Constructive Negation by Pruning and Optimization Higher-order Predicates for CLP and CC Languages

François FAGES

Laboratoire d'Informatique, URA 1327 du CNRS
Département de Mathématiques et d'Informatique
Ecole Normale Supérieure

LIENS - 94 - 19

October 1994

Constructive negation by pruning and optimization higher-order predicates for CLP and CC languages

François Fages

LIENS CNRS*, Ecole Normale Supérieure, 45 rue d'Ulm, 75230 Paris Cedex 05.
E-mail: fages@dmi.ens.fr

Abstract. We survey several forms of negation in constraint logic programming following the program's completion approach. We show that a new scheme called constructive negation by pruning provides a generic operational semantics which is correct and complete w.r.t. Kunen's three-valued logic semantics. We emphasis a full abstraction result which permits to go beyond the theorem proving point of view and to completely characterize the operational behavior of CLP programs with negation. We derive from these results a complete scheme for optimization higher-order predicates in CLP languages, and an operational semantics for concurrent constraint (CC) languages extended with negation and optimization higher-order agents.

1 Introduction

The amalgamation of constraint programming, logic programming and concurrent programming results in a very powerful model of computation that is conceptually simple and semantically elegant [13] [23].

Several constraint logic programming (CLP) systems and concurrent constraint (CC) systems have been developed over the last decade. These systems have been proved successful in complex problem modeling and combinatorial optimization problems across a variety of application domains, ranging from digital and analog circuits analysis and synthesis, to options trading and financial planning, job-shop scheduling, crew management, etc. [14].

In these realizations the real components of the problem at hand are modeled by relations over interface variables. These relations are defined with primitive constraints, recursively defined predicates, conjunctions and disjunctions. Relational models can thus be arbitrarily assembled with the CLP and CC programs constructors.

Extending the CLP and CC classes of languages with a negation operator is a major issue as it allows the user to express arbitrary logical combination of relational models. The full power of expression of first-order logic is then

* This work has been partially supported by MESR contracts 92 S 0777 and PRC AMN 93 S 0051.

accessible. In this way we obtain a framework to express also optimization and preferred solutions [10].

However negation in logic programming is known to be a delicate problem which raises many difficulties. Simply inferring negative information from a positive logic program is already a form of non-monotonic inference that shows essential differences between the two main approaches to the model theoretic semantics of logic programs: namely the *standard model* approach and the *program's completion* approach [1].

From a programming language point of view the standard model approach is not viable because it is highly undecidable. However from a knowledge representation point of view standard models correspond naturally to the intended semantics of programs. Therefore the challenge is to provide constructs which capture the essential aspects of standard models, in a recursively enumerable setting.

In this article we survey in a progressive manner the program's completion approach to CLP programs with negation. We introduce a new principle called constructive negation by pruning which is correct and complete w.r.t. the three-valued logical consequences of the program's completion. We emphasize a full abstraction result which permits to go beyond the theorem proving point of view and to completely characterize the operational behavior of normal CLP programs. These results are based on [8].

Then we show how constructive negation by pruning allows to define optimization higher-order predicates for CLP programs. We show that in this context the operational semantics specializes into an efficient branch and bound like procedure proved correct and complete in a full first-order setting.

Finally we study the fundamental extension of the class CC of concurrent constraint logic languages [23] with a negation operator. We show that the principle of constructive negation by pruning can be applied in this context. We derive from this principle an operational semantics for CC languages extended with negation and optimization higher-order agents.

2 Preliminaries on definite Constraint Logic Programs

A language of constraints is defined on a signature Σ of constants, function and predicate symbols (containing $true$, $false$ and $=$), and on a denumerable set V of variables. A *primitive constraint* is an atom 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 conjunction of primitive constraints. The set of free variables in an expression e is denoted by $FV(e)$.

CLP programs are defined using an extra 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 an atom or a negated atom $\neg A$.

A *definite CLP program* is a finite set of clauses of the form:

$$A \leftarrow c|A_1, ..., A_n$$

where $n \geq 0$, A is an atom, called the head, c a constraint, and the A_i's are atoms (the sequence is denoted by $\Box$ if $n = 0$). The *local variables* of a program clause is the set of free variables in the clause which do not occur in the head. A *definite goal* is a clause of the form

$$\leftarrow c|A_1, ..., A_n$$

where the A_i's are atoms (resp. literals). In the rest of this paper we shall assume that all atoms in programs and goals contain no constant or function symbol. Of course this is not a restriction as any program or goal can be rewritten in such a form by introducing new variables and equality constraints with terms. For instance $p(x+1) \leftarrow p(x)$ should be read as $p(y) \leftarrow y = x+1|p(x)$.

A $CLP(\mathcal{A})$ program is a CLP program given with a Σ-structure $\mathcal{A}$ which fixes the interpretation of constraints. An $\mathcal{A}$-valuation for a Σ-expression is a mapping $\theta : V \rightarrow \mathcal{A}$ which extends by morphism to terms and constraints. A constraint c is $\mathcal{A}$-solvable iff there exists an $\mathcal{A}$-valuation θ, s.t. $\mathcal{A} \models c\theta$.

We shall not suppose that $\mathcal{A}$ is solution compact [13], we suppose only that the constraints are decidable in $\mathcal{A}$, so that $\mathcal{A}$ can be presented by a first-order theory $th(\mathcal{A})$, satisfying:

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

.

$CLP(\mathcal{A})$ programs are interpreted operationally by a simple transition system on goals, $\rightarrow \in \mathcal{G} \times \mathcal{G}$, defined by the following CSLD resolution rule:

$$CSLD: \ \frac{(p(X) \leftarrow d|\beta) \in P \quad \mathcal{A} \models \exists(c \wedge d)}{c|\alpha, p(X), \alpha' \ \rightarrow \ c \wedge d|\alpha, \beta, \alpha'}$$

In such a transition, $p(X)$ is called the *selected* atom. A *CSLD derivation* is a sequence of transitions. A derivation is *successful* if it is finite and ends with a pure constraint goal containing no atom. A *computed answer constraint* (abbrev. c.a.c.) for a goal G is a constraint c such that there exists a successful derivation from G to $c|\Box$. A (CSLD) *derivation tree* for a goal G is the tree of all derivations from G obtained by fixing a selected atom in each node. The result of independence of the selection rule [17] states that all CSLD derivation trees of a given goal have the same set of computed answer constraints.

From a programming language point of view, computed answer constraints constitute a natural notion of observation which is finer than the simple existence of a successful derivation for a goal, characterized by ground success set semantics [13]. In this paper two programs will be said operationally equivalent if they have the same sets of c.a.c. (see [4] for other notions of observations). We shall thus consider formal semantics of CLP programs which permit to characterize computed answer constraints [18], [12], instead of ground semantics which, outside the case of pure logic programs over the Herbrand domain [2], generally do not suffice to modelize the operational behavior of CLP programs.

The c.a.c. for a composite goal can be retrieved from the c.a.c. for the atoms which appear in the goal (and-compositionality lemma 1), therefore the operational semantics of a $CLP(\mathcal{A})$ program is defined as a set of constrained atoms which gives the set of c.a.c. for unconstrained atomic goals solely:

Lemma 1 (And-compositionality lemma). *Let P be a $CLP(\mathcal{A})$ program. d is a computed answer constraint for the goal $c|A_1, ..., A_n$ if and only if there exist computed answer constraints $d_1, ..., d_n$ for the goals $true|A_1, ..., true|A_n$ respectively such that $d = c \wedge \bigwedge_{i=1}^{n} d_i$.*

Definition 2. The operational semantics of a CLP program P is the set $\mathcal{O}(P) = \{c|p(X) \mid p \in \Pi,\ c$ is a c.a.c. for the goal $true|p(X)\}$.

The corresponding logical semantics of a definite $CLP(\mathcal{A})$ program is given by the logical consequences of the clauses of the program together with the theory of the structure, whereas the algebraic semantics is defined by the truth in all $\mathcal{A}$-models of P.

Definition 3. The logical semantics of a $CLP(\mathcal{A})$ program P is the set

$$\mathcal{L}(P) = \{c|p(X) \mid p \in \Pi,\ P, th(\mathcal{A}) \models \forall(c \rightarrow p(X)) \wedge \exists(c)\}.$$

The algebraic semantics of P is the set

$$\mathcal{A}lg(P) = \{c|p(X) \mid p \in \Pi,\ P, \mathcal{A} \models \forall(c \rightarrow p(X)) \wedge \exists(c)\}.$$

The equivalence between the semantics can be expressed by inclusion and by several covering pre-orders on sets of constrained atoms:

- *strong covering*: $I \sqsubseteq J$ iff for all $c|A \in I$ there exists $d|A \in J$ such that $th(\mathcal{A}) \models c \rightarrow d$,
- *finite covering*: $I \sqsubseteq_f J$ iff for all $c|A \in I$, there exists $\{d_1|A, ..., d_n|A\} \subseteq J$ such that $th(\mathcal{A}) \models c \rightarrow \bigvee_{i=1}^{n} d_i$,
- *infinite covering*: $I \sqsubseteq_\infty J$ iff for all $c|A \in I$, there exists a (possibly infinite) set $\{d_k|A\}_{k \in K} \subseteq J$ such that[2] $\mathcal{A} \models c \rightarrow \bigvee_{k \in K} d_k$.

Theorem 4 (Soundness of CSLD resolution) [13]. $\mathcal{O}(P) \subseteq \mathcal{L}(P)$.

Theorem 5 ($th(\mathcal{A})$-completeness) [18]. $\mathcal{L}(P) \sqsubseteq_f \mathcal{O}(P)$.

Theorem 6 ($\mathcal{A}$-completeness). $\mathcal{A}lg(P) \sqsubseteq_\infty \mathcal{O}(P)$.

[2] Note that the infinite covering is defined via the truth in structure $\mathcal{A}$ of an infinite formula, whereas the finite and strong coverings are syntactic notions based on the logical consequences of $th(\mathcal{A})$.

It is worth noting that for the observation of answer constraints it is not equivalent to consider the logical consequences of $P \wedge th(\mathcal{A})$ or the truth in all $\mathcal{A}$-models of P. For the latter stronger algebraic semantics, the completeness result involves a possibly infinite set of c.a.c.

For example, with the logic program $P = \{p(0),\ p(s(X)) \leftarrow p(X)\}$ over the Herbrand universe $\mathcal{H}$ formed over function symbols 0 and s, the goal $\leftarrow p(X)$ admits an infinity of c.a.c. of the form $X = s^i(0)$ for $i \geq 0$. We have $P, \mathcal{H} \models \forall x p(x)$, and $\mathcal{H} \models \forall x \bigvee_{i \geq 0} x = s^i(0)$ indeed. However $P, CET \not\models \forall x p(x)$, we have $CET \not\models \forall x \bigvee_{i \geq 0} x = s^i(0)$, where CET is Clark's equational theory or any first-order theory of the Herbrand structure, as such a theory necessarily contains non-standard models in which $\bigvee_{i \geq 0} x = s^i(0)$ doesn't hold (otherwise the compactness theorem of first-order logic would be violated).

The departure of the algebraic semantics from the logical semantics becomes even more important with the study of finite failure.

3 Negative answers to definite goals

A further natural observable property of definite CLP program is finite failure, observed when all fair derivations of a goal are finite and not successful. A CSLD derivation is *fair* if it is finite or any atom in a goal of the derivation is selected within a finite number of steps. A goal G is *finitely failed* if any fair CSLD tree for G is finite and contains no successful derivations. Now the answer "no" is thus another possible outcome of an execution in addition to the computed answer constraints.

For this extra notion of observation, the logical semantics can no longer be based on the logical consequences of the clauses of the program as the set of all atoms instanciated in $\mathcal{A}$ (i.e. the $\mathcal{A}$-base) is a model of the program where everything is true. The solution proposed by Clark in 1978 is to consider instead the formula obtained from P by reading the definitions of the predicates with an equivalence symbol instead of implications.

The Clark's completion of a $CLP(\mathcal{A})$ program P is the conjunction of $th(\mathcal{A})$ with formulae
1) of the form

$$\forall X\ p(X) \leftrightarrow \bigvee_i \exists Y_i\ c_i \wedge \alpha_i$$

obtained for each predicate symbol p by collecting the clauses $\{p(X) \leftarrow c_i|\alpha_i\}$ in P, where $Y_i = FV(c_i|\alpha_i) \setminus X$,
2) or of the form $\forall X\ \neg p(X)$ if p doesn't appear in any head in P.
The Clark's completion of P is denoted by $P^*, th(\mathcal{A})$.

The finite failure rule is correct and complete w.r.t. the logical consequences of the program's completion without any restriction on the structure. It is instructive to see that the proof relies on the compactness theorem of first-order logic which holds for $P^*, th(\mathcal{A})$ but not necessarily in the $\mathcal{A}$-models of P^*.

Theorem 7 [13]. *A goal G is finitely failed if and only if $P^*, th(\mathcal{A}) \models \neg\exists(G)$.*

Proof. $\Rightarrow$ By induction on the height of the CSLD derivation tree.

$\Leftarrow$ We show that if G is not finitely failed then $\{P^*, th(\mathcal{A}), \exists G\}$ is satisfiable. If G is not finitely failed, then either G admits a successful derivation, in which case $P^*, th(\mathcal{A}) \models \exists G$ by the soundness theorem, or G admits a fair infinite derivation

$$G = c_0|\alpha_0 \longrightarrow c_1|\alpha_1 \longrightarrow c_2|\alpha_2 \longrightarrow ...$$

By the compactness theorem of first-order logic, $c_\omega = \bigcup c_i$ is $th(\mathcal{A})$-satisfiable. Let $\mathcal{B}$ be a model of $th(\mathcal{A})$ such that $\mathcal{B} \models \exists c_\omega$.
Let $I_0 = \{A\theta \mid A \in G_i \text{ for } i \geq 0 \text{ and } \mathcal{B} \models c_\omega\theta\}$ and let us consider the immediate consequence operator $T_P^{\mathcal{B}}$ of [13]. We have $I_0 \subseteq T_P^{\mathcal{B}}(I_0)$ (by fairness) hence as $T_P^{\mathcal{B}}$ is monotonic, by Knaster-Tarski's theorem, $T_P^{\mathcal{B}}$ admits a fixed point containing I_0, hence containing $G\theta$.
A fixed point of $T_P^{\mathcal{B}}$ is a $\mathcal{B}$-model of P^*, the previous fixed point is thus a $\mathcal{B}$-model of $P^*, \exists G$, therefore $P^*, th(\mathcal{A}), \exists G$ is satisfiable.

For example with the program $P = \{q(s(X)) \leftarrow q(X)\}$ over the Herbrand universe formed over function symbols 0 and s, the goal $\leftarrow q(0)$ is finitely failed and we have $P^*, CET \models \neg q(0)$. On the other hand the goal $\leftarrow q(X)$ admits an infinite fair derivation, we have $P^*, CET \not\models q(X)$ and $P^*, CET \not\models \neg q(X)$, despite the fact that $q(X)$ is false in all Herbrand models of P^*: $P^*, \mathcal{H} \models \neg q(X)$.

Therefore the logical consequences of $P^*, th(\mathcal{A})$ correctly capture the operational behavior of finite failure whereas the truth in all $\mathcal{A}$-models of P^* is not recursively enumerable and corresponds to an abstract notion of failure by ground derivations [14]. The situation is that from a knowledge representation point of view, the algebraic semantics is likely to reflect the intuition of the programmer who reasons in a fixed "domain of discourse" [13], but from a programming language point of view the algebraic semantics is not viable as it is highly undecidable. The logical semantics provides a declarative semantics which is faithful to the operational behavior of the program, and which constitutes a computable approximation of the "intended" algebraic semantics.

4 Goals with negation

The next step is to allow negation inside goals. The logical semantics can be extended accordingly with the following:

Definition 8. Let P be a definite $CLP(\mathcal{A})$ program. The logical semantics for positive and negative goals is defined by $\mathcal{L}_2(P) =< \mathcal{L}_2^+(P), \mathcal{L}_2^-(P) >$ where
$\mathcal{L}_2^+(P) = \{c|p(X) \mid P^*, th(\mathcal{A}) \models \forall(c \rightarrow p(X)) \wedge \exists c\}$
$\mathcal{L}_2^-(P) = \{c|p(X) \mid P^*, th(\mathcal{A}) \models \forall(c \rightarrow \neg p(X)) \wedge \exists c\}$

The operational semantics based on finite failure does not allow to compute an answer constraint for a negative goal containing variables, it is thus too weak w.r.t. the logical semantics. The principle of constructive negation proposed by

Chan [6] and Wallace provides a complete scheme. The main notion is the one of a frontier of a CSLD tree. A *frontier* of a CSLD tree is a finite set of nodes such that every derivation in the tree is either finitely failed or passes through exactly one frontier node. We can easily state the following:

Lemma 9. *Let P be a definite CLP($\mathcal{A}$) program, G a goal, and $\{c_i|\alpha_i\}_{1\leq i\leq n}$ be a frontier in a CSLD tree for G. Then $P^*, th(\mathcal{A}) \models G \leftrightarrow (\exists Y_1 c_1 \wedge \alpha_1) \vee \ldots \vee (\exists Y_n c_n \wedge \alpha_n)$.*

To resolve a goal $c|\neg A$, constructive negation amounts simply to develop a fair CSLD tree for $c|A$, take a frontier $\{c_i|\alpha_i\}_{1\leq i\leq n}$ in that tree, and return the answer constraint $c \wedge \bigwedge_{1\leq i\leq n} \neg\exists Y_i c_i$, whenever it is satisfiable, the deeper the frontier, the more general the answer (see figure 1).

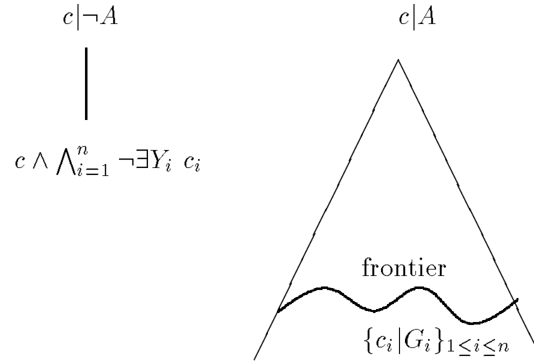

Fig. 1. Constructive negation for definite CLP programs.

As constraints must be negated, we shall suppose from now on that the language of constraints is closed by negation, and thus that a constraint can be any Σ, V-formula (a weaker assumption based on the notion of admissible structure is studied in [24]). The satisfaction completeness condition is then equivalent to say that the theory of the structure is complete, and thus that all models of the theory are elementary equivalent. In this setting soundness and completeness of constructive negation w.r.t. the logical semantics are a simple corollary of the previous theorem for finite failure.

Theorem 10. *Let P be a CLP($\mathcal{A}$) program, c be an $\mathcal{A}$-satisfiable constraint, and G a goal. Then $P^*, th(\mathcal{A}) \models c \rightarrow \neg G$ if and only if there exists a computed answer constraint d to the goal $\neg G$ such that $th(\mathcal{A}) \models c \rightarrow d$.*

Proof.

$\Leftarrow$ Let $\{c_i|\alpha_i\}_{i\in I}$ be a frontier in a CSLD tree for G such that $d = \bigwedge_{i\in I} \neg\exists Y_i\ c_i$ is satisfiable. We have $P^*, th(\mathcal{A}) \models G \leftrightarrow \bigvee_{i\in I} \exists Y_i\ c_i \wedge \alpha_i$,
so $P^*, th(\mathcal{A}) \models \neg G \leftrightarrow \bigwedge_{i\in I} \neg\exists Y_i\ c_i \vee \neg\alpha_i$,
hence $P^*, th(\mathcal{A}) \models \neg G \leftarrow d$.
$\Rightarrow$ We have $p^*, th(\mathcal{A}) \models \neg(c \wedge G)$, hence by theorem 7, the goal $c|G$ admits a finitely failed fair CSLD tree. Let $\{c_i|\alpha_i\}$ be the frontier corresponding to the lifting of that tree to the goal $true|G$.
For all $i \in I$, $c \wedge \exists Y_i c_i$ is $\mathcal{A}$-unsatisfiable, thus $c \wedge \bigvee_{i\in I} \exists Y_i c_i$ is $\mathcal{A}$-unsatisfiable. Let $d = \bigwedge_{i\in I} \neg\exists Y_i\ c_i$, we have $\mathcal{A} \models c \rightarrow d$, hence d is $\mathcal{A}$-satisfiable, therefore d is a computed answer constraint to the goal $\neg G$.

Note that for a negative goal a correct answer constraint is covered by a single computed answer constraint (strong completeness). Hence under the constructive negation rule a goal is not operationally equivalent to its double negation. In the later form the covering of a correct answer constraint is done by a single computed answer constraint which basically collects in a disjunction a finite set of computed answer constraints obtainable with the former form.

5 Programs with negation

5.1 Logical semantics

General (or normal) CLP programs allow negation in clause bodies. A *general (or normal) clause* is noted

$$A \leftarrow c|\alpha$$

where α is a finite sequence of literals (α^+ denotes the subsequence of atoms in α and $\Box$ denotes the empty sequence). General logic programs have the power of expression of full first-order logic, as any definition of a predicate by a first order formula can be transformed into a normal CLP program [17], obtained basically by replacing a clause of the form $p(X) \leftarrow \alpha, \forall Y q(X,Y), \beta$ by $p(X) \leftarrow \neg q'(X)$ and $q'(X) \leftarrow \neg q(X,Y)$ where q' is a new predicate symbol.

Now the Clark's completion of a normal program can be inconsistent, e.g. with $P = \{p \rightarrow \neg p\}$, $P^* = \{p \leftrightarrow \neg p\}$, in which case the logical semantics demands that there exists a successful derivation for each goal, that is clearly not the intended semantics of logic programs. The solution proposed by Kunen [15] to make the program's completion consistent is to define the logical semantics within three valued logic [11]. The usual strong 3-valued interpretations of the connectives and quantifiers are assumed, except for the connective $a \leftrightarrow b$ used to form the Clark's completion, 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 $\leftrightarrow$). In the previous example taking p undefined we have $u \leftrightarrow \neg u$ true. In this way the Clark's completion of a normal program is always 3-valued consistent. The logical consequence relation in three-valued logic is denoted by $\models_3$.

The *logical semantics* of a normal CLP($\mathcal{A}$) program P w.r.t. answer constraints is thus defined by the following partial constrained interpretation[3]:
$\mathcal{L}_3(P) =< \mathcal{L}_3^+(P), \mathcal{L}_3^-(P) >$ where
$\mathcal{L}_3^+(P) = \{c|p(X) \in \mathcal{B} \ : \ P^*, th(\mathcal{A}) \models_3 c \rightarrow p(X)\}$,
$\mathcal{L}_3^-(P) = \{c|p(X) \in \mathcal{B} \ : \ P^*, th(\mathcal{A}) \models_3 c \rightarrow \neg p(X)\}$.

5.2 Operational semantics

A first way to adapt the principle of constructive negation to general logic programs, that is to use it not only for the top-level goal, but recursively at each resolution step with a negative literal, is to transform the entire frontier obtained for a negated atom into disjunctive normal form, and produce a resolvant with each complex goal in a disjunct. The constructive negation rule is then the following:

$$CN : \quad (c|\alpha, (\neg\exists Y \ \beta), \alpha') \ \rightarrow \ (c \wedge c_j\,|\alpha, \beta_j, \alpha')$$

for each $1 \leq j \leq n$ where $\bigvee_{1\leq j\leq n} c_j \wedge \beta_j$ is a disjunctive normal form of $\bigwedge_{1\leq k\leq m} \neg\exists Z_k(c \wedge d_k \wedge \alpha_k)$, where $\{c \wedge d_k|\alpha_k\}_{1\leq k\leq m}$ is a frontier in a CSLDCN derivation tree for $c|\beta$, and $Z_k = (V(d_k|\alpha_k) \setminus V(c|\beta)) \cup Y$.

This is the way undertaken by Chan [6] for logic programs, and by Stuckey [24] for constraint logic programs. The effect is to introduce complex subgoals with explicit quantifiers and to compute the disjunctive normal form at each resolution step with a negative literal. This makes the scheme hardly amenable to a practical implementation for normal CLP programs in all generality. The compilative version proposed by Bruscoli et al. [5], named intensional negation, performs all disjunctive normal form transformations once and for all at compile time, but still all quantifiers need be explicit at run time and derivation rules need be defined for complex goals.

Another way undertaken independently in [7] and [8] is to use the principle of constructive negation as a concurrent pruning mechanism over standard CSLD

---

[3] It is worth noting that a complete notion of three-valued logic programming could also take into account the set
$\mathcal{L}_3^u(P) = \{c|p(X) \in \mathcal{B} \ : \ P^*, th(\mathcal{A}) \models_3 c \rightarrow (p(X) = u)\}$
of constrained atoms which are undefined in all three-valued models of the program's completion. This set is recursively enumerable. If it is not empty then the program's completion is clearly inconsistent. Of course the converse of that proposition doesn't hold. For instance the program

$$P = \{p \leftarrow \neg p, r, \ q \leftarrow \neg q, \neg r, \ r \leftarrow r\}$$

is inconsistent but $\mathcal{L}_3^u(P) = \emptyset$. To our knowledge, this generalized three-valued logic semantics has not been considered up to now, and there has been some confusion about the undefined truth value standing for goals which operationally loop forever. In fact the *absence* of a fixed truth valued for a goal should model operational loops, but a fixed undefined truth value for a goal could be distinguished and reported as a local contradiction in the program. A refinement of the operational semantics presented in the next section in order to detect such contradictions will be presented elsewhere.

trees. The idea to resolve a goal $c|\alpha, \neg A$ where $\neg A$ is the selected literal is to develop concurrently two CSLD-derivation trees, one Ψ for $c|\alpha, (\neg A)$ in which $\neg A$ is not selected, and one Ψ' for $c|A$.

Once a successful derivation is found in Ψ', say with answer constraint d, then Ψ is pruned by adding the constraint $\neg\exists Y d$ where $Y = V(d) \setminus V(c|A)$, to the nodes in Ψ where that constraint is satisfiable, and by removing the other nodes. This operation is called "pruning by success".

Once a successful derivation is found in Ψ, say with answer constraint e, we get a successful derivation for the main goal with answer constraint $f = e \wedge \bigwedge_{i=1}^{n} \neg\exists Y_i d_i$ where $Y_i = V(d_i) \setminus V(c|A)$, for each frontier $\{d_i|\alpha_i\}_{1\leq i\leq n}$ in Ψ' such that f is satisfiable (the deeper the frontier is, the more general is the computed answer). This operation is called "success by pruning".

The main goal is finitely failed if Ψ gets finitely failed after pruning. Figure 2 illustrates the pruning mechanism.

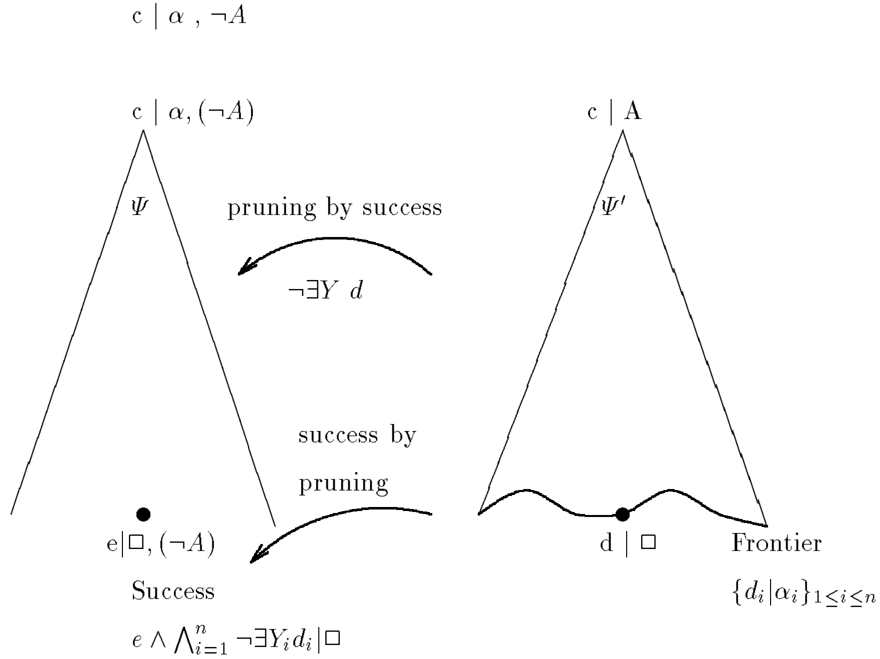

Fig. 2. Constructive negation by pruning.

The pruning by success rule is not redundant with the success by pruning rule. The former modifies the non successful nodes in a frontier of the first tree. This is necessary for finite failure or more generally for the completeness of the scheme if there are chains of dependencies through several negations.

Example 1. The nesting of negation can be illustrated by the following program:

```
p(0).
p(X):-p(X).
q(X):-not p(X).
```

with the goal:

```
? not q(X)
X=0
```

As the goal contains no positive literal the first derivation tree is a trivial success with constraint $true$. The second derivation tree for $q(X)$ contains one derivation to the goal $true|\neg p(X)$, thus a third derivation tree is developed for $p(X)$. As $X = 0$ is a success for $p(X)$, the pruning by success rule has for effect to prune the second with $X \neq 0$ (note that the success by pruning rule doesn't apply as any frontier in the third tree contains the goal $true|p(X)$ coming from the clause $p(X) \leftarrow p(X)$, and that goal cannot be negated). Hence by negating the frontier in the second tree after pruning we get a successful derivation for the main goal with answer constraint $X = 0$.

The practical advantage of constructive negation by pruning is that it relies on standard CSLD derivation trees for definite goals only. The only extra machinery to handle negation is a concurrent pruning mechanism over standard CSLD derivation trees, in particular there is no need for considering complex subgoals with explicit quantifiers.

In [8] constructive negation by pruning is formalized as a calculus over frontiers. The set of frontiers is the set $\mathcal{P}_f(\mathcal{G})$ of finite sets of goals. The main operation is the cross product of frontiers. Given two frontiers $F = \{c_i|\alpha_i\}_{1\leq i\leq m}$, and $F' = \{d_j|\beta_j\}_{1\leq j\leq n}$, the cross product of F and F' is the frontier:

$$F \times F' = \{(c_i \wedge d_j|\alpha_i, \beta_j) \ \mid \ 1 \leq i \leq m, 1 \leq j \leq n, \ \mathcal{A} \models \exists(c_i \wedge d_j)\}.$$

The negation of the projection of the constraint in a frontier F on a set of variables X is denoted by

$$\neg_X F = \bigwedge_{c|\alpha\in F, \ Y=V(c)\setminus X} \neg\exists Y c.$$

The operational semantics of general CLP programs is then defined by a relation $\lhd \in \mathcal{G} \times \mathcal{P}_f(\mathcal{G})$ which associates a frontier to a goal (big step semantics). Relation $\lhd$ is defined as the least relation satisfying the axioms and rules given in table 1.

The first rule (RES) is the usual resolution rule for positive literals (note that $c|p(x) \lhd \emptyset$ if $k = 0$). The second rule (FRT) expresses the formation of frontiers by cross products[4]. The third rule called "pruning" (PRN) is the new inference rule introduced for negative literals. It is worth noting that the usual negation as failure rule is the restriction of the pruning rule to the case $F = \emptyset$.

[4] Note that a more standard operational semantics where frontiers are not formed by cross products but by elementary CSLD resolution steps can be defined by replacing

TRIV: $c\vert\alpha \;\triangleleft\; \{c\vert\alpha\}$
RES: $\dfrac{c \wedge c_1\vert\alpha_1 \;\triangleleft\; F_1 \quad \ldots \quad c \wedge c_k\vert\alpha_k \;\triangleleft\; F_k}{c\vert p(X) \;\triangleleft\; F_1 \cup \ldots \cup F_k}$
where $\{(p(X) \leftarrow c_i\vert\alpha_i)\}_{1 \leq i \leq k}$ is the set of renamed clauses defining $p(X)$ in P such that $\mathcal{A} \models \exists(c \wedge c_i)$.
FRT: $\dfrac{c\vert A \;\triangleleft\; F \qquad c\vert\alpha, \alpha' \;\triangleleft\; F'}{c\vert\alpha, A, \alpha' \;\triangleleft\; F \times F'}$
PRN: $\dfrac{c\vert A \;\triangleleft\; F \qquad c \wedge \neg_X S\vert\alpha, \alpha' \;\triangleleft\; F'}{c\vert\alpha, \neg A, \alpha' \;\triangleleft\; \{c\vert\neg A\} \times F' \cup F''}$
where $X = V(c\vert A)$, S is a set of successful nodes in F and $F'' = \{(c \wedge \neg_X F\vert\alpha) \;:\; c\vert\alpha \in F', \alpha^+ = \emptyset\}$

Table 1. Definition of the goal-frontier relation for normal CLP languages.

Example 2. Going back to example 1, the goal $\neg q(x)$ has the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{x = 0|\Box \triangleleft \{x = 0|\Box\} \qquad true|p(X) \triangleleft \{true|p(X)\}}{true|p(x) \triangleleft \{x = 0|\Box,\; true|p(x)\}} \qquad x \neq 0|\Box \triangleleft \{x \neq 0|\Box\}}{true|\neg p(x) \triangleleft \{x \neq 0|\Box, x \neq 0|\neg p(x)\}}}{true|q(x) \triangleleft \{x \neq 0|\Box,\; x \neq 0|\neg p(x)\} \quad x = 0|\Box \triangleleft \{x = 0|\Box\}}}{true|\neg q(x) \triangleleft \{x = 0|\Box,\; x = 0|\neg q(x)\}}$$

Definition 11. The operational semantics of a general $CLP(\mathcal{A})$ program is the tuple: $\mathcal{O}(P) = < \mathcal{O}^+(P), \mathcal{O}^-(P) >$
$\mathcal{O}^+(P) = \{\exists Y\, c|p(X) \in \mathcal{B} \;:\; true|p(X) \triangleleft \{c|\Box\} \cup F,\; Y = V(c) \setminus X\}$
$\mathcal{O}^-(P) = \{c|p(X) \in \mathcal{B} \;:\; true|\neg p(X) \triangleleft \{c|\Box\} \cup F\}$

The next section presents completeness results for constructive negation by pruning through a fixed point semantics which is fully abstract for the observation of computed answer constraints

---

the RES and FRT rules by the following CSLD rule

$$CSLD : \dfrac{c \wedge c_1|\alpha, \alpha_1, \alpha' \;\triangleleft\; F_1 \quad \ldots \quad c \wedge c_k|\alpha, \alpha_k, \alpha' \;\triangleleft\; F_k}{c|\alpha, p(X), \alpha' \;\triangleleft\; F_1 \cup \ldots \cup F_k}$$

The only effect of this variant is to generate additional unnecessary redundant answer constraints. We refer to [8] for the details.

5.3 Fixed point semantics

Fitting [11] first introduced the idea that the formal semantics of normal logic programs should be defined in three-valued logic by partial interpretations. A partial interpretation I is a couple $< I^+, I^- >$ which determines truth and false things and leave undefined remaining atoms, and remaining formula by extension. Fitting's immediate consequence operator $\Phi_P^{\mathcal{A}}$ is defined for normal $CLP(\mathcal{A})$ programs by:

Definition 12. $\Phi_P^{\mathcal{A}}(I) =< {\Phi_P^{\mathcal{A}}}^+(I), {\Phi_P^{\mathcal{A}}}^-(I) >$ where
${\Phi_P^{\mathcal{A}}}^+(I) = \{A\rho \mid \rho$ is a valuation s.t. for some clause $(A \leftarrow c|\alpha) \in P$,
$\mathcal{A} \models c\rho,\ I(\alpha\rho) = t\}$
${\Phi_P^{\mathcal{A}}}^-(I) = \{A\rho \mid \rho$ is a valuation s.t. for all clause $(A \leftarrow c|\alpha) \in P$,
either $\mathcal{A} \not\models c\rho$, or $I(\alpha\rho) = t\}$.

Non-ground versions of Fitting's operator based on pairs of sets of constrained atoms have appeared in [15], [24], [5], [3], [8], as they are more suitable to establish the links with the operational semantics. A partial constrained interpretation is a pair $I =< I^+, I^- >$ of constrained interpretations such that $[I^+]_{\mathcal{A}} \cap [I^-]_{\mathcal{A}} = \emptyset$. Partial constrained interpretations form a semi-lattice for pairwise set inclusion (not a lattice as the union of two partial interpretations may be inconsistent), it is denoted by $(\mathcal{I}, \subseteq_3)$. The covering preorders are also extended pairwise to partial constrained interpretations.

The operator used in [8] is a *finitary* non-ground version of Fitting's operator: each constrained atom in the image of a constrained interpretation depends on a finite number of constrained atoms, such an operator is thus continuous in the semi-lattice of constrained partial interpretation.

Definition 13. Let P be a $CLP(\mathcal{A})$ program. $T_P^{\mathcal{A}}$ is an operator over $2^{\mathcal{B}} \times 2^{\mathcal{B}}$ defined by $T_P^{\mathcal{A}}(I) =< {T_P^{\mathcal{A}}}^+(I), {T_P^{\mathcal{A}}}^-(I) >$ where:
${T_P^{\mathcal{A}}}^+(I) = \{c|p(X) \in \mathcal{B}$: there exists a clause in P with local variables Y,
$p(X) \leftarrow d|A_1, ..., A_m, \neg A_{m+1}, ..., \neg A_n$
there exist $c_i|A_i \in I^+$ for $1 \le i \le m$,
$c_j|A_j \in I^-$ for $m+1 \le j \le n$,
such that $c = d \wedge \bigwedge_{i=1}^n c_i$ is $\mathcal{A}$-satisfiable$\}$
${T_P^{\mathcal{A}}}^-(I) = \{c|p(X) \in \mathcal{B}$: for any clause defining p in P, with local variable Y_k,
$p(X) \leftarrow A_{k,1}, ..., A_{k,m_k}, \alpha_k$, where $m_k \ge 0$,
there exist $\{e_{k,i}|A_{k,i}\}_{1 \le i \le m_k} \subseteq I^-$, $n_k \ge m_k$
$\{e_{k,j}|A_{k,j}\}_{m_k+1 \le j \le n_k} \subseteq I^+$ where $(\neg A_{k,j}) \in \alpha_k$,
such that $c_k = \forall Y_k(\neg d_k \vee \bigvee_{i=1}^{n_k} e_{k,i})$ is $\mathcal{A}$-satisfiable,
and $c = \bigwedge_k c_k$ is $\mathcal{A}$-satisfiable$\}$

Proposition 14. *$T_P^{\mathcal{A}}$ is a continuous operator in the semi-lattice $(\mathcal{I}, \subseteq_3)$.*

Definition 15. The fixed point semantics of a general CLP program is the set of constrained atoms

$$\mathcal{F}(P) = lfp(T_P^{\mathcal{A}}) = T_P^{\mathcal{A}} \uparrow \omega.$$

A somewhat surprising result from [8] is that this fixed point semantics is fully abstract for the observation of computed answer constraints with constructive negation by pruning.

Theorem 16 (Full abstraction)[8]. *$\mathcal{O}(P) = \mathcal{F}(P)$.*

Constructive negation by pruning is the first scheme to receive a fully abstract fixed point semantics w.r.t. computed answer constraints. This result means that the fixed point semantics fully characterizes the operational behavior of general CLP programs. It is thus possible to analyze and transform general CLP programs by reasoning at the fixed point semantics level of abstraction while preserving the equivalence based on the observation of computed answer constraints.

Completeness w.r.t. the logical semantics follows from the fact that the finite powers of $T_P^{\mathcal{A}}$ define the same ground partial interpretation as the finite powers of $\Phi_P^{\mathcal{A}}$. Hence by the result of Kunen [15] the fixed point semantics defines the same three-valued consequences as the Clark's completion of the program.

Lemma 17 [8]. *$[\mathcal{F}(P)] = \Phi_P^{\mathcal{A}} \uparrow \omega = [\mathcal{L}_3(P)]$.*

Theorem 18 (Completeness of the operational semantics) [8]. *$\mathcal{O}(P) \subseteq \mathcal{L}_3(P)$. $\mathcal{L}_3^+(P) \sqsubseteq_f \mathcal{O}^+(P)$ and $\mathcal{L}_3^-(P) \sqsubseteq \mathcal{O}^-(P)$.*

Here again one can remark that putting double negations on positive goals in the program suffices to obtain a strong completeness result w.r.t. the logical semantics (i.e. $\mathcal{L}_3(P) \sqsubseteq \mathcal{O}(P)$ instead of $\mathcal{L}_3(P) \sqsubseteq_f \mathcal{O}(P)$).

6 Optimization higher-order predicates

For crucial practical reasons, most CLP systems with arithmetic constraints, such as CHIP, CLP(R) or Prolog III, include metalevel facilities for finding optimal solutions to a goal w.r.t. an objective function [26]. These constructs do not belong however to the formal scheme of constraint logic programming. In [9] and [21] it is shown that optimization higher-order predicates can be defined with a faithful logical semantics based on constructive negation. It is interesting to see that constructive negation by pruning specializes in this context into an efficient concurrent branch and bound like procedure. Furthermore, completeness w.r.t. logical semantics and full abstraction of the fixed point semantics continue to hold without any restriction on the degree of nesting of, and the degree of recursion through, optimization predicates in the program.

Definition 19. Let $(\mathcal{A}, \leq)$ be a total order. The *minimization* higher-order predicate

$$min(G(X), f(X))$$

where $G(X)$ is a goal and $f(X)$ is a term, is defined as an abbreviation for the formula:

$$G(X) \wedge \neg\exists Y (f(Y) < f(X) \wedge G(Y))$$

A μCLP program over $\mathcal{A}$ is a definite CLP program over $\mathcal{A}$ which may contain minimization predicates in clause bodies.

μCLP programs can be transformed into normal CLP programs by reading $min(G(X), f(X)$ as:

$$G(X), \neg gf(X)$$

where gf is a new predicate symbol, and by adding the following clause to the program:

$$gf(X) \leftarrow f(Y) < f(X) | G(Y).$$

It is easy to see that in this context negation of constraints and negation of frontiers amount to a simple form of term minimization:

Proposition 20. *Let $(\mathcal{A}, \leq)$ be a total order.*
Let $d(X,Y,Z) = c(X) \wedge f(X) < f(Y) \wedge d'(Y,Z)$, then $c(X) \wedge \neg \exists Y,Z\ d(X,Y,Z)$ is $\mathcal{A}$-equivalent to $c(X) \wedge f(X) \leq v$, if $v = min_{d(X,Y,Z)} f(Y)$ exists, false otherwise.

Corollary 21. *If $c(X) \wedge f(Y) < f(X) | G(Y) \triangleleft F$ then the $c(X) \wedge \neg_X F$ is $\mathcal{A}$-equivalent to $c(X) \wedge f(X) \leq v$ if $v = min_{d|\alpha \in F} min_d f(Y)$ exists, false otherwise.*

The pruning by success rule of the general scheme can thus be replaced by a restricted form of pruning with a term minimization constraint. The next corollary shows that the success by pruning rule can be replaced by a check for finite failure after pruning.

Corollary 22. *Let $c(X) \wedge f(Y) < f(X) | G(Y) \triangleleft F$, $c(X)|\alpha, G(X), \alpha' \triangleleft \{d|\Box\} \cup F'$. Then $d \wedge \neg_X F$ is $\mathcal{A}$-satisfiable iff $w = min_d f(X)$ exists and $(f(X) \leq w) \times F = \emptyset$.*

The procedural interpretation can thus be simplified accordingly by replacing frontier computations with a check for finite failure. To resolve a goal of the form $c|\alpha, min(G(X), f(X)), \alpha'$, two CSLD derivation trees are developed, one Ψ for $c|\alpha, G(X), \alpha'$, and one Ψ' for $c \wedge f(Y) < f(X)|G(Y)$.

Once a successful derivation is found in Ψ', say with answer constraint d, then Ψ is pruned by adding the constraint $f(X) \leq v$ if $v = min_d f(Y)$ exists, false otherwise.

Once a successful derivation is found in Ψ, say with answer constraint e, then Ψ and Ψ' are pruned by adding the constraint $f(X) \leq w$ if $w = min_e f(X)$ exists, false otherwise.

By definition a successful derivation for the minimization goal is a successful derivation in Ψ such that Ψ' is finitely failed after pruning. The minimization goal is finitely failed if Ψ is finitely failed after pruning. Figure 3 illustrates the pruning mechanism.

As these modifications preserve the equivalence with the general scheme in the context of optimization predicates the results of the previous section continue to hold.

Theorem 23. *[8] Let P be a $\mu CLP(\mathcal{A})$ program. The fixed point semantics $\mathcal{F}(P)$ is fully abstract w.r.t. computed answer constraints. The operational semantics is sound and complete w.r.t. the logical semantics $\mathcal{L}(P)$.*

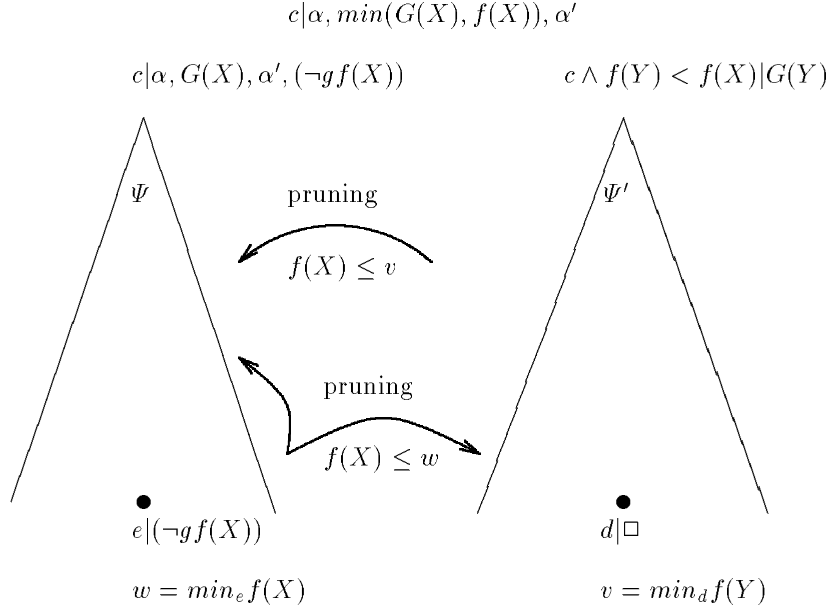

Fig. 3. Procedural interpretation of optimization predicates.

Example 3. An important particular case is when the only occurrence of the optimization predicate is in the head of the top-level goal. This is the case studied in [20]. This case is typical of scheduling applications for example where the top-level goal is

```
? min(schedule([X1,...,Xn], Xn))
```

and `schedule(L,X)` is defined by a definite CLP program over some numerical domain.

In that case both CSLD trees Ψ and Ψ' are identical up to variable renaming. The mutual pruning mechanism of the optimization scheme can thus be simplified into a single pruning in Ψ with constraint $f(X) \leq w$, as described in [20] or [26].

This is no longer true if the top-level goal contains a constraint or an atom outside the minimization predicate.

Example 4. Consider the $\mu CLP(R)$ program P

```
p(0)
p(X) :- X>1, p(X).
```

and the goal `X>1|min(p(X),X)`.

The first CSLD tree for `X>1|p(X)` is infinite. The second CSLD tree for `X>1,Y<X|p(Y)` contains a success with answer constraint $Y = 0$. The pruning

by success rule has for effect to prune the first tree with the constraint $X \leq 0$, therefore the first tree gets finitely failed and the answer to the minimization goal is no, in accordance to the logical semantics.

Note that the optimization procedures described in [26], [9] and [21] either incorrectly answer $X = 1$, or loop forever on this example. This shows the difficulty to define a complete scheme for optimization w.r.t. logical failures, and w.r.t. successes as well when minimization predicates are nested.

Completeness theorem 23 holds without any restriction on $\mu CLP(\mathcal{A})$ programs. Minimization predicates can thus be composed arbitrarily in a program, and, as an extreme case, one can also remark that recursion through optimization predicates is supported by the scheme.

The optimization predicates defined in [9] or [21] are however more general than those considered here as they allow to protect a set of variables in the goal subject to optimization. The effect is to localize the optimization to the remaining variables, and relativize the result to the set of *protected variables*.

Definition 24. The *local minimization* predicate

$$min(G(X,Y),[X],f(X,Y))$$

where $[X]$ is the set of protected variables is defined as an abbreviation for the formula

$$G(X,Y) \wedge \neg\exists Z(f(X,Z) < f(X,Y) \wedge G(X,Z)).$$

The local maximization predicate is defined similarly.

Example 5. For example local optimization predicates allow to express the min-max method of game theory simply with the following goal (for depth 2)

```
? max( min((move(X,Y),move(Y,Z)),[X,Y],val(Z)), [X], val(Z))
```

Note that protected variables are necessary in this example to conform to the intended semantics.

For local optimization predicates the previous operational scheme of optimization predicates is not correct as proposition 20 becomes irrelevant in presence of protected variables. This is not surprising and there is no hope to significantly improve a general scheme for negation in the context of local optimization predicates, as it is easy to see that any normal logic program can be encoded into a CLP program with local optimization predicates encoding negations [9].

7 Negation and optimization in CC languages

The class CC of concurrent constraint languages of Saraswat [23] adds to CLP languages a synchronization mechanism based on constraint entailment, called the *ask* operator. In this section we study the fundamental extension of CC languages with negation.

CC programs are usually presented with the following process algebra syntax to which we have added a negation operator, we call these programs *normal* CC programs:

Programs P::=	D.A	(declarations and empty agent)
Declarations D::=	ϵ	(empty declaration)
	$\mid D.D$	(sequence of declarations)
	$\mid p(X) :: A$	(procedure declaration)
Agents A::=	c	(tell)
	$\mid c \rightarrow A$	(ask)
	$\mid p(X)$	(procedure call)
	$\mid \exists X\ A$	(hiding)
	$\mid A \wedge A$	(conjunction)
	$\mid A \vee A$	(disjunction)
	$\mid \neg A$	(negation)

Provided the underlying structure $(\mathcal{A}, \leq)$ is a total order, normal CC programs allow to define an *optimization higher-order agent*

$$min(A(X), f(X))$$

as an abbreviation for the agent

$$A(X) \wedge \neg\exists Y.(f(Y) < f(X) \wedge A(Y)).$$

For the purpose of this article and to make clear the links with previous sections, we shall stick to the CLP syntax augmented with guarded literals to model the ask operator. A *guarded literal* will have the following syntax

$$(c \rightarrow L)$$

where c is a constraint and L a literal. A (normal) *CCLP program clause* is a clause

$$A \leftarrow c|\alpha$$

where A is a an atom, c is a (tell) constraint and α is a finite sequence of literals or guarded literals (α^+ will denote the subsequence of atoms in α, $\Box$ will denote the empty sequence). A (normal) *concurrent constraint logic program* (CCLP) is a finite set of guarded clauses. A (normal) CCLP goal has the following syntax

$$\leftarrow c|\alpha$$

where c is a (tell) constraint and α is a conjunction of literals or guarded literals. Clearly CC declarations can rewritten as CCLP programs, CC agents as CCLP goals, and CC programs as CCLP programs with a query goal,

Example 6. Several Prolog implementations make it possible to execute a goal in coroutine. For example the system predefined predicate `freeze(X,A)` has for effect to delay the selection of atom A as long as X is a free variable (i.e. unless `nonvar(X)` becomes true). This is a typical use of the ask operator over the Herbrand domain $\mathcal{H}$. Predicate `freeze` can be defined by the $CCLP(\mathcal{H})$ program `freeze(X,A):-(nonvar(X) -> A).`

Now the principle of constructive negation by pruning can be used to provide normal CCLP programs with an operational semantics. However the formation of frontiers by cross products for composite goals is not compatible with the guard mechanism based on the ask operator. Therefore we present the operational semantics of normal CCLP programs with a goal-frontier relation $\lll \in \mathcal{G} \times \mathcal{P}_f(\mathcal{G})$ defined with the CSLD rule in place of the RES and FRT rules, with a new rule for the ask, and with the PRN rule kept unchanged. Relation $\lll$ is defined as the least relation satisfying the axioms and rules given in table 2.

TRIV: $c|\alpha \lll \{c|\alpha\}$

CSLD: $$\frac{c \wedge c_1|\alpha, \alpha_1, \alpha' \lll F_1 \quad \ldots \quad c \wedge c_k|\alpha, \alpha_k, \alpha' \lll F_k}{c|\alpha, p(X), \alpha' \lll F_1 \cup \ldots \cup F_k}$$
where $\{(p(X) \leftarrow c_i|\alpha_i)\}_{1 \le i \le k}$ is the set of renamed clauses defining $p(X)$ in P such that $\mathcal{A} \models \exists(c \wedge c_i)$.

ASK: $$\frac{c|\alpha, L, \alpha' \lll F \qquad \mathcal{A} \models c \rightarrow d}{c|\alpha, (d \rightarrow L), \alpha' \lll F}$$

PRN: $$\frac{c|A \lll F \qquad c \wedge \neg_X S|\alpha, \alpha' \lll F'}{c|\alpha, \neg A, \alpha' \lll \{c|\neg A\} \times F' \cup F''}$$

where $X = V(c|A)$, S is a set of successful nodes in F and $F'' = \{(c \wedge \neg_X F|\alpha) \ : \ c|\alpha \in F', \alpha^+ = \emptyset\}$

Table 2. Definition of the goal-frontier relation for normal CCLP languages.

Note that the negation of frontiers ($\neg_X F$) is not affected by the presence of guarded literals. The guards are simply ignored just as in the standard semantics of CC [22] an agent $c \rightarrow A$ is blocked forever if the store entails $\neg c$[5].

Definition 25. Let P be a normal $CCLP(\mathcal{A})$ program. A *computed answer constraint* for a goal $c|\alpha$ is a constraint d such that $c|\alpha \lll \{c|\Box\} \cup F$. A *computed answer suspension* for a goal $c|\alpha$ is a goal $G = d|(d_1 \rightarrow \mathcal{L}_1), \ldots, (d_n \rightarrow \mathcal{L}_n)$ composed of a constraint and guarded literals, such that $c|\alpha \lll \{G\} \cup F$.

[5] Clearly one can also argue that if the store entails $\neg c$ the ask agent $(c \rightarrow A)$ should fail [23]. This convention can be accommodated in our scheme simply by negating the ask constraints as well as the tell constraints in the negation of a frontier.

Note that the opposite convention is used in [25], i.e. $c \rightarrow A$ succeeds if the store entails $\neg c$. This choice corresponds to the interpretation of the arrow as an implication in classical logic. This should not be confused with the interpretation of the ask operator as a pure control mechanism. The classical implication operator $c \Rightarrow A$ is defined with two ask operators by $c \rightarrow A \wedge \neg c \rightarrow true$

Example 7. Consider the following CCLP program

```
p(0).
p(1).
q(X,Y):-p(X),(X#0 -> p(Y))
```

The goal `?q(X,Y)` has one answer suspension and two answer constraints:

```
? q(X,Y)
X=1,Y=0;
X=1,Y=1;
X=0,(X#0 -> p(Y));
```

These answers are obtained with the following derivation:

$$\cfrac{\cfrac{x=0|\ (x\neq 0\to p(y))\lll\{x=0|(x\neq 0\to p(y))\} \qquad \begin{array}{c}\cfrac{\cfrac{x=1,y=0|\Box\lll\{x=1,y=0|\Box\} \qquad x=1,y=1|\Box\lll\{x=1,y=1|\Box\}}{x=1|p(y)\lll\{x=1,y=0|\Box,\ x=1,y=1|\Box\}}}{x=1|(x\neq 0\to p(y))\lll\{x=1,y=0|\Box,\ x=1,y=1|\Box\}}\\ ''\end{array}}{true|p(x),(x\neq 0\to p(y)\lll\{x=0|(x\neq 0\to p(y)),\ x=1,y=0|\Box,\ x=1,y=1|\Box\}}}{true|q(x,y)\lll\{x=0|(x\neq 0\to p(y)),\ x=1,y=0|\Box,\ x=1,y=1|\Box\}}$$

Example 8. If we place a negation on $p(x)$ in the definition of q in the previous example, then the program

```
p(0).
p(1).
q(X,Y):-not p(X),(X#0 -> p(Y))
```

produces two answer constraints and no answer suspension to the goal `?q(X,Y)`:

```
? q(X,Y)
X#0,X#1,Y=0;
X#0,X#1,Y=1;
```

The derivation is obtained basically with the following PRN derivation step:

$$\cfrac{p(x)\lll\{x=0|\Box,\ x=1|\Box\} \qquad \begin{array}{l}x\neq 0,x\neq 1|(x\neq 0\to p(y))\lll\\ \{x\neq 0,x\neq 1,y=0|\Box,\ x\neq 0,x\neq 1,y=1|\Box\}\end{array}}{\begin{array}{r}\neg p(x),(x\neq 0\to p(y))\lll\{x\neq 0,x\neq 1,y=0|\Box,\ x\neq 0,x\neq 1,y=1|\Box,\\ x\neq 0,x\neq 1,y=0|\neg p(x),\ x\neq 0,x\neq 1,y=1|\neg p(x)\}\end{array}}$$

If we look at the procedural interpretation, in order to resolve a goal $c|\alpha,\neg A,\alpha'$ where $\neg A$ is the selected literal, two CSLD resolution trees are developed following the CSLD and ASK rules, one Ψ for $c|\alpha,\alpha'$ and one Ψ' for $c|A$. Suspended goals in Ψ can be unblocked by the pruning by success rule, while suspensions

in Ψ' limit the choice of frontiers. Suspended goals in Ψ' however are never unblocked once $\neg A$ has been selected, although the constraints of some nodes in Ψ may entail the guard. This is a limitation of our operational scheme due to the principle of developing a single auxiliary CSLD derivation tree for the resolution of a negative literal.

8 Conclusion

The principle of constructive negation by pruning provides normal CLP programs with a complete operational semantics w.r.t. Kunen's three-valued logic semantics. The practical advantage of constructive negation by pruning for constraint logic programming is that it relies on standard CSLD derivation trees for definite goals only. The only extra machinery to handle negation is a concurrent pruning mechanism over standard CSLD derivation trees, in particular there is no need for considering complex subgoals with explicit quantifiers.

Constructive negation by pruning provides also a fundament to branch and bound procedures and min-max methods lifted to a full first-order setting. We have indicated a class of optimization higher-order predicates for which the operational semantics of constructive negation by pruning simplifies into a concurrent branch and bound like procedure without frontier computation.

On the theoretical side, constructive negation by pruning possesses a fully abstract fixed point semantics w.r.t. computed answer constraints. The fixed point semantics is based on a simple finitary version of Fitting's operator. The full abstraction result shows that the operational behavior of the program is fully characterized by the fixed point semantics.

Finally we have shown that the principle of constructive negation by pruning could be used to extend the class CC of concurrent constraint languages with negation and optimization higher-order agents. The operational semantics of normal CLP programs has been generalized to deal with normal CC programs. This fundamental extension now raises many interesting questions both on the operational aspects of the interactions between ask and negation, and on the nature of a denotational semantics in the style of [23] for normal CC languages.

References

1. K. Apt, R. Bol, “Logic programming and negation: a survey”, Journal of Logic Programming, 19-20, pp.9-71, 1994.
2. K. Apt, M. Gabrielli, “Declarative interpretations reconsidered”, Proc. 11th ICLP'94, MIT Press, 1994.
3. A. Bossi, M. Fabris, M.C. Meo, “A bottom-up semantics for constructive negation”, Proc of the 11th Int. Conf. on Logic Programming, pp.520-534, MIT Press, 1994.
4. 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.
5. P. Bruscoli, F. Levi, G. Levi, M.C. Meo, “Intensional negation”, GULP'93, eight conference on logic programming, Italy. June 1993.

6. 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.
7. W. Drabent, ”What is failure? An approach to constructive negation”, to appear in Acta Informatica, 1994.
8. F. Fages, “Constructive negation by pruning”, Technical report 94-14, Ecole Normale Supérieure, Paris. Sept. 1994. Submitted for publication.
9. F. Fages, “On the semantics of optimization predicates in CLP languages”, 13th FSTTCS conference, Bombay, LNCS 761, Springer-Verlag, pp. 193-204, 1993.
10. F. Fages, J. Fowler, T. Sola, “Handling preferences in constraint logic programming with relational optimization”, Proc of PLILP’94, Madrid, 1994.
11. M. Fitting, “A Kripke/Kleene semantics for logic programs”, Journal of Logic Programming, 2(4), pp.295-312, 1985.
12. M. Gabbrielli, G. Levi, “Modeling answer constraint in constraint logic programs”, ICLP’91, Paris, MIT Press, 1991.
13. J. Jaffar, J.L. Lassez, “Constraint Logic Programming”, Proc. of POPL’87, Munich. 1987.
14. J. Jaffar, M.J. Maher, “Constraint logic programming: a survey”, Journal of Logic Programming, 19-20, 1994.
15. K. Kunen, “Negation in logic programming”, Journal of Logic Programming, 4(3), pp.289-308, 1987.
16. K. Kunen, “Signed data dependencies in logic programming”, Journal of Logic Programming, 7(3), pp.231-245, 1989.
17. J.W. Lloyd, “Foundations of Logic Programming”, Springer Verlag. 1987.
18. M.J. Maher, “Logic semantics for a class of committed-choice languages”, Proc. 4th International Conference on Logic Programming, pp.858-876, MIT Press, 1987.
19. M.J. Maher, “A logic programming view of CLP”, Proc. 10th International Conference on Logic Programming, pp.737-753, MIT Press, 1993.
20. M. Maher, P.J. Stuckey, “Expanding query power in constraint logic programming languages”, Proc. NACLP’89, MIT Press, 1989.
21. K. Marriott, P.J. Stuckey, “Semantics of CLP programs with optimization”, Technical report, Univ. of Melbourne, 1993.
22. V. Saraswat, “Concurrent constraint programming”, Proc. POPL’90, San Francisco, pp.232-245, 1990.
23. V. Saraswat, “Concurrent constraint programming”, MIT Press, 1993.
24. P. Stuckey, “Constructive negation for constraint logic programming”, Proc. LICS’91, 1991.
25. P. Van Hentenryck, H. Simonis, M. Dincbas, “Constraint satisfaction using constraint logic programming”, Artificial Intelligence 58, pp.11-159, 1992.
26. P. Van Hentenryck : ”Constraint Satisfaction in Logic Programming”, MIT Press 1989.

This article was processed using the LaTeX macro package with LLNCS style