*negative independence of the fair computation rule*).

References

[1] Annalisa Bossi, Maurizio Gabrielli, Giorgio Levi, and Maurizio Martelli. The S-Semantics Approach: Theory and Applications. *Journal of Logic Programming*, 19&20:149–198, 1994.

[2] Patrick Cousot and Radhia Cousot. Inductive definitions, semantics and abstract interpretation. In *Conference Record of the 19$^{\text{th}}$ ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages*, pages 83–94. ACM Press, 1992.

[3] Pierre Deransart and Jan Małuszyński. *A Grammatical View of Logic Programming*. MIT Press, 1993.

[4] François Fages. *Programmation Logique par Contraintes*. ellipses, 1996.

[5] François Fages. Constructive negation by pruning. *Journal of Logic Programming*, 32(2):85–118, 1997.

[6] Gérard Ferrand and Alexandre Tessier. Clarification of the bases of Declarative Diagnosers for CLP. Deliverable D.WP2.1.M1.1-1, 1997. Debugging Systems for Constraint Programming (ESPRIT 22532).

[7] Gérard Ferrand and Alexandre Tessier. Positive and Negative Diagnosis for Constraint Logic Programs in terms of Proof Skeletons. In Mariam Kamkar, editor, *International Workshop on Automated Debugging*, pages 141–154, 1997.

[8] Joxan Jaffar and Jean-Louis Lassez. Constraint Logic Programming. In 14^{th} *ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[9] Joxan Jaffar and Michael J. Maher. Constraint Logic Programming: a survey. *Journal of Logic Programming*, 19-20:503–581, 1994.

[10] Giorgio Levi and Catuscia Palamidessi. Contributions to the Semantics of Logic Perpetual Processes. *Acta Informatica*, 25:691–711, 1988.

[11] Michael J. Maher. Logic Semantics for a class of Committed-Choice Programs. In *International Conference on Logic Programming*, pages 858–876, 1987.

[12] Michael J. Maher. A Logic Programming view of CLP. In Warren, editor, *International Conference on Logic Programming*, pages 737–753. MIT Press, 1993.

[13] Robin Milner and Mads Tofte. Co-induction in relational semantics. *Theoritical Computer Science*, 87:209–220, 1991.

[14] Mohand-Areski Nait Abdallah. Fair derivations in logic programming : operational and greatest fixpoint semantics. *Fundamenta Informaticae*, X(3):247–308, 1987.

[15] Lawrence C. Paulson and Andrew W. Smith. Logic Programming, Functional Programming, and Inductive Definitions. In *Extensions of Logic Programming*, volume 475 of *Lecture Notes in Computer Science*, pages 283–309. Springer-Verlag, 1989.

[16] Ehud Y. Shapiro. *Algorithmic Program Debugging.* ACM Distinguished Dissertation. MIT Press, 1982.

[17] Alexandre Tessier. *Approche, en termes de squelettes de preuve, de la sémantique et du diagnostic d'erreur des programmes logiques avec contraintes.* PhD thesis, LIFO, University of Orléans, 1996.

[18] Alexandre Tessier. Declarative Debugging in Constraint Logic Programming. In Joxan Jaffar, editor, *Asian Computing Science Conference*, volume 1179 of *Lecture Notes in Computer Science*, pages 64–73. Springer-Verlag, 1996.

[19] Alexandre Tessier. Une caractérisation des arbres SLD en programmation logique avec contraintes. In Philippe Devienne, editor, *actes du pôle Contraintes et Programmation Logique*, pages 1–11. Journées du GDR Programmation du CNRS, 1996.

This research was supported in part by LOCO Project, common to University of Orléans and INRIA Rocquencourt.