A Hierarchy of Semantics for Normal Constraint Logic Programs

François FAGES[1] and Roberta GORI[2]

[1] LIENS CNRS, Ecole Normale Supérieure, 45 rue d'Ulm, 75005 Paris, France,
fages@dmi.ens.fr
[2] Dip. di Informatica, Universita di Pisa, Corso Italia 40, 56125 Pisa, Italy,
gori@di.unipi.it

Abstract. The different properties characterizing the operational behavior of logic programs can be organized in a hierarchy of fixpoint semantics related by Galois insertions, having the least Herbrand model as most abstract semantics, and the SLD operational semantics as most concrete semantics. The choice of a semantics in the hierarchy allows to model precisely the program properties of interest while getting rid of useless details of too concrete semantics, which is crucial for the development of efficient program analysis tools.
The aim of this paper is to push forward these methods by making them apply to normal (constraint) logic programs, that is full first-order (non Horn) programs. The fixpoint semantics defined by the first author for the rule of constructive negation by pruning is at the center of the hierarchy developed in this paper. We show that that semantics can be obtained by *concretization* of Kunen's semantics defined as a fixpoint, taken as the most abstract semantics of the hierarchy, and that by further concretization it leads to a new operational semantics for normal CLP programs. The different observable properties of the program, such as successful derivations, finite failure, set of computed answer constraints, etc. are modeled by precise semantics in the hierarchy.

1 Introduction

One of the striking features of constraint logic programming (CLP) languages is the existence of abstract model-theoretic semantics that allow powerful forms of reasoning about CLP programs at a high level of abstraction. Some program properties of interest may necessitate however the definition of appropriate more concrete semantics allowing for semantic-based data-flow analysis, error diagnosis, abstract interpretation. While the concrete operational semantics should contain all the relevant details, being the most concrete semantics of the program, there is still a need to use more abstract semantics in order to get rid of useless details. The *collecting semantics* approach was originally intended to give a solution to this problem, and the theory of abstract interpretation [2] [3] provides order-theoretic tools to model abstraction operators. In logic programming, we shall impose also that a collecting semantics includes the standard least

Herbrand model. This gives rise to a hierarchy of fixpoint semantics related by Galois insertions, having the least Herbrand model as most abstract semantics, and the SLD operational semantics as most concrete semantics. This approach has been progressively undertaken for the semantics of definite (constraint) logic programs [18, 1, 10] and is now well understood [5, 12, 11]. In particular the main observable properties of a program can be modeled by both operational (top-down) and denotational (bottom-up) abstract semantics in the hierarchy.

The aim of this paper is to push forward these methods by making them apply to *normal* CLP programs, that is full first-order (non Horn) CLP. We shall limit ourselves in this paper to the necessary basic contributions. The lifting of the results obtained for definite CLP programs to normal CLP programs raises indeed a lot of difficulties.

First, the abstract logical semantics of normal CLP programs has been controversial for a long time. One can say that it has now stabilized on Kunen's three-valued logical consequences of the program's completion[1] [16]. Kunen's semantics enjoys the main desirable properties, in particular it is recursively enumerable, general (i.e. defined for the whole class of normal CLP programs), compatible with negation as failure [17], and faithful to constructive negation [20]. Kunen's semantics is not defined as a fixpoint however. It can be characterized by the finite powers of Fitting's operator whose least fixpoint gives a different, highly non effective semantics [9].

Second, the negation as failure rule is a sound but incomplete inference rule for normal CLP programs. The constructive negation by pruning rule introduced by Chan [4] provides normal CLP programs with a complete operational semantics w.r.t. Kunen's semantics [20]. However that scheme has not received yet a fully abstract fixpoint semantics characterizing the computed answer constraints for instance. In [8] such a fixpoint semantics has been given for a variant of the constructive negation rule, called constructive negation by pruning. That semantics generalizes the S-semantics of definite logic programs [1] to normal CLP programs.

In this paper we show that the different properties characterizing the operational behavior of a normal CLP program can be organized in a *hierarchy* of fixpoint semantics related by Galois insertions, corresponding to a hierarchy of observable properties and program equivalences (see figure **??**). The S-semantics and the fixpoint operator T_P^S defined in [8] are at the center of the hierarchy developed in this paper.

In section 3 we define the most abstract Kunen's semantics, as a fixpoint semantics and define a collecting semantics as being a pair $(\mathcal{C}, T)$ where $\mathcal{C}$ is a semantic domain, and T a monotone operator on $\mathcal{C}$. The other semantics of the hierarchy will be obtained by a concretization.

[1] Note that in absence of function symbols, as it is the case in deductive databases, a different choice can be made. Van Gelder's well-founded semantics is often viewed as the standard semantics in such a context. Nevertheless in presence of function symbols the well-founded semantics is highly non effective and is thus inappropriate.

Section 4 defines the s-collecting semantics based on operator T_P^S. It is related to the top of the hierarchy through an analogue to Clark's semantics defined in section 5.

In section 6 we obtain by a pure concretization of the s-collecting semantics, a new operational semantics for normal CLP programs, based on a rewriting system on T_P^S-trees. We take it as the most concrete operational semantics in the hierarchy.

In this approach the operational semantics based on the constructive negation by pruning rule [8] is not a collecting semantics w.r.t. the s-collecting semantics and is thus not in this hierarchy. The reason is that, despite the full-abstraction theorem of the S-semantics w.r.t. constructive negation by pruning [8], one step of application of operator T_P^S doesn't correspond to one derivation step under the constructive negation by pruning rule. That operational semantics thus cannot be obtained as a systematic concretization process, a different operational semantics more faithful to the operator T_P^S is obtained instead.

The results of this paper show that in this approach the operational behavior of normal CLP programs can be described at different levels of abstraction inside a hierarchy of semantics similarly to the Horn fragment. The hierarchy can be used to systematically design appropriate or optimal semantics w.r.t. the observable properties of interest. The S-semantics characterizes exactly those programs having the same set of computed answer constraints (c.a.c.). The Clark's semantics characterizes those programs having the same set of queries with *true* as c.a.c.. The Kunen's semantics characterizes those programs having the same set of successful queries.

The main theoretical tools used in this paper are on the one hand the recourse to recursively saturated models for the abstract semantics, and on the other hand the definition of several continuous finitary versions of Fitting's operator.

2 Preliminaries

2.1 The language of constraints

The first-order language of constraints is defined on a countable infinite set of variables V and on a signature Σ composed of a set of predicate symbols containing *true* and $=$, and of sets of n-place function symbols for each arity n (constants are functions with arity 0). A *primitive constraint* is an atomic proposition of the form $p(t_1, ..., t_n)$, where p is a predicate symbol in Σ and the t_i's are Σ, V-terms. A *constraint* is a well-formed first-order Σ, V-formula. The set of free variables in an expression e is denoted by $V(e)$. Sets of variables will be denoted by $X, Y, ...$. For a constraint c, we shall use the notation $\exists c$ (resp. $\forall c$) to represent the closed constraint $\exists X\ c$ (resp. $\forall X\ c$) where $X = V(c)$.

The intended interpretation of constraints is defined by fixing a Σ-structure $\mathcal{A}$. An $\mathcal{A}$-valuation for a Σ, V-expression is a mapping $\theta : V \to \mathcal{A}$ which extends by morphism to terms and primitive constraints. Logical connectives and quantifiers are interpreted as usual, a constraint c is $\mathcal{A}$-solvable iff $\mathcal{A} \models \exists c$.

The only property that we require on $\mathcal{A}$ is that constraints are decidable in $\mathcal{A}$, so that $\mathcal{A}$ can be presented by a decidable first-order theory $th(\mathcal{A})$, i.e. satisfying:

1. (soundness) $\mathcal{A} \models th(\mathcal{A})$,
2. (satisfaction completeness) either $th(\mathcal{A}) \models \exists c$ or $th(\mathcal{A}) \models \neg\exists c$, for any constraint c.

As a constraint is any Σ, V-formula, these conditions are equivalent to say that $th(\mathcal{A})$ is a complete first-order theory, and thus that all models of $th(\mathcal{A})$ are elementary equivalent. For example, Clark's equational theory CET (augmented with the domain closure axiom DCA if the signature is finite) provides such a complete decidable theory for the Herbrand universe with first-order equality constraints [16].

2.2 CLP(A) programs

$CLP(\mathcal{A})$ programs are defined using an extra finite set of predicate symbols Π disjoint from Σ. An *atom* has the form $p(t_1, ..., t_n)$ where $p \in \Pi$ and the t_i's are Σ, V-terms. A *literal* is either an atom (positive literal) or a negated atom $\neg A$ (negative literal).

A *definite (resp. normal) $CLP(\mathcal{A})$ program* is a finite set of clauses of the form $A \leftarrow c|L_1, ..., L_n$ where $n \geq 0$, A is an atom, called the head, c is a constraint, and $L_1, ..., L_n$ are atoms (resp. literals). The *local variables* of a program clause is the set of free variables in the clause which do not occur in the head.

In order to characterize precise operational aspects of CLP programs, such as sets of computed anser constraints, the formal semantics of $CLP(\mathcal{A})$ programs will be defined by sets of constrained atoms. A *constrained atom* is a couple $c|A$ where c is an $\mathcal{A}$-solvable constraint such that $V(c) \subseteq V(A)$. The set of constrained atoms is denoted by $\mathcal{B}$. A *constrained interpretation* is a subset of $\mathcal{B}$. The set of *ground instances* of a constrained atom over $\mathcal{A}$ is defined by:

$$[c|A]_{\mathcal{A}} = \{A\theta \mid \theta : V \to \mathcal{A},\ \mathcal{A} \models c\theta\}$$

We denote also by $[I]_{\mathcal{A}}$ the set of ground instances of a constrained interpretation I. The set of ground atoms is denoted by $\mathcal{B}_{\mathcal{A}}$. A ground atom $A\theta$ is true (resp. false) in I if $A\theta \in [I]_{\mathcal{A}}$ (resp. $A\theta \notin [I]_{\mathcal{A}}$).

The logical semantics of *normal* $CLP(\mathcal{A})$ programs is defined via the Clark's completion of the program, that is the the conjunction of $th(\mathcal{A})$ with a formula P^* obtained from P by putting in a conjunction the following formulae:

$$\forall X\ p(X) \leftrightarrow \bigvee_{i=1}^{n} \exists Y_i\ c_i \wedge \alpha_i$$

for each predicate symbol p defined in P by a set of clauses $\{p(X) \leftarrow c_i|\alpha_i\}_{1 \leq i \leq n}$, where $Y_i = V(c_i|\alpha_i) \setminus X$, and the formula $\forall X\ \neg p(X)$ for the other predicate symbols which don't appear in any head in P.

The completion of a normal program can be inconsistent, e.g. with the program $P = \{p \to \neg p\}$, $P^* = (p \leftrightarrow \neg p)$, in that case any constraint should be a correct answer constraint for any goal. In order to define a faithful logical semantics for normal programs, the solution proposed by Kunen is to consider the *3-valued logical consequences* of $P^*, th(\mathcal{A})$. The usual strong 3-valued interpretations of the connectives and quantifiers are assumed, except for the connective $a \leftrightarrow b$ which is interpreted as t if a and b have the same truth value (f, t or u), and f otherwise (i.e. Lukasiewicz's 2-valued interpretation of $\leftrightarrow$).

Intuitively, the semantics of a normal CLP program is the pair of true and false consequences similarly to a three-valued model. The formal semantics of normal $CLP(\mathcal{A})$ programs will be thus defined by partial interpretations. A *partial constrained interpretation* for a $CLP(\mathcal{A})$ program is a couple of sets of constrained atoms, $I =< I^+, I^- >$, satisfying the following consistency condition: $[I^+]_{\mathcal{A}} \cap [I^-]_{\mathcal{A}} = \emptyset$. The set of partial interpretations forms a semi-lattice for set inclusion on true and false constrained atoms, we denote it by $(\mathcal{I}, \subseteq_3)$. It is not a lattice as the union of two partial interpretations may not be a partial interpretation due to the consistency condition.

The (Kunen's) *logical semantics* of a normal $CLP(\mathcal{A})$ program P can be defined by the following partial interpretation:
$\mathcal{L}(P) =< \mathcal{L}^+(P), \mathcal{L}^-(P) >$ where
$\mathcal{L}^+(P) = \{c|p(X) \in \mathcal{B} \ : \ P^*, th(\mathcal{A}) \models_3 c \to p(X)\}$,
$\mathcal{L}^-(P) = \{c|p(X) \in \mathcal{B} \ : \ P^*, th(\mathcal{A}) \models_3 c \to \neg p(X)\}$.

2.3 Other preliminary notions

In the following, we shall also assume familiarity with the standard notion of *abstract interpretation* [2].

Given two posets $(\mathcal{D}_a, \leq_a)$ and $(\mathcal{D}_c, \leq_c)$, A *Galois insertion* is a quadruple $(\mathcal{C}_a, \alpha, \mathcal{C}_c, \gamma)$ where $(\mathcal{C}_a, \leq_a)$ and $(\mathcal{C}_c, \leq_c)$ are posets, and $\alpha : \mathcal{C}_c \to \mathcal{C}_a$, $\gamma : \mathcal{C}_c \to \mathcal{C}_a$ are maps such that:

- α and γ are monotone.
- $\forall\, x, \ \ x \in \mathcal{C}_\dashv \quad \alpha \cdot \gamma(x) = x$
- $\forall\, y \ \ y \in \mathcal{C}_\rfloor \quad \gamma \cdot \alpha(y) \geq_c y$

Let A be a set, the powerset of of A is denoted by $\mathcal{P}(A)$. A equipped with a partial order $\sqsubseteq$ is denoted by $A_\sqsubseteq$. We write $f : A \to B$ to mean that f is a total function of A into B.
The set of fixpoints of a function f is denoted by $fp(f)$, and the least fixpoint, if it exists, is denoted by $lfp(f)$.

3 Kunen's semantics and collecting semantics

A *semantics definition* for a normal CLP program is a pair $< \mathcal{D}, T >$, where the domain $\mathcal{D} = \mathcal{D}^+ \times \mathcal{D}^-$ forms a semi-lattice $\mathcal{D}_\sqsubseteq$ (the semantics domain) and

$T : \ Program \times \mathcal{D} \rightarrow \mathcal{D}$ is such that given a program P, T is a monotone operator on $\mathcal{D}$.

For technical reasons T is not supposed to be continuous, however the semantics will be identified to the least fixpoint of T which will be always reached at ordinal omega in the following.

We consider as reference semantics for normal $CLP(\mathcal{A})$ programs, Kunen's semantics, i.e. the set of formulas ϕ, such that ϕ is a three-valued logical consequences of the program completion P^* and $Th(\mathcal{A})$. Kunen's semantics can be characterized by the *finite* powers of Fitting's operator over the algebra $\mathcal{A}$ [16], but this is not a fixpoint in general, as the closure ordinal of Fitting's operator can be far beyond ordinal omega, in fact it can be larger than all recursive ordinals [9]. We recall the definition of the Fitting's operator:

Definition 1. *Let P be a $CLP(\mathcal{A})$ program. The immediate consequence operator $\Phi_P^{\mathcal{A}} : \mathcal{P}(\mathcal{B}_{\mathcal{A}}) \rightarrow \mathcal{P}(\mathcal{B}_{\mathcal{A}})$ is defined by:*

$$\begin{array}{ll} \Phi_P^{\mathcal{A}^+}(I) = \{\ A \in \mathcal{B}_{\mathcal{A}} : & \textit{there exist a clause in } P, p(X) \leftarrow \alpha, \textit{ and a valuation } \vartheta \\ & \textit{such that } A = p(X)\vartheta \textit{ and } I(\alpha\vartheta) = true\} \\ \Phi_P^{\mathcal{A}^-}(I) = \{\ A \in \mathcal{B}_{\mathcal{A}} : & \textit{for any clause in } P, p(X) \leftarrow \alpha \textit{ and any valuation } \vartheta \\ & \textit{such that } A = p(X)\vartheta \textit{ then } I(\alpha\vartheta) = false\} \end{array}$$

Next theorem gives a characterization of Kunen's semantics in terms of the finite powers of Fitting's operator.

Theorem 1 ([16][20]). *Let P be a normal $CLP(\mathcal{A})$ program and ϕ be a Π, Σ, V-formula, then $P^*, th(\mathcal{A}) \models_3 \phi$ iff ϕ is true in $\Phi_P^{\mathcal{A}} \uparrow n$ for some integer n.*

In [8] Kunen's semantics is characterized as the least fixed point of a non-ground immediate consequence operator. It is also possible however to give a different fix point characterization based on a ground operator. We make this choice here as it allows to define a canonical (three-valued) model in the usual sense for the program.

Such a fixpoint characterization has been given in [6] by considering the Fitting's operator defined on a particular structure that is a *saturated* model of the constraint language.

Definition 2. *Let $\Phi = (\phi^i |\ i \in \mathcal{N})$ be a sequence of formulas ϕ^i in finitely many free variables $x_1, \ldots, x_k, y_1, \ldots, y_m$ and let $\mathcal{J}$ be a model.*
$\mathcal{J}$ is called Φ-saturated if for all assignments $\tau : \{y_1, \ldots, y_m\} \rightarrow \mathcal{J}$, either some $\sigma \geq \tau$ exists such that for all i, $\mathcal{J} \models \phi^i\sigma$, or there exists $N \in \mathcal{N}$ such that for no $\sigma \geq \tau$, $\mathcal{J} \models \bigwedge_{i<N} \phi^i\sigma$.
$\mathcal{J}$ is saturated if it is Φ-saturated for every sequence Φ. $\mathcal{J}$ is recursively saturated if it is Φ-saturated for every computable sequence Φ.

It is well-known that every model in first-order logic has a saturated elementary extension. The following results due to Doets provide the fixpoint characterization of the Kunen's semantics.

Lemma 1 ([6]). *If a $Th(\mathcal{A})$-algebra $\mathcal{J}$ is recursively saturated then, for every $\mathcal{J}$-sentence ϕ (not containing $\leftrightarrow$): if ϕ is true (resp. false) in $\Phi_P^{\mathcal{J}} \uparrow \omega$, then for some $n \in \mathcal{N}$, ϕ is true (resp. false) in $\Phi_P^{\mathcal{J}} \uparrow n$.*

Theorem 2 ([6]). *If a $Th(\mathcal{A})$-algebra $\mathcal{J}$ is recursively saturated then, for every program P:*
$\Phi_P^{\mathcal{J}} \uparrow \omega = lfp(\Phi_P^{\mathcal{J}})$

Now we are ready to define our *core* semantics:

Definition 3. *Let P be a $CLP(\mathcal{A})$ normal program, let $\mathcal{J}$ be a saturated model of $Th(\mathcal{A})$ the three-valued logical* core semantics *of P (let us call it $TLC(P^*, Th(\mathcal{A}))$) is the pair $< \mathcal{D}_G, \Phi_P^{\mathcal{J}} >$ where $\mathcal{D}_G = \mathcal{P}([\mathcal{B}_{\mathcal{J}}) \times \mathcal{P}(\mathcal{B}_{\mathcal{J}})$ and $\Phi_P^{\mathcal{J}}$ is the Fitting's operator on the algebra $\mathcal{J}$.*

A *collecting semantics* is intended to provide a precise (concrete) description of those program properties of interest. As [12] we shall use the standard framework of abstract interpretation [2] in terms of *Galois insertions*, and shall obtain the other semantics of the hierarchy by a pure concretization process.

Definition 4. *A collecting semantics (with respect to $TLC(P^*, Th(\mathcal{A}))$) is a semantics $< \mathcal{C}, T >$ such that there exists α and γ where: $(\mathcal{C}, \alpha, \mathcal{D}_G, \gamma)$ is a Galois insertion and for any $P \in Program$: $\Phi_P^{\mathcal{J}} = \alpha \cdot T \cdot \gamma$.*

Then the core semantics is an abstract interpretation of any collecting semantics. In the following $< \mathcal{C}, T > \underset{\alpha}{\overset{\gamma}{\rightleftharpoons}} < \mathcal{C}', T' >$ will denote a Galois insertion $(\mathcal{C}, \alpha, \mathcal{C}', \gamma)$ such that $T = \alpha \cdot T' \cdot \gamma$, i.e., $< \mathcal{C}', T' >$ is a collecting semantics with respect to $< \mathcal{C}, T >$. When it will be not specified otherwise a collecting semantics will be always with respect to $TLC(P^*, Th(\mathcal{A}))$. In this case, when the abstraction α and the concretization γ are given, a collecting semantics will be simply denoted $< \mathcal{C}, T >_\alpha^\gamma$.

Proposition 1. *Let $< \mathcal{C}, T >_\alpha^\gamma$ be a collecting semantics and let $< \mathcal{C}', T' >$ such that*
$< C, T > \underset{\alpha'}{\overset{\gamma'}{\rightleftharpoons}} < C', T' >$. Then $< \mathcal{C}', T' >_{\alpha \cdot \alpha'}^{\gamma' \cdot \gamma}$ is a collecting semantics.

4 S-semantics for normal CLP programs

The aim of the S-semantics is to provide a collecting semantics capable to characterize the equivalence of normal $CLP(\mathcal{A})$ programs based on the observation of the set of computed answer constraints.

In this section we consider the fixpoint semantics introduced in [8] in connection to the operational principle of constructive negation by pruning. That semantics generalizes to normal programs the S-semantics given in [10] for definite logic programs, but it is worth noting that when restricted to definite programs that semantics also models finite failures an observable that was out of the scope of the original S-semantics approach. We shall show in the following that the S-semantics is a collecting semantics in the hierarchy. We recall here the definition of the fix point operator.

The S-semantics of a normal CLP program will be defined as a partial constrained interpretation, i.e. a subset of the non-ground base $\mathcal{B}$. The fixpoint operator T_P^S has thus to be a non-ground version of Fitting's operator $\Phi_P^{\mathcal{A}}$. The non-ground version defined in [20] is not convenient as its closure ordinal can be far beyond ordinal ω, just as Fitting's operator over a non-saturated model.

The idea underlying the definition of the operator T_P^S in [8] is to obtain a *continuous* non-ground operator, simply by taking a *finitary* version of Fitting's operator[2]. Thus in this approach, a constrained atom (resp. literal) is true in $T_P^{S^+}(I)$ (resp. $T_P^{S^-}(I)$) if it is an immediate consequence of the program clauses and of a *finite* part of the constrained interpretation I. This leads to the following:

Definition 5 ([8]). *Let P be a $CLP(\mathcal{A})$ program. T_P^S is an operator over $\mathcal{P}(\mathcal{B}) \times \mathcal{P}(\mathcal{B})$ defined by*

$$\begin{array}{ll}
T_P^{S^+}(I) = \{c|p(X) \in \mathcal{B} : & \textit{there exist a clause in } P \textit{ with local variables } Y, \\
& p(X) \leftarrow d|A_1, ..., A_m, \neg A_{m+1}, ..., \neg A_n. \\
& c_1|A_1, ..., c_m|A_m \in I^+,\ c_{m+1}|A_{m+1}, ..., c_n|A_n \in I^- \\
& \textit{such that } c = \exists Y\ d \wedge \bigwedge_{i=1}^n c_i \textit{ is } \mathcal{A}\textit{-satisfiable}\} \\
T_P^{S^-}(I) = \{c|p(X) \in \mathcal{B} : & \textit{for each clause defining } p \textit{ in } P \textit{ with local variables } Y_k, \\
& p(X) \leftarrow d_k|A_{k,1}, ..., A_{k,m_k}, \alpha_k. \\
& \textit{there exist } e_{k,1}|A_{k,1}, ..., e_{k,m_k}|A_{k,m_k} \in I^-, \\
& e_{k,m_k+1}|A_{k,m_k+1}, ..., e_{k,n_k}|A_{k,n_k} \in I^+,\ n_k \geq m_k \\
& \textit{where for } m_k + 1 \leq j \leq n_k,\ \neg A_{k,j} \textit{ occurs in } \alpha_k, \\
& \textit{such that } c = \bigwedge_k \forall Y_k(\neg d_k \vee \bigvee_{i=1}^{n_k} e_{k,i}) \textit{ is } \mathcal{A}\textit{-satisfiable}\}.
\end{array}$$

Being monotone and finitary, T_P^S is continuous in the semi-lattice $(S, \subseteq_3)$.

Definition 6. *The* S-semantics *is the pair $< \mathcal{S}, T_P^S >$, where $\mathcal{S} = \mathcal{P}(\mathcal{B}) \times \mathcal{P}(\mathcal{B})$.*

We refer to [8] for a complete study of this operator and full abstraction results w.r.t. constructive negation by pruning. In the sequel we show that the S-semantics is indeed a collecting semantics in the hierarchy and we derive from it a new operational semantics obtained by a pure concretization process.

5 Clark's semantics for normal CLP programs

The generalization of Clark's semantics to normal CLP programs can now be defined as a simple upward closure of the S-semantics. We first define the Up operator over $\mathcal{P}(\mathcal{B}) \times \mathcal{P}(\mathcal{B})$:

[2] An operator f over a powerset is finitary if $\forall x, y\ x \in f(y) \Rightarrow \exists y' \subseteq y$ finite s.t. $x \in f(y')$. One can easily check that a finitary operator which is monotonic w.r.t. set inclusion, is necessarily continuous.

Definition 7. *The set of* ground instances *of a constrained atom over* $\mathcal{A}$ *is defined by :*

$$[c|A]_{\mathcal{A}} = \{A\vartheta \mid \vartheta : V \to \mathcal{A}, \mathcal{A} \models c\vartheta\}$$

Definition 8. *Let* I *be a subset of* $\mathcal{B}$

$Up(I) = \{d|A \ : \ \textit{there exists } c|A \in I \textit{ s.t. } th(\mathcal{A}) \models d \to c \textit{ and } d \textit{ is } \mathcal{A}\textit{-satisfiable}\}$

A partial Clark interpretation for a $CLP(\mathcal{A})$ program is a couple of upward closed sets of constrained atoms $U =< U^+, U^- >$, satisfying the following consistency condition:

$$[U^+]_{\mathcal{A}} \cap [U^-]_{\mathcal{A}} = \emptyset$$

The set of partial Clark interpretation forms a semi-lattice for set inclusion, we denote it by $(U, \subseteq_3)$.

Definition 9. *The* Clark's semantics *of a normal* $CLP(\mathcal{A})$ *program* P *is the pair*
$< \mathcal{U}, T_P^U >$, *where* $\mathcal{U} = Up(\mathcal{P}(\mathcal{B})) \times Up(\mathcal{P}(\mathcal{B}))$ *and* $T_P^U = Up \circ T_P^S$.

We want to show that the Clark's semantics is a collecting semantics. First we prove that there exist α and γ which provide a Galois insertion between the domain of the semantics we are considering.
It is worth noting that since our interpretations I are pairs of interpretations $< I^+, I^- >$, then also the abstraction operator α has to be considered as a pair $< \alpha^+, \alpha^- >$ and in the same way $\gamma =< \gamma^+, \gamma^- >$.

We recall that we note by $\mathcal{J}$ the saturated model of $Th(\mathcal{A})$ considered in the core semantics. Now by a cautious proof process we can establish:

Lemma 2 ([13]). $< \alpha_g, \gamma_g >$, *where* $\alpha_g =< \alpha_g^+, \alpha_g^- >$,
$\alpha_g^+ = \lambda I^+ \in \mathcal{P}(\mathcal{B}).[I^+]_{\mathcal{J}}$, $\alpha_g^- = \lambda I^- \in \mathcal{P}(\mathcal{B}).[I^-]_{\mathcal{J}}$,
$\gamma_g =< \gamma_g^+, \gamma_g^- >$,
$\gamma_g^+ = \lambda I^+ \in \mathcal{P}(\mathcal{B}_{\mathcal{J}}).\bigcup\Big\{ \rfloor|\mathcal{A} \ : \ [\rfloor|\mathcal{A}]_{\mathcal{J}} \in \mathcal{I}^+ \Big\}$, $\gamma_g^- = \lambda I^- \in \mathcal{P}(\mathcal{B}_{\mathcal{J}}).\bigcup\Big\{ \rfloor|\mathcal{A} \ : \ [\rfloor|\mathcal{A}]_{\mathcal{J}} \in \mathcal{I}^- \Big\}$,
is a Galois insertion of $\mathcal{U}$ *(and* $\mathcal{S}$*) into* $\mathcal{D}$.

Theorem 3 ([13]). *For every* $\mathcal{J}$*-interpretation* I, $\Phi_P^{\mathcal{J}}(I) = \alpha_g(T_P^U(\gamma_g(I))) = \alpha_g(T_P^S(\gamma_g(I)))$.

Corollary 1. $< \mathcal{U}, T_P^U >$ *and* $< \mathcal{S}, T_P^S >$ *are collecting semantics.*

It is also immediate from the definition of the Clark's semantics that the S-semantics is a collecting semantics w.r.t. the Clark's semantics, and that in fact $T_P^C \circ Up = Up \circ T_P^S$, a property stronger than the one requested for being a collecting semantics.

6 Trees operational semantics

6.1 Bottom-up semantics

The operational semantics obtained by a direct concretization of operator T_P^S leads to the definition of a domain $\mathcal{T}$ of finite AND-OR-trees labeled by goals.

AND-nodes are labeled by constrained literals, $c|A$ or $c|\neg A$, where c is the conjunction of the constraints on its childrens. OR-nodes are labeled by complex goals of the form $c|\neg(d|\alpha)$, where c is the disjunction of the constraints on its childrens. The leaves are AND-nodes or OR nodes with zero successors, or special nodes labeled with a constraint, noted $c|\square$. The root will be always an AND-node.

With $\underset{\triangle_i}{c|G}$ we indicate a tree with root labeled by $c|G$. *Root* is a function: $\mathcal{T} \rightarrow \mathcal{B}$ which gives the label of the root of a tree, $Root(\underset{\triangle}{c|L}) = c|L$.

With $\begin{array}{c} c|L \\ \bigwedge_{and} \\ T_1, \ldots, T_n \end{array}$ we denote the tree with root $c|L$, and with the T_i's are subtrees in an AND-relation. With $\begin{array}{c} c|G \\ \bigwedge_{or} \\ T_1, \ldots, T_n \end{array}$ we denote the tree with root $c|G$, and with the T_i's as subtrees in an OR-relation.

A *partial tree interpretation* is a couple of sets of trees $=< I^+, I^- >$ satisfying the consistency property on the roots of the trees in I^+ and I^-. Now the concretization of the T_P^S operator leads to the definition of an operator $T_P^{Trees} =< T_P^{Trees^+}, T_P^{Trees^-} >$ over the domain of partial tree interpretations.

The first component specifies the AND-compositional execution tree for a positive literal A. This is obtained by composing with an AND-node the successful tree for the positive atoms in the body and the successful tree for the negative atoms in the body.
$T_P^{Trees^+}(I)$ is defined as the set of trees of the form

$$\left(\begin{array}{cccccccc} & & & c|A & & & & \\ & & & \bigwedge_{and} & & & & \\ d|\square & c_1|A_1 & \ldots & c_m|A_m & c_{m+1}|\neg A_{m+1} & \ldots & c_n|\neg A_n \\ & \triangle_1 & \ldots & \triangle_n & \triangle_{n+1} & \ldots & \triangle_m \end{array} \right)$$

such that $A : -d|A_1, \ldots, A_m, \neg A_{m+1}, \ldots, \neg A_n$, is a program clause,

$$\begin{array}{ccc} c_1|A_1 & \ldots & c_n|A_n \\ \triangle_1 & \ldots & \triangle_n \end{array} \in I^+ \qquad \begin{array}{ccc} c_{m+1}|\neg A_{m+1} & \ldots & c_n|\neg A_n \\ \triangle_{n+1}, & \ldots, & \triangle_{n+m} \end{array} \in I^-$$

and $c = d \wedge \bigwedge_{i=1}^n c_i$ is $\mathcal{A}$-satisfiable.

The definition of $T_P^{Trees^-}$ follows similarly the definition of $T_P^{S^-}$. $T_P^{Trees^-}(I)$ is defined as the set of trees of the form

$$\left(\begin{array}{cccccc}
 & & & & c|\neg A & \\
 & & & & \bigwedge_{and} & \\
 & & c_1|\neg(d_1|A_{1,1},...,A_{1,m_1},\alpha_1) & & & \dots \\
 & & \bigwedge_{or} & & & \dots \\
\neg d_1|\Box & e_{1,1}|\neg A_{1,1} \dots & e_{1,m_1}|\neg A_{1,m_k} & e_{1,m_{k+1}}|A_{1,m_k+1} \dots & e_{1,n_1}|A_{1,n_1} & \dots \\
 & \triangle_{1,1} & \triangle_{1,m_1} & \triangle_{1,m_1+1} & \triangle_{1,n_1} & \dots \triangle_{f_{k,n_k}}
\end{array}\right)$$

where $\{A \leftarrow d_1|A_{1,1},...,A_{1,m_1},\alpha_1,\ ...,\ A \leftarrow d_k|A_{k,1},...,A_{k,m_k},\alpha_k\}$ is the set of program clauses defining A,
the $A_{i,j}$'s, for $m_i + 1 \leq j \leq n_i$, are atoms such that $\neg A_{i,j}$ occurs in α_i,

for all clause index i, $1 \leq i \leq k$,

$$\begin{array}{c} e_{i,1}|\neg A_{i,1} \ \dots \ e_{i,m_i}|\neg A_{i,m_i} \\ \triangle_{1,1} \quad \dots \quad \triangle_{1,m_1} \end{array} \in I^- \qquad \begin{array}{c} e_{i,m_i+1}|A_{i,m_i+1} \ \dots \ e_{i,n_i}|A_{i,n_i} \\ \triangle_{1,m_1+1}, \quad \dots, \quad \triangle_{1,n_1} \end{array} \in I^+$$

$c_i = \neg d_i \vee \bigvee_{j=1}^{n_i} e_{i,j}$ and $c = \bigwedge_{i=1}^{k} c_i$ is $\mathcal{A}$-satisfiable.

That semantics will be the most concrete semantics definition of the hierarchy.

Definition 10. *The* T_P-Trees semantics *is the pair* $< \mathcal{P}(\mathcal{T}) \times \mathcal{P}(\mathcal{T}), T_P^{Trees} >$

Being monotone and finitary one can easily check that T_P^{Trees} is in fact a continuous operator over $\mathcal{P}(\mathcal{T}) \times \mathcal{P}(\mathcal{T})$. The T_P-Trees semantics is a concrete semantics which allows to distinguish programs having different T_P-Trees derivations, both for positive or negative literals. Now we can relate the T_P-Trees semantics to the S-semantics.

Lemma 3. *The pair* $< \alpha, \gamma >$*, where* $\alpha = < \alpha^+, \alpha^- >$*,*
$\alpha^+ = \lambda I^+ \in \mathcal{P}(\mathcal{T}).\left\{ c|A \ : \ \begin{array}{c} c|A \\ \triangle \end{array} \in I^+ \right\}$,
$\alpha^- = \lambda I^- \in \mathcal{P}(\mathcal{T}).\left\{ c|A \ : \ \begin{array}{c} c|\neg A \\ \triangle \end{array} \in I^- \right\}$ *and*
$\gamma = < \gamma^+, \gamma^- >$,
$\gamma^+ = \lambda I^+ \in \mathcal{P}(\mathcal{B}).\bigcup \left\{ \begin{array}{c} c|A \\ \triangle \end{array} \in \mathcal{T} \ : \ c|A \in I^+ \right\}$,
$\gamma^- = \lambda I^- \in \mathcal{P}(\mathcal{B}).\bigcup \left\{ \begin{array}{c} c|\neg A \\ \triangle \end{array} \in \mathcal{T} \ : \ c|A \in I^- \right\}$,
is a Galois insertion of $\mathcal{P}$(Trees) *into* $\mathcal{S}$.

Theorem 4. *The* T_P*-Trees-semantics is a collecting semantics w.r.t. the S-semantics.*

Hence by proposition 1 and theorem 4 we get

Corollary 2. *The T_P-Trees semantics $< \mathcal{P}(Trees), \mathcal{T}_{\mathcal{P}}^{\mathcal{T}\nabla\rceil\rceil\int} >$ is a collecting semantics.*

We thus obtain the hierarchy pictured in figure **??**. The generalization of the Heyting's semantics to normal CLP program has not been discussed. The idea, following [15, 12], is to collect in that semantics the successful upward closed finite T_P-trees. This leads to a semantics which is not comparable with the S-semantics (because of the upward closure) but which is more concrete than the Clark's semantics. For definite logic programs such a semantics provides an intuitionistic interpretation [15].

6.2 Top down semantics

It is interesting to examine the top down semantics corresponding to the concretization of operator T_P^S. Such a top-down semantics describes an inference rule of constructive negation for normal programs. As a matter of fact the inference rule we obtain has no simple relationship with the rule of constructive negation by pruning studied in [8], although both rules define the same set of computed answer constraints for each goal.

Thanks to the AND-compositional property we just have to define the rewriting process for literal goals only. So we define a transition relation $\rightarrow: \mathcal{T} \times \mathcal{T}$ over T_P-trees.

In the top-down semantics the tree is constructed by rewriting its leaves, then the constraints have to be propagated bottom-up to the root for the check of satisfiability. To this end we define a function *label*: $\mathcal{T} \rightarrow \mathcal{T}$ which labels a tree in a bottom-up fashion according to the labels on its leaves. Formally :

$$\mathit{label}\,(c|G) = c|G \text{ for a leaf,}$$

$$\mathit{label}\,\left(\begin{array}{c} c|L \\ \bigwedge_{and} \\ \triangle_1 \ldots \triangle_n \end{array}\right) = \begin{array}{c} \bigwedge_{i=1}^{n} c_i|G \\ \bigwedge_{and} \\ \triangle'_1 \ldots \triangle'_n \end{array} \text{ where } \begin{array}{c} c_i|G_i \\ \triangle'_i \end{array} = \mathit{label}(\triangle_i).$$

$$\mathit{label}\,\left(\begin{array}{c} c|L \\ \bigwedge_{or} \\ \triangle_1 \ldots \triangle_n \end{array}\right) = \begin{array}{c} \bigvee_{i=1}^{n} c_i|G \\ \bigwedge_{or} \\ \triangle'_1 \ldots \triangle'_n \end{array} \text{ where } \begin{array}{c} c_i|G_i \\ \triangle'_i \end{array} = \mathit{label}(\triangle_i).$$

The idea of the transition relation is to rewrite an atomic goal $true|A$ with a program clause, thereby producing and AND node, and to rewrite a goal $true|\neg A$ with all the program clauses defining A, thereby producing and AND node with OR nodes as leaves. But note that due to the non-deterministic choice of literals with repetition in program bodies for the formation of OR nodes (cf. def. of $T_P^{Trees^-}$), a tree will be rewritten top-down either by expanding a leaf or by adding a successor to an OR node.

Let $\triangle[true|L]$ denotes a tree containing either a leaf node labeled by $true|L$ or an OR node labeled by $c|\neg(d|\alpha)$ where $\neg L$ occurs in α. Then let us denote by

$$\triangle[\triangle']$$

the tree $\triangle$ in which either the leaf node has been replaced by $\triangle'$ or the OR node has been added a successor $\triangle'$.

The transition relation $\rightarrow: \mathcal{T} \times \mathcal{T}$ can now be defined as the least transition satisfying the following rules for positive and negative literals :

$$\triangle[true|A] \quad \rightarrow \quad \textit{label}\,(\triangle\ [\ \begin{array}{c} true|A \\ \bigwedge_{and} \\ d|\Box \quad true|L_1 \ldots true|L_n \end{array}\ ]\ )$$

where $A \leftarrow d|L_1, ..., L_n$ is a clause of the program and the root of the rewritten tree is satisfiable,

$$\triangle[true|\neg A] \quad \rightarrow \quad \textit{label}\,(\triangle\ [\ \begin{array}{c} true|\neg A \\ \bigwedge_{and} \\ d|\Box \quad \begin{array}{c} true|\neg(d_1|\alpha_1) \\ \bigwedge_{or} \end{array} \ \ldots \ \begin{array}{c} true|\neg(d_k|\alpha_k) \\ \bigwedge_{or} \end{array} \end{array}\ ]\ )$$

where $\{A \leftarrow d_1|\alpha_1., \ldots, A \leftarrow d_k|\alpha_k\}$ is the set of clauses defining A in the program, and the root of the rewritten tree is satisfiable,

Now a successful tree is a tree whose leaves are labeled by constraints only. The successful trees defined by the top-down rewriting relation are exactly the trees of the bottom-up semantics. For lack of space we cannot give more details, moreover a precise comparison of this top-down semantics (automatically) synthesized from the fixpoint semantics, and the known ones of for instance [20], [7] or [8] is not easy to establish, and should serve further investigation.

7 Conclusion and Perspectives

We have shown that the hierarchy of semantics for definite logic programs defined in [12] can be generalized to normal CLP programs. Note that this result, when restricted to definite CLP programs, still generalize the previous results on definite CLP programs, as we model finite failure, an observable which was out of the scope of the semantics described in [11, 12, 5].

More work is needed to effectively use the results presented here for the static analysis of normal CLP programs. In particular one should further investigate

– the systematic design of collecting semantics, and the notion of optimal collecting semantics w.r.t. an observable property,

– the abstract interpretation of normal CLP programs, and the definition of appropriate domains for groundness analysis, mode analysis, type analysis, etc.
– the study of compositional semantics w.r.t. various operators of program combination [5].
– the design of more flexible hierarchies capable of incorporating other standard operational semantics.

The results contained in this paper should serve as a basic ground for such extensions.

References

1. A. Bossi, M. Gabbrielli, G. Levi, M. Martelli, "The s-semantics approach: theory and applications", Journal of Logic Programming, 19-20, pp.149-197, 1994.
2. P. Cousot, R. Cousot, "Systematic design of program analysis frameworks", In Proc. 6th Annual Symposium on Principles of Programming Languages, pp.269-282, 1979.
3. P. Cousot, R. Cousot, "Abstract interpretation and application to logic programs", Journal of Logic Programming, 13(2 and 3), pp.103-179, 1992.
4. D. Chan, "Constructive negation based on the completed database", in: R.A. Kowalski and K.A. Bowen (eds), Proc. of the fifth International Conference on Logic Programming, MIT Press, Cambridge, MA, pp.11-125, 1988.
5. M. Comini, G. Levi, M.C. Meo, "Compositionality in SLD-derivations and their abstractions", Proc. of International Symposium on Logic Programming, ILPS'95, Portland, MIT Press, pp.561-575, 1995.
6. K. Doets "From Logic to Logic Programming", MIT Press, 1992.
7. W. Drabent, "What is failure? An approach to constructive negation", Acta Informatica, 32:1, pp.27-59, 1995.
8. F. Fages, "Constructive negation by pruning", LIENS technical report 94-14, revised 95-24. To appear in the Journal of Logic Programming, 1996.
9. M. Fitting, "A Kripke/Kleene semantics for logic programs", Journal of Logic Programming, 2(4), pp.295-312, 1985.
10. M. Falaschi, G. Levi, M. Martelli, C. Palamidessi, "Declarative modeling of the operational behavior of logic programs", Theoretical Computer Science, 69(3), pp.289-318, 1989.
11. M. Falaschi, G. Levi, M. Martelli, C. Palamidessi, "A model-theoretic reconstruction of the operational semantics of logic programs", Information and Computation, 103, pp.86-113, 1993.
12. R. Giacobazzi, ""Optimal" collecting semantics for analysis in a hierarchy of logic program semantics", Proc of 13th STACS'96, C. Puech and R Reischuk Ed., LNCS 1046, Springer Verlag, pp.503-514. 1996.
13. R. Gori, "A hierarchy of semantics for $CLP(\mathcal{A})$ general programs", Report of DEA IMA, LIENS, École Normale Supérieure, Paris, 1995.
14. J. Jaffar, J.L. Lassez, "Constraint Logic Programming", Proc. of POPL'87, Munich, pp.111-119, 1987.
15. R. Kemp, G. Ringwood, "Reynolds base, Clark models and Heyting semantics of logic programs", Technical Report, Queen Mary and Westfield College, 1991, revised 1995.

16. K. Kunen, “Negation in logic programming”, Journal of Logic Programming, 4(3), pp.289-308, 1987.
17. K. Kunen, “Signed data dependencies in logic programming”, Journal of Logic Programming, 7(3), pp.231-245, 1989.
18. M. Maher, “Equivalences of logic programs”, in: J. Minker (ed.) Foundations of deductive databases and logic programming, Morgan Kaufmann, pp.627-658, Los Altos, CA, 1988.
19. P. Stuckey, “Constructive negation for constraint logic programming”, Proc. LICS’91, pp.328-339, 1991.
20. P. Stuckey “Negation and constraint logic programming”, Information and Computation 118(1), pp.12-33 (1995).