ECOLE NORMALE SUPERIEURE

Constructive Negation by Pruning

(revised version of Liens 94-14)

François FAGES

LIENS - 95 - 24

Département de Mathématiques et Informatique

CNRS URA 1327

Constructive Negation by Pruning

(revised version of Liens 94-14)

François FAGES

LIENS - 95 - 24

September 1995

Laboratoire d'Informatique de l'Ecole Normale Supérieure
45 rue d'Ulm 75230 PARIS Cedex 05

Tel : (33)(1) 44 32 00 00

Adresse électronique : .. @dmi.ens.fr

CONSTRUCTIVE NEGATION BY PRUNING

François FAGES

▷ We show that a simple concurrent pruning mechanism over standard SLD derivation trees, called constructive negation by pruning, provides a correct and complete operational semantics for normal constraint logic programs w.r.t. Fitting-Kunen's 3-valued logic semantics. We argue that this scheme is simple enough to lead to practical implementations as the principle of concurrent pruning is the only extra machinery needed to handle negation, in particular there is no need for considering complex subgoals with explicit quantifiers outside the constraint part. We study a non-ground continuous finitary version of Fitting's operator, and we show that the corresponding fixpoint semantics is fully abstract for the observation of computed answer constraints.

In the context of optimization higher-order predicates, that are common practice in CLP systems and that can be expressed by a logical formula with negation, we show that constructive negation by pruning specializes to an efficient concurrent branch and bound like procedure, proved correct and complete without any restriction on the degree of nesting of, and on the degree of recursion through, optimization predicates in the program. ◁

Contents

Address correspondence to LIENS, URA 1327 CNRS, Ecole Normale Supérieure, 45 rue d'Ulm, 75230 Paris Cedex 05, France. E-mail: `fages@dmi.ens.fr`.

1. Introduction

Constraint logic programming and concurrent constraint programming are simple and powerful models of computation that have been implemented in several systems over the last decade, and proved successful in a variety of applications ranging from combinatorial optimization problems to complex system modeling [14]. Extending these classes of languages with a negation operator is a major issue as it allows the user to express arbitrary logical combinations of relations and provides a framework for optimization higher-order predicates [8].

Negation in logic programming has been extensively studied due to the problems of non-monotonicity and non recursive enumerability of the canonical model approach [1] [17]. On the theoretical side these difficulties have been satisfactorily solved by Kunen [16] and Fitting [11] who proposed to define the declarative semantics of a program by the set of the 3-valued logical consequences of its Clark's completion, and to construct fixpoint semantics in the semi-lattice of partial interpretations. On the implementation side, most constraint logic programming systems allow restricted forms of negation, but the operational mechanism based for instance on negation by failure is too weak w.r.t. Kunen's logical semantics, and the restriction to negative goals containing no variable doesn't fit well with constraint programming. Other ad hoc mechanisms are thus added in most CLP systems for dealing with optimization predicates for instance [24].

Constructive negation, as introduced by Chan [6] for logic programs, and generalized to CLP programs by Stuckey [23], provides an operational mechanism that is correct and complete w.r.t. Kunen's three-valued logical semantics of programs with negation. However the schemes proposed by Chan and Stuckey are not easily amenable to a practical implementation as they necessitate to deal with explicitly quantified complex subgoals, and to compute the disjunctive normal form of a complex formula at each resolution step with a negative subgoal. 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.

In this paper we present a new scheme for constructive negation based on a pruning mechanism over standard SLD-derivation trees, without the need for considering explicitly quantified complex subgoals. The formalism we develop is based on a simple frontier calculus. The resulting execution model is essentially equivalent to the one proposed independently by Drabent for normal logic programs [7]. We argue that this scheme is simple enough to lead to practical implementations as the principle of concurrent pruning is the only extra machinery needed to handle negation. We study a non-ground continuous finitary version of Fitting's operator (similar to the operators studied in [23], [5] and [3]), and we show that the corresponding fixpoint semantics is fully abstract for the observation of computed answer constraints.

In the context of optimization higher-order predicates, that are common practice in CLP systems and that can be expressed logically by a formula with negation, we show that our general scheme specializes to an efficient concurrent branch and bound like procedure, proved correct and complete without any restriction on the degree of nesting of, and on the degree of recursion through, optimization predicates in the program.

2. Preliminaries on Constraint Logic Programming

We recall the basic concepts of constraint logic programming (CLP) as defined in [13], with some different emphasis due to our interest in negation. Concerning the declarative semantics of CLP programs we focus on the logical semantics instead of the algebraic semantics which is highly undecidable, doing so some conditions such as solution compactness [13] become irrelevant. We adopt also the point of view of [12] and [18] that for a programming language the observation of computed answer constraints is a more natural choice of observable than the success set considered in [13], and that the formal semantics of CLP programs should characterize the set of computed answer constraints. We shall thus present formal semantics accordingly with sets of constrained atoms [4]. Before that we fix notations and make precise the constraint languages and structures considered for CLP programs with negation.

2.1. Constraint languages with negation

The first-order language of constraints is defined on a countably 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, $Y, ...$ and we shall sometimes write $e(X)$ if $V(e) = X$. 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 $\theta : V \rightarrow \mathcal{A}$ 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 $\mathcal{A} \models \exists c$.

It is not necessary for our purpose to suppose that $\mathcal{A}$ is solution compact [13] [19], we suppose only that the 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 [15].

In practice however, the language of constraints will often be a restricted class of Σ, V-formulae, assumed to be closed only by renaming, conjunction and existential quantification, not by negation. Stuckey [23] calls such a restriction a language of *admissible constraints*, which intuitively represents the constraints the solver can deal with. A structure $\mathcal{A}$ is then said to be *admissible* if the negation of an admissible constraint is equivalent to a disjunction of admissible constraints:

$$\mathcal{A} \models \forall X(\neg\exists Y\, c(X,Y) \;\leftrightarrow\; \exists Z_1\; d_1(X,Z_1) \vee ... \vee \exists Z_n d_n(X,Z_n))$$

For the sake of simplicity, we shall assume in this paper that the language of constraints is closed by negation, but we shall indicate latter in section 6 how our scheme can be easily modified to deal with admissible constraints only, when the structure $\mathcal{A}$ is admissible.

2.2. $CLP(\mathcal{A})$ 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. A *definite (resp. normal) goal* is a formula $c|L_1, ..., L_n$ where $L_1, ..., L_n$ are atoms (resp. literals). We will identify conjunction "," and multiset union, greek letters, α, β,... will be used to denote multisets of literals, so that a goal (resp. a clause) will be sometimes written $c|\alpha$ (resp. $A \leftarrow c|\alpha$), we shall denote by α^+ (resp. α^-) the multiset of positive (resp. negative) literals in α, the empty multiset is noted $\Box$. The set of goals is denoted by $\mathcal{G}$. In the rest of this paper we shall assume that all atoms in programs and goals contain no constant, no function symbol and no multiple occurrences of a same variable. Of course this is not a restriction as any program or goal can be rewritten in such a *standard form* by introducing new variables and equality constraints with terms. For instance the clause $p(x+1) \leftarrow p(x)$ will be read as $p(y) \leftarrow y = x+1|p(x)$.

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. A ground atom $A\theta$ is true (resp. false) in I if $A\theta \in [I]_{\mathcal{A}}$ (resp. $A\theta \notin [I]_{\mathcal{A}}$).

Constraint entailment defines a natural preorder on constrained atoms, called the *covering preorder*: $c|A \sqsubseteq d|A$ iff $th(\mathcal{A}) \models c \to d$. Note that as $th(\mathcal{A})$ is a complete theory, $c|A \sqsubseteq d|A$ is equivalent to $[c|A]_{\mathcal{A}} \subseteq [d|A]_{\mathcal{A}}$. The covering preorder extends to sets of constrained atoms in two ways: *strong covering* (used for strong completeness results),

$$I \sqsubseteq J\ iff\ \forall c|A \in I\ \exists d|A \in J\ th(\mathcal{A}) \models c \to d$$

and *finite covering*,

$$I \sqsubseteq_f J\ iff\ \forall c|A \in I\ \exists \{d_1|A, ..., d_n|A\} \subseteq J\ th(\mathcal{A}) \models c \to \bigvee_{i=1}^{n} d_i.$$

The operational semantics of *definite* $CLP(\mathcal{A})$ programs is based on a simple transition relation on definite goals, defined by the following SLD derivation rule:

$$SLD:\ c|\alpha, p(X), \alpha' \ \to\ c \wedge c_i|\alpha, \alpha_i, \alpha'$$

where $p(X) \leftarrow c_i|\alpha_i$ is any renamed clause defining p in P such that $\mathcal{A} \models \exists(c \wedge c_i)$. A computed answer constraint (c.a.c.) for a definite goal $c|\alpha$ is a constraint of the form $\exists Y\ d$ where $Y = V(d) \setminus V(c|\alpha)$ such that

$$c|\alpha \ \to^*\ d|\Box$$

where $\to^*$ is the reflexive transitive closure of $\to$. An and-compositionality lemma states that a c.a.c. d for a composite goal $c|A_1, ..., A_n$ is of the form $d = c \wedge \bigwedge_{i=1}^{n} c_i$ where the c_i's are c.a.c. for atomic goals $true|A_i$. Thus the operational behavior of definite $CLP(\mathcal{A})$ programs w.r.t. answer constraints is fully characterized by the following set of constrained atoms:

$$O(P) = \{\exists Y c|p(X) \in \mathcal{B}\ :\ true|p(X) \to^* c|\Box,\ Y = V(c) \setminus X\}$$

Taking as logical semantics

$$\mathcal{L}(P) = \{c|p(X) \in \mathcal{B}\ :\ P, th(\mathcal{A}) \models c \to p(X)\}$$

we obtain the well-known soundness, $\mathcal{O}(P) \subseteq \mathcal{L}(P)$, and completeness, $\mathcal{L}(P) \sqsubseteq_f \mathcal{O}(P)$, results of SLD-resolution for definite $CLP(\mathcal{A})$ programs w.r.t. answer constraints [18] [12].

The logical semantics of *normal* $CLP(\mathcal{A})$ programs is defined via the Clark's completion of the program. The *Clark's completion* of a $CLP(\mathcal{A})$ program P is 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 with a set of clauses $\{p(X) \leftarrow c_i|\alpha_i\}_{1\leq i\leq n} \in P$, where $Y_i = V(c_i|\alpha_i) \setminus X$, and

$$\forall X \; \neg p(X)$$

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 $P = \{p \rightarrow \neg p\}$, $P^* = (p \leftrightarrow \neg p)$, 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, such contradictions must be localized in the program, the solution proposed by Kunen is to define the logical semantics as the set of 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 $\leftrightarrow$). In the previous example we can assign the undefined truth value to predicate p so that $u \leftrightarrow \neg u$ is true, more generally Fitting [11] showed that any normal logic program has 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 preorder $\sqsubseteq$ extends to partial interpretation by $I \sqsubseteq J$ iff $I^+ \sqsubseteq J^+$ and $I^- \sqsubseteq J^-$. And similarly for $\sqsubseteq_f$.

The *logical semantics* of a normal CLP($\mathcal{A}$) program P is defined by the following partial interpretation:

$\mathcal{L}(P) =< \mathcal{L}^+(P), \mathcal{L}^-(P) >$ where
$\mathcal{L}^+(P) = \{c|p(X) \in \mathcal{B} \; : \; P^*, th(\mathcal{A}) \models_3 c \rightarrow p(X)\}$,
$\mathcal{L}^-(P) = \{c|p(X) \in \mathcal{B} \; : \; P^*, th(\mathcal{A}) \models_3 c \rightarrow \neg p(X)\}$.

The aim of this paper is to study a complete operational semantics for normal $CLP(\mathcal{A})$ programs.

3. Constructive negation by pruning

3.1. Procedural interpretation on SLD derivation forests

Constructive negation by pruning can be presented informally as a simple pruning mechanism over standard SLD-derivation trees. The idea to resolve a goal $c|\alpha, \neg A$ where $\neg A$ is the selected literal is to develop concurrently two SLD-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" (PBS).

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[1] $\{d_i|\alpha_i\}_{1\leq i\leq n}$ in Ψ' such that f is satisfiable (the deeper the frontier, the more general the computed answer). This operation is called "success by pruning" (SBP).

The main goal is finitely failed if Ψ gets finitely failed after pruning. Figure 1 illustrates the pruning mechanism.

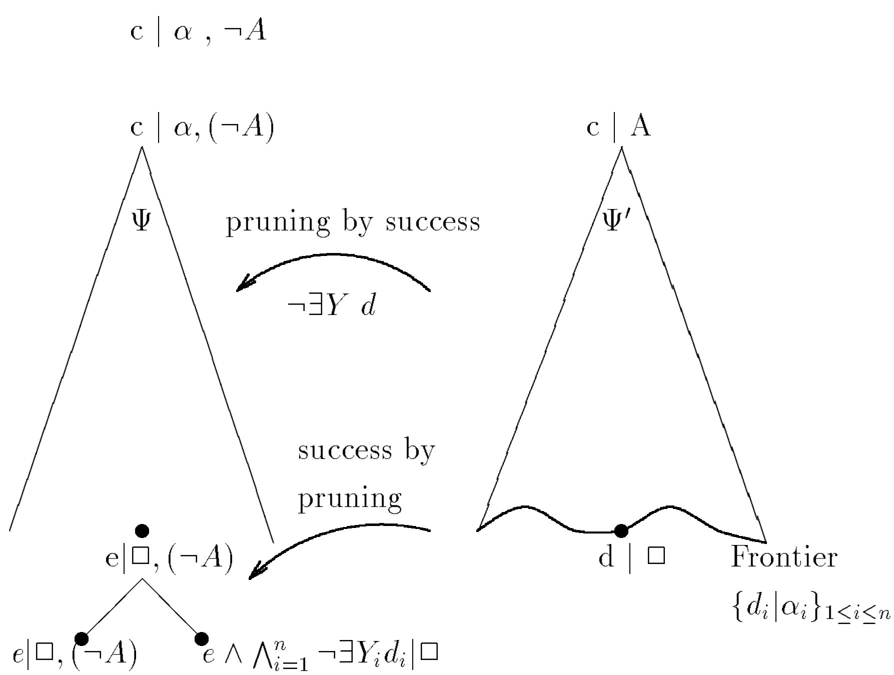


Figure 1. Constructive negation by pruning.

Example 3.1. The nesting of negation can be illustrated by the following program:

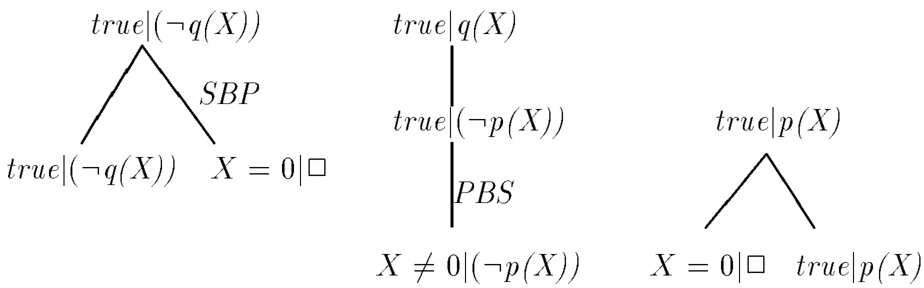
```
p(X):-X=0.
p(X):-p(X).
q(X):-not p(X).
```

with the goal:

[1] A frontier in a SLD-derivation tree is a finite set of nodes in the tree such that every derivation in the tree is either finitely failed or passes through exactly one node of the frontier.

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

As the query contains no positive literal the first derivation tree is initially trivial. A second derivation tree is developed for $true|q(X)$*, that tree contains one derivation to the goal* $true|\neg p(X)$*, thus a third derivation tree is developed for* $true|p(X)$*. As* $X = 0$ *is a success for* $p(X)$*, the second tree can be pruned with* $X \neq 0$ *by using the PBS rule (note that the SBP rule doesn't apply here as any frontier in the third tree contains the goal* $true|p(X)$ *whose constraint cannot be negated). Then by negating the frontier in the second tree after pruning and by applying the SBP rule we get a successful derivation for the query with answer constraint* $X = 0$*.*



3.2. Operational semantics

3.2.1. Uniform derivations We shall first define the operational semantics of constructive negation by pruning with a simple calculus on frontiers of *uniform* SLD trees, i.e. SLD trees such that a tree for $c|\alpha, \alpha'$ is a combination of a tree for $c|\alpha$ and of a tree for $c|\alpha'$.

The set of frontiers is the set $\mathcal{P}_f(\mathcal{G})$ of finite sets of goals. The calculus is based on a binary operator: the usual cross product of frontiers, $\times$, and on a negation operator for frontiers w.r.t. a set of variables V, noted $\neg_V F$, which associates to a frontier F the constraint representing the negation of the projection on V of the constraints in F:

Definition 3.2. Given two frontiers $F = \{c_i|\alpha_i\}_{i\in I}$, $F' = \{d_j|\beta_j\}_{j\in J}$, *let us define*

$$F \times F' = \{(c_i \wedge d_j|\alpha_i, \beta_j) \mid i \in I,\ j \in J,\ \mathcal{A} \models \exists(c_i \wedge d_j)\}$$

$$c \times F = \{c|\square\} \times F = \{(c \wedge c_i|\alpha_i) \mid i \in I,\ \mathcal{A} \models \exists(c \wedge c_i)\}$$

$$\neg_V F = \bigwedge_{i\in I} \neg\exists Y_i\ c_i,\ \textit{where}\ Y_i = V(c_i) \setminus V$$

$$\mathcal{S}(F) = \{c|\square \in F \mid \mathcal{A} \models \exists c\}$$

$\mathcal{S}(F)$ is the set of successes in F. $c \times F$ is called the pruning of F by constraint c, that operation will be used to formalize the "pruning by success" rule (PBS) of the previous section.

One can easily check that $(\mathcal{P}_f(\mathcal{G}), \cup, \emptyset, \times, \{true|\Box\})$ is a commutative semi-ring:

$$\times \textit{ is associative and commutative}, \tag{1}$$
$$F \times \emptyset = \emptyset, \tag{2}$$
$$\times \textit{ distributes over } \cup, \tag{3}$$
$$\text{furthermore, } \neg_V \emptyset = true, \tag{4}$$
$$(\neg_V F) \times F = \emptyset, \tag{5}$$
$$\neg_V(F \cup F') = (\neg_V F) \wedge (\neg_V F'), \tag{6}$$
$$\neg_V(F \times F') = (\neg_V F) \vee (\neg_V F'), \tag{7}$$
$$\mathcal{S}(F \times F') = \mathcal{S}(F) \times \mathcal{S}(F'). \tag{8}$$

Now the relation $\triangleleft \in \mathcal{G} \times \mathcal{P}_f(\mathcal{G})$ which associates a frontier to a goal, can be defined inductively as the least relation satisfying the following axiom and rules[2]:

TRIV: $c|\alpha \ \triangleleft\ \{c|\alpha \ : \ \mathcal{A} \models \exists(c)\}$

RES: $$\frac{c \wedge c_1|\alpha_1 \ \triangleleft\ F_1 \quad \ldots \quad c \wedge c_k|\alpha_k \ \triangleleft\ F_k}{c|p(X) \ \triangleleft\ F_1 \cup \ldots \cup F_k}$$

where $\{(p(X) \leftarrow c_i|\alpha_i)\}_{1 \leq i \leq k}$ is the set of renamed apart clauses defining $p(X)$ in P such that $\mathcal{A} \models \exists(c \wedge c_i)$, and we assume $(V(F_i) \setminus V(c \wedge c_i|\alpha_i)) \cap X) = \emptyset$ in order to avoid variable clashes.

FRT: $$\frac{c|\alpha_1 \ \triangleleft\ F_1 \qquad c|\alpha_2 \ \triangleleft\ F_2}{c|\alpha_1, \alpha_2 \ \triangleleft\ F_1 \times F_2}$$

where $\alpha_1 \neq \Box$, $\alpha_2 \neq \Box$ and in order to avoid variable clashes we assume $V_1 \cap V(\alpha_2) = \emptyset$, $V_2 \cap V(\alpha_1) = \emptyset$, $V_1 \cap V_2 = \emptyset$ where $V_1 = V(F_1) \setminus V(c|\alpha_1)$ and $V_2 = V(F_2) \setminus V(c|\alpha_2)$.

PRN: $$\frac{c|A \ \triangleleft\ F}{c|\neg A \ \triangleleft\ c \times \{\neg_V S|\neg A,\ \neg_V F|\Box\}}$$

where $S \subseteq \mathcal{S}(F)$ and $V = V(c|A)$.

Figure 2. Inductive definition of the goal-frontier relation for uniform derivations.

Rule RES is the usual resolution rule for positive literals. Rule FRT expresses the formation of frontiers by cross products (a more standard operational semantics

[2]This presentation of the operational semantics is not in the SOS format of Plotkin insofar as we do not specify a transition relation over states, corresponding to elementary execution steps, but directly its transitive closure representing the possible results of a computation. It is of course possible to give an incremental SOS presentation of our system but we did not find it elegant nor useful for our purpose. Similar difficulties have been noted for the definition of SLDNF resolution (see [1]). An inductive definition of SLDNF resolution is given in [16].

where frontiers are not formed by cross products but by elementary SLD resolution steps is studied in the next section). The last rule called "pruning" (PRN) is the new inference rule introduced for negative literals. The two elements of the inferred frontier formalize the pruning by success rule (PBS) and the success by pruning rule (SBP) of the procedural interpretation respectively[3]. Note that the negation as failure rule is the restriction of the pruning rule to the case $F = \emptyset$ (if $c|A \lhd \emptyset$ then $c|\neg A \lhd \{c|\neg A,\ c|\Box\}$ by equation 4).

Definition 3.3. A computed answer constraint (c.a.c.) for a goal $c|\alpha$ is a constraint of the form $\exists Y d$ such that $c|\alpha \ \lhd\ \{d|\Box\} \cup F$ and $Y = V(d) \setminus V(c|\alpha)$. A goal $c|\alpha$ is finitely failed if $c|\alpha \ \lhd\ \emptyset$.

Example 3.4. Going back to example 3.1, the answer constraint $x = 0$ for the goal $true|\neg q(x)$ can be obtained by the following proof tree:

$$\dfrac{x=0|\Box \lhd \{x=0|\Box\} \qquad true|p(x) \lhd \{true|p(x)\}}{\dfrac{true|p(x) \lhd \{x=0|\Box,\ \ true|p(x)\}}{\dfrac{true|\neg p(x) \lhd \{x \neq 0|\neg p(x)\}}{\dfrac{true|q(x) \lhd \{x \neq 0|\neg p(x)\}}{true|\neg q(x) \lhd \{true|\neg q(x),\ \ x=0|\Box\}}}}}$$

By a simple inspection of the rules we can easily state several lemma on the goal-frontier relation $\lhd$. For some proofs we shall use the principle of structural induction on proof trees for $\lhd$, that is we shall show that a property holds for $\lhd$, simply by showing that it holds for the axiom TRIV, and for the conclusion of the rules RES, FRT and PRN assuming it holds for the premises of these rules.

Lemma 3.5. (instanciation lemma) If $c|\alpha \lhd F$ then for any constraint d there exists a frontier F' such that $c \wedge d|\alpha \lhd F'$ and $F' = d \times F$.

Proof: The proof is by structural induction on a proof tree for $c|\alpha \lhd F$.

TRIV: We have $F = \{c|\alpha : \mathcal{A} \models \exists c\}$. By rule TRIV we have also $c \wedge d|\alpha \lhd F'$ with $F' = \{c \wedge d|\alpha\ :\ \mathcal{A} \models \exists(c \wedge d)\} = d \times F$.

RES: We have $\alpha = p(X)$ and $F = \bigcup_{i \in I} F_i$ where $\{p(X) \leftarrow c_i|\alpha_i\}_{i \in I}$ is the set of renamed rules defining $p(X)$ in P such that $\mathcal{A} \models \exists(c \wedge c_i)$, and $c \wedge c_i|\alpha_i \lhd F_i$. By the induction hypothesis we get $c \wedge c_i \wedge d|\alpha_i \lhd d \times F_i$. Let $J \subseteq I$ be the subset of indices such that $c \wedge c_i \wedge d$ is $\mathcal{A}$-satisfiable, then by the RES rule we get $c \wedge d|p(X) \lhd F'$ with $F' = \bigcup_{j \in J} d \times F_j = d \times \bigcup_{i \in I} F_i = d \times F$.

FRT: We have $\alpha = \alpha_1, \alpha_2$, $c|\alpha_1 \lhd F_1$, $c|\alpha_2 \lhd F_2$ and $F = F_1 \times F_2$. By induction we get $c \wedge d|\alpha_1 \lhd d \times F_1$ and $c \wedge d|\alpha_2 \lhd d \times F_2$, hence by rule FRT and equation 1 we have $c \wedge d|\alpha \lhd d \times F$.

[3]The fact that in the procedural interpretation the SBP rule need be applied only to successful derivations in the main tree is justified at the end of this section (cf. prop. 3.12).

PRN: We have $\alpha = \neg A$ and $F = c\times\{\neg_V S|\neg A,\ \neg_V F''|\Box\}$ with $c|A\triangleleft F''$, $S \subseteq \mathcal{S}(F'')$, and $V = V(c|A)$.
By the induction hypothesis we get $c\wedge d|A\triangleleft d\times F''$, so by the PRN rule we have $c\wedge d|\neg A\triangleleft F'$ with $F' = (c\wedge d)\times\{\neg_{V'}(d\times S)|\neg A,\ \neg_{V'}(d\times F'')|\Box\}$ and $V' = V\cup V(d)$. Now

$$\begin{aligned} F' &= c\times(d\times\{\neg_{V'}d\vee\neg_V S|\neg A,\ \neg_{V'}d\vee\neg_V F''|\Box\}) \text{ by eq. 1 and 7,}\\ &= c\times\{d\wedge\neg_V S|\neg A,\ d\wedge\neg_V F''|\Box\},\\ &= d\times c\times\{\neg_V S|\neg A,\ \neg_V F''|\Box\},\\ &= d\times F. \end{aligned}$$

□

Lemma 3.6. (lifting lemma) If $c|\alpha\triangleleft F$ then there exists F' such that $true|\alpha\triangleleft F'$ and $F = c\times F'$.

PROOF: By structural induction, similarly to the proof of lemma 3.5. □

Lemma 3.7. (And-compositionality of uniform derivations)
$c|\alpha_1,\alpha_2\ \triangleleft\ F$ if and only if there exist F_1 and F_2 such that $true|\alpha_1\ \triangleleft\ F_1$, $true|\alpha_2\ \triangleleft\ F_2$, and $F = c\times F_1\times F_2$.

PROOF:

$\Rightarrow$ The proof is by cases on the root rule of a proof tree for $c|\alpha_1,\alpha_2\triangleleft F$.

TRIV: we have $F = \{c|\alpha_1,\alpha_2\ :\ \mathcal{A}\models\exists c\}$. By rule TRIV we can take $F_1 = \{true|\alpha_1\}$ and $F_2 = \{true|\alpha_2\}$, thus $F = c\times F_1\times F_2$.

RES: we have $\alpha_1 = p(X)$ and $\alpha_2 = \Box$, by lifting lemma 3.6 we get $true|p(X)\triangleleft F_1$ with $F = c\times F_1$, and by rule TRIV we can take $F_2 = \{true|\Box\}$ so that $F = c\times F_1\times F_2$.

FRT: By lifting lemma 3.6 we immediately get $F = c\times F_1\times F_2$.

PRN: same proof as for the RES case.

$\Leftarrow$ By instanciation lemma 3.5, we get $c|\alpha_1\triangleleft c\times F_1$ and $c|\alpha_2\triangleleft c\times F_2$, hence by rule FRT we have $c|\alpha_1,\alpha_2\triangleleft F$ with $F = (c\times F_1)\times(c\times F_2) = c\times F_1\times F_2$.

□

Corollary 3.8. (Canonical proof trees)[4]
Any derivation admits a canonical proof tree in which in each application of the FRT rule α_1 is a literal.

PROOF: By taking the first literal of the goal for α_1 in lemma 3.7 we can build recursively a canonical proof tree for any derivation. □

Corollary 3.9. (And-compositionality of computed answer constraints)
d is a computed answer constraint for the goal $c|A_1,...,A_m,\neg A_{m+1},...,\neg A_n$, if and only if there exists computed answer constraints $c_1,...,c_n$ for the goals $true|A_1,...,true|A_m,true|\neg A_{m+1},...,true|\neg A_n$ respectively, such that $d = c\wedge\bigwedge_{i=1}^n c_i$.

[4]Canonical proof trees will be used only in the proof of lemma 3.18.

PROOF: By n applications of the lemma. $\square$

Uniform derivations can thus be decomposed into elementary derivations, one for each literal in the query. That fundamental property does not hold for arbitrary SLD derivations, but we shall show in the next subsection that any finite SLD derivation can be extended to a uniform derivation (theorem 3.19).

Lemma 3.10. (finite failure lemma) If c is a computed answer constraint for $true|\neg p(X)$ then $c|p(X) \lhd \emptyset$. Conversely, if $c|p(X) \lhd \emptyset$ then there exists a computed answer constraint d for $true|\neg p(X)$ such that $\mathcal{A} \models c \rightarrow d$.

PROOF: First let us suppose $true|\neg p(X) \lhd F$ with $d|\square \in \mathcal{S}(F)$, $c = \exists Y d$, $Y = V(d) \setminus X$. Necessarily the PRN rule is applied at the root of a proof tree for $true|\neg p(X) \lhd F$, hence we have $true|p(X) \lhd F'$ with $F = true \times \{\neg_X S|\neg p(X),\ \neg_X F'|\square\}$ and $S \subseteq \mathcal{S}(F')$. Thus $d = \neg_X F' = c$. Hence by instanciation lemma 3.5, we have $c|p(X) \lhd c \times F'$, and $c \times F' = \neg_X F' \times F' = \emptyset$ by eq. 5.

Conversely, let us suppose $c|p(X) \lhd \emptyset$. Then by applying the PRN rule we get $c|\neg p(X) \lhd c \times \{true|\neg p(X), true|\square\}$, hence c is a c.a.c. for $c|\neg p(X)$. Therefore by corollary 3.9, there exists a c.a.c. d for $true|\neg p(X)$ such that $\mathcal{A} \models c \rightarrow d$. $\square$

In view of these lemmas, the observation of finite failure on an atom is equivalent to the observation of a success on the negation of the atom (lemma 3.10), and the computed answer constraints for a goal can be retrieved from the computed answer constraints for the unconstrained literals that appear in the goal (lifting lemma 3.6, and corollary 3.9). Therefore we can define the operational semantics of the program as the set of computed answer constraints for unconstrained literals solely.

Definition 3.11. $\mathcal{O}(P) =< \mathcal{O}^+(P), \mathcal{O}^-(P) >$

$\mathcal{O}^+(P) = \{c|p(X) \in \mathcal{B}\ :\ c \text{ is a c.a.c for the goal } true|p(X)\}$

$\mathcal{O}^-(P) = \{c|p(X) \in \mathcal{B}\ :\ c \text{ is a c.a.c. for the goal } true|\neg p(X)\}$

Note that in the procedural interpretation of the previous section the SBP rule need be applied only to the success nodes in the main tree, not to all nodes as in the PRN rule. This difference obviously does not affect successful derivations in the main tree, nor does it affect the negation of a frontier in that tree:

Proposition 3.12. (negation of frontiers obtained by the PRN rule) Let U, V be two sets of variables and S, F be two frontiers s.t. $S \subseteq F$. Then $\neg_U\{\neg_V S|\alpha, \neg_V F|\beta\} = \neg_U\{\neg_V S|\alpha\}$.

PROOF: Let $F = \{c_i|\alpha_i\}_{i\in I}$, and $S = \{c_j|\alpha_j\}_{j\in J}$ where $J \subseteq I$. For all $i \in I$ let $Y_i = V(c_i) \setminus V$ and $Z_i = (V(c_i) \setminus V) \setminus U$. We have

$$\begin{aligned} \neg_U\{\neg_V S|\alpha,\ \neg_V F|\beta\} &= \neg_U\{\textstyle\bigwedge_{j\in J} \neg\exists Y_j c_j|\alpha,\ \bigwedge_{i\in I} \neg\exists Y_i c_i|\beta\} \\ &= \textstyle\bigvee_{j\in J} \exists Z_j \exists Y_j\ c_j \wedge \bigvee_{i\in I} \exists Z_i \exists Y_i\ c_i \\ &= \textstyle\bigvee_{j\in J} \exists Z_j \exists Y_j\ c_j \\ &= \neg_U\{\neg_V S|\alpha\}. \end{aligned}$$

$\square$

The successful derivations are thus the same in the procedural interpretation and in the $\lhd$ relation. This shows that the operational semantics and the procedural interpretation are indeed equivalent w.r.t. computed answer constraints.

3.2.2. Non-uniform derivations Standard SLD trees are formed by elementary SLD derivation steps instead of cross products. Although not necessary for the rest of this paper, it is thus interesting to study the goal-frontier relation $\lhd\!\!\lhd \in \mathcal{G} \times \mathcal{P}_f(\mathcal{G})$ defined as the $\lhd$ relation except that the RES and FRT rules are replaced by the standard SLD resolution rule, and the PRN rule is generalized to conjunctive goals. The inductive definition of the $\lhd\!\!\lhd$ relation is given in figure 3.

TRIV: $c|\alpha \;\lhd\!\!\lhd\; \{c|\alpha \;:\; \mathcal{A} \models \exists c\}$

SLD: $$\frac{c \wedge c_1|\alpha, \alpha_1, \alpha' \;\lhd\!\!\lhd\; F_1 \quad \ldots \quad c \wedge c_k|\alpha, \alpha_k, \alpha' \;\lhd\!\!\lhd\; F_k}{c|\alpha, p(X), \alpha' \;\lhd\!\!\lhd\; F_1 \cup \ldots \cup F_k}$$

where $\{(p(X) \leftarrow c_i|\alpha_i)\}_{1 \leq i \leq k}$ is the set of renamed clauses defining p in P such that $\mathcal{A} \models \exists(c \wedge c_i)$, and $(V(F_i) \setminus V(c \wedge c_i|\alpha, \alpha_i, \alpha')) \cap X = \emptyset$.

PRN: $$\frac{c|A \;\lhd\!\!\lhd\; F_1 \qquad\qquad c|\alpha, \alpha' \;\lhd\!\!\lhd\; F_2}{c|\alpha, \neg A, \alpha' \;\lhd\!\!\lhd\; c \times \{\neg_V S|\neg A,\; \neg_V F_1|\Box\} \times F_2}$$

where $S \subseteq \mathcal{S}(F_1)$, $V = V(c|A)$ and $(V(F_2) \setminus V(c|\alpha, \alpha')) \cap V = \emptyset$.

Figure 3. Inductive definition of the goal-frontier relation for non-uniform derivations.

Example 3.13. Let $P = \{p(x) \leftarrow x = 0,\; p(x) \leftarrow x = 1,\; q(x, y) \leftarrow p(x), p(y)\}$. *We have the following proof tree for the goal* $true|q(x, y)$*:*

$$\dfrac{\dfrac{x = 0|p(y) \lhd\!\!\lhd \{x = 0|p(y)\} \qquad \dfrac{\begin{array}{c}x = 1 \wedge y = 0|\Box \lhd\!\!\lhd \{x = 1 \wedge y = 0|\Box\} \\ x = 1 \wedge y = 1|\Box \lhd\!\!\lhd \{x = 1 \wedge y = 1|\Box\}\end{array}}{x = 1|p(y) \lhd\!\!\lhd \{x = 1 \wedge y = 0|\Box, x = 1 \wedge y = 1|\Box\}}}{true|p(x), p(y) \lhd\!\!\lhd \{x = 0|p(y)\;,\; x = 1 \wedge y = 0|\Box,\; x = 1 \wedge y = 1|\Box\}}}{true|q(x, y) \lhd\!\!\lhd \{x = 0|p(y)\;,\; x = 1 \wedge y = 0|\Box,\; x = 1 \wedge y = 1|\Box\}}$$

hence $c = (x \neq 0 \wedge (x \neq 1 \vee (y \neq 0 \wedge y \neq 1))$ *is now a computed answer constraint for the query* $true|\neg q(x, y)$. *On the other hand we have*

$$true|q(x, y) \lhd \{x = 0|p(y)\;,\; x = 1|p(y)\},\; or$$

$$true|q(x, y) \lhd \{x = 0 \wedge y = 0|\Box,\; x = 0 \wedge y = 1|\Box,\; x = 1 \wedge y = 0|\Box,\; x = 1 \wedge y = 1|\Box\}$$

but the answer constraint c *for* $true|\neg q(x, y)$ *cannot be computed by a uniform derivation, as* $p(y)$ *cannot be developed in one branch and not in another as in a non-uniform derivation.*

Definition 3.14. *Let us define the operational semantics for non-uniform derivations as* $\tilde{\mathcal{O}}(P) =< \tilde{\mathcal{O}}^+(P), \tilde{\mathcal{O}}^-(P) >$ *where*
$\tilde{\mathcal{O}}^+(P) = \{c|p(X) \in \mathcal{B} \ : \ c$ *is a* $\lhd\!\lhd$*-c.a.c. for the goal* $true|p(X)\}$
$\tilde{\mathcal{O}}^-(P) = \{c|p(X) \in \mathcal{B} \ : \ c$ *is a* $\lhd\!\lhd$*-c.a.c. for the goal* $true|\neg p(X)\}$

Not surprisingly, one can easily check that uniform derivations can be simulated by non-uniform derivations, thus $\mathcal{O}(P) \subseteq \tilde{\mathcal{O}}(P)$.

Proposition 3.15. *If* $c|\alpha \lhd F$ *then* $c|\alpha \lhd\!\lhd F$.

Proof: By structural induction on a proof tree for $c|\alpha \lhd F$. □

Corollary 3.16. $\mathcal{O}(P) \subseteq \tilde{\mathcal{O}}(P)$.

Of course the previous example shows that the converse of that proposition doesn't hold but we can show that any non-uniform derivation can be extended to a uniform derivation, and thus that computed answers obtained by non-uniform derivations are covered by computed answers obtained by uniform derivations. For this result an extra technical lemma is needed on uniform derivations.

Definition 3.17. *Let* V *be a set of variables, and* S, T *be two sets of success goals. The (strong) covering preorder w.r.t.* V *is defined by* $S \sqsubseteq_V T$ *iff for all* $c|\Box \in S$ *there exists* $d|\Box \in T$ *s.t.* $\mathcal{A} \models \exists Y c \rightarrow \exists Z d$ *where* $Y = V(c) \setminus V$ *and* $Z = V(d) \setminus V$.

Lemma 3.18. *Let* $c|\alpha$ *be a goal and* $V = V(c|\alpha)$. *If* $c|\alpha \lhd F$ *and* $c|\alpha \lhd F'$ *then there exists* F'' *such that* $c|\alpha \lhd F''$ *with* $\mathcal{S}(F) \sqsubseteq_V \mathcal{S}(F'')$, $\mathcal{S}(F') \sqsubseteq_V \mathcal{S}(F'')$, $\mathcal{A} \models \neg_V F \rightarrow \neg_V F''$ *and* $\mathcal{A} \models \neg_V F' \rightarrow \neg_V F''$.

Proof: The proof is by structural induction on the cartesian product of canonical proof trees (cf. proposition 3.8) for $c|\alpha \lhd F$ and $c|\alpha \lhd F'$. As the rules RES, FRT and PRN are mutually exclusive there are only 5 cases.

- TRIV — We just have to take $F'' = F'$.
- — TRIV We take $F'' = F$.
- RES-RES Then $\alpha = p(X)$, let $\{p(X) \leftarrow c_k|\alpha_k\}_{k \in K}$ be the set of clauses defining p in P s.t. $c \wedge c_k$ is $\mathcal{A}$-satisfiable. We have $F = \bigcup_{k \in K} F_k$ with $c \wedge c_k|\alpha_k \lhd F_k$ for all $k \in K$, and $F' = \bigcup_{k \in K} F'_k$ with $c \wedge c_k|\alpha_k \lhd F'_k$ for all $k \in K$.
  By the induction hypothesis for all $k \in K$ there exist F''_k such that $c \wedge c_k|\alpha_k \lhd F''_k$, $\mathcal{S}(F_k) \sqsubseteq_V \mathcal{S}(F''_k)$, $\mathcal{S}(F'_k) \sqsubseteq_V \mathcal{S}(F''_k)$, $\mathcal{A} \models \neg_V F_k \rightarrow \neg_V F''_k$ and $\mathcal{A} \models \neg_V F'_k \rightarrow \neg_V F''_k$.
  Hence by the RES rule we get $c|p(X) \lhd F''$ with $F'' = \bigcup_{k \in K} F''_k$. Furthermore $\mathcal{S}(F) = \bigcup_{k \in K} \mathcal{S}(F_k) \sqsubseteq_V \mathcal{S}(F'')$, and similarly $\mathcal{S}(F') \sqsubseteq_V \mathcal{S}(F'')$. We have also $\neg_V F = \bigwedge_{k \in K} \neg_V F_k$ and $\neg_V F'' = \bigwedge_{k \in K} \neg_V F''_k$ by eq. 6, thus $\mathcal{A} \models \neg_V F \rightarrow \neg_V F''$ and similarly $\mathcal{A} \models \neg_V F' \rightarrow \neg_V F''$.

FRT-FRT As the proof trees are canonical, we have $\alpha = L, \alpha_2$, $c|L \lhd F_1$, $c|\alpha_2 \lhd F_2$, $F = F_1 \times F_2$, $c|L \lhd F'_1$, $c|\alpha_2 \lhd F'_2$, and $F' = F'_1 \times F'_2$. Now let $F'' = F''_1 \times F''_2$ where $c|L \lhd F''_1$ and $c|\alpha_2 \lhd F''_2$ are given by the induction hypothesis, by the FRT rule we have $c|\alpha \lhd F''$ and we easily check that the rest of the induction hypothesis is satisfied.

PRN-PRN Here $\alpha = \neg A$, $F = c \times \{\neg_V S_1 | \neg A,\ \neg_V F_1 | \Box\}$ with $c|A \lhd F_1$, $S_1 \subseteq \mathcal{S}(F_1)$, and $F' = c \times \{\neg_V S_2 | \neg A,\ \neg_V F_2 | \Box\}$ with $c|A \lhd F_2$, $S_2 \subseteq \mathcal{S}(F_2)$.
By the induction hypothesis there exists F''_1 such that $c|A \lhd F''_1$, $\mathcal{S}(F_1) \sqsubseteq_V \mathcal{S}(F''_1)$, $\mathcal{S}(F_2) \sqsubseteq_V \mathcal{S}(F''_1)$, $\mathcal{A} \models \neg_V F_1 \rightarrow \neg_V F''_1$ and $\mathcal{A} \models \neg_V F_2 \rightarrow \neg_V F''_1$.
Now let $F'' = c \times \{\neg_V S | \neg A,\ \neg_V F''_1 | \Box\}$ where $S = \mathcal{S}(F''_1)$. By the PRN rule we have $c|\neg A \lhd F''$, furthermore $\mathcal{S}(F'') = \{c \wedge \neg_V F''_1 | \Box\} \sqsupseteq_V \mathcal{S}(F)$, similarly $\mathcal{S}(F') \sqsubseteq_V \mathcal{S}(F'')$. Finally by proposition 3.12 we have $\neg_V F = \neg c \vee \bigvee_{s|\Box \in S_1} \exists Y_s s$ and $\neg_V F'' = \neg c \vee \bigvee_{s|\Box \in S} \exists Y_s s$ where $Y_s = V(s) \setminus V$, thus $\mathcal{A} \models \neg_V F \rightarrow \neg_V F''$ and similarly $\mathcal{A} \models \neg_V F' \rightarrow \neg_V F''$.

$\Box$

Theorem 3.19. (Extension to uniform derivations) Let G be a goal and $V = V(G)$. If $G \lhd\!\lhd F$ then there exists F' such that $G \lhd F'$, $\mathcal{S}(F) \sqsubseteq_V \mathcal{S}(F')$ and $\mathcal{A} \models \neg_V F \rightarrow \neg_V F'$.

Proof: The proof is by structural induction on a proof tree for $G \lhd\!\lhd F$.

TRIV: We take $F' = F$.

SLD: We have $G = \alpha, p(X), \alpha'$, $F = \bigcup_{k \in K} F_k$ where $\{p(X) \leftarrow c_k | \alpha_k\}_{k \in K}$ is the set of clauses defining p in P such that $\mathcal{A} \models \exists(c \wedge c_k)$, and $c \wedge c_k | \alpha, \alpha_k, \alpha' \lhd\!\lhd F_k$.
By induction for all $k \in K$ there exist F'_k such that $c \wedge c_k | \alpha, \alpha_k, \alpha' \lhd F'_k$, $\mathcal{S}(F_k) \sqsubseteq \mathcal{S}(F'_k)$ and $\mathcal{A} \models \neg F_k \rightarrow \neg F'_k$.
By lemmas 3.7 and 3.5, for all $k \in K$ there exist F''_k and F'''_k such that $c \wedge c_k | \alpha_k \lhd F''_k$, $c|\alpha, \alpha' \lhd F'''_k$ and $F'_k = F''_k \times F'''_k$.
Hence by the RES rule we get

$$c|p(X) \lhd \bigcup_{k \in K} F''_k.$$

Furthermore by lemma 3.18 there exists F''' such that

$$c|\alpha, \alpha' \lhd F'''$$

and for all $k \in K$ $\mathcal{S}(F'''_k) \sqsubseteq_V \mathcal{S}(F''')$ and $\mathcal{A} \models \neg_V F'''_k \rightarrow \neg_V F'''$. Let $F' = \bigcup_{k \in K} F''_k \times F'''$, by the FRT rule we get

$$c|\alpha, p(X), \alpha' \lhd F'$$

Now $\mathcal{S}(F)$ $= \bigcup_{k \in K} \mathcal{S}(F_k)$
$\sqsubseteq_V \bigcup_{k \in K} \mathcal{S}(F'_k)$
$\sqsubseteq_V \bigcup_{k \in K} \mathcal{S}(F''_k) \times \mathcal{S}(F'''_k)$ by eq. 8
$\sqsubseteq_V \bigcup_{k \in K} \mathcal{S}(F''_k) \times \mathcal{S}(F''')$
$\sqsubseteq_V \mathcal{S}(F')$ by eq. 8.

Furthermore $\neg_V F = \bigwedge_{k \in K} \neg_V F_k$ by eq. 6,
thus $\mathcal{A} \models \neg_V F \rightarrow \bigwedge_{k \in K} \neg_V F''k \vee \neg_V F'''k$ by eq. 7
$\mathcal{A} \models \neg_V F \rightarrow \bigwedge_{k \in K} \neg_V F''k \vee \neg_V F'''$
$\mathcal{A} \models \neg_V F \rightarrow \neg_V F'$ by eq. 6 and 7.

PRN: We have $G = c|\alpha, \neg A, \alpha'$, $F = c \times \{\neg_V S|\neg A,\ \neg_V F_1|\Box\} \times F_2$ with $S \subseteq \mathcal{S}(F_1)$, $c|A \lhd\!\lhd F_1$ and $c|\alpha, \alpha' \lhd\!\lhd F_2$. By induction there exist F_1' and F_2' such that $c|A \lhd F_1'$, $c|\alpha, \alpha' \lhd F_2'$, $\mathcal{S}(F_1) \sqsubseteq_V \mathcal{S}(F_1')$, $\mathcal{S}(F_2) \sqsubseteq_V \mathcal{S}(F_2')$, $\mathcal{A} \models \neg_V F_1 \rightarrow \neg_V F_1'$ and $\mathcal{A} \models \neg_V F_2 \rightarrow \neg_V F_2'$. Let $F' = c \times \{\neg_V \mathcal{S}(F_1')|\neg A,\ \neg_V F_1'|\Box\} \times F_2'$, by the PRN rule we have $G \lhd F'$ and we easily check that $\mathcal{S}(F) \sqsubseteq_V \mathcal{S}(F')$ and $\mathcal{A} \models \neg_V F \rightarrow \neg_V F'$ by proposition 3.12.

□

Corollary 3.20. $\tilde{\mathcal{O}}(P) \sqsubseteq \mathcal{O}(P)$.

4. Fully abstract fixpoint semantics

In this section we study a non-ground continuous finitary version of Fitting's operator for constraint logic programs. We show that the least fixed point of our operator is *equal* to the operational semantics of constructive negation by pruning.

We first recall the definition of Fitting's operator $\Phi_P^{\mathcal{A}}$. Given a $\Pi - \Sigma$-algebra $\mathcal{A}$, the $\mathcal{A}$-ground base $\mathcal{B}_A = [\mathcal{B}]_{\mathcal{A}}$ is the set of ground instances of the base $\mathcal{B}$ of constrained atoms. A *partial ground interpretation* is a couple $I = <I^+, I^->$ such that $I^+, I^- \subseteq \mathcal{B}_A$ and $I^+ \cap I^- = \emptyset$. A ground atom A is true (resp. false) in I iff $A \in I^+$ (resp. $A \in I^-$). A constraint is true in I if it is true in $\mathcal{A}$. A first-order $\Sigma - \Pi$-formula Ψ is true in I, noted $I \models_3 \Psi$, if it is true under the usual strong three-valued interpretation of the logical symbols. The set of partial ground interpretations forms a semi-lattice for set inclusion on true and false atoms, we denote it by $(\mathcal{G}I, \subseteq_3)$.

Definition 4.1 ([23][16][11]). Let P be a normal $CLP(\mathcal{A})$ program, the immediate consequence operator $\Phi_P^{\mathcal{A}} : \mathcal{G}I \rightarrow \mathcal{G}I$ is defined by:

$$\begin{aligned}
\Phi_P^{\mathcal{A}+}(I) = \ & \{A \in \mathcal{B}_A \mid \textit{there exist a clause in } P,\ p(X) \leftarrow c|\alpha, \\
& \textit{and a valuation } \theta \textit{ such that } A = p(X)\theta \textit{ and } I \models_3 (c \wedge \alpha)\theta\} \\
\Phi_P^{\mathcal{A}-}(I) = \ & \{A \in \mathcal{B}_A \mid \textit{for any clause in } P,\ p(X) \leftarrow c|\alpha, \\
& \textit{and any valuation } \theta \textit{ such that } A = p(X)\theta \textit{ then } I \models_3 \neg(c \wedge \alpha)\theta\}.
\end{aligned}$$

$\Phi_P^{\mathcal{A}}$ is a monotonic operator in the semi-lattice of partial ground interpretations. It thus admits a least fixpoint which is the least three-valued $\mathcal{A}$-model of the program's completion [11]. It is not continuous however, so its power at ordinal ω is generally not a fixpoint (cf. example 4.3).

In order to abstract from a given algebra $\mathcal{A}$ and to prove completeness results, Stuckey [23] defined a non-ground version of Fitting's operator based on partial constrained interpretations. In his definition the downward closure of constrained atoms by their instances prevents however a characterization of the operational behavior of the program w.r.t. answer constraints. Furthermore the operator of Stuckey is not continuous either, so it doesn't provide CLP programs with a fixpoint semantics.

The idea of our operator T_P for obtaining a fully abstract fixpoint semantics is simply to take the finitary, hence continuous, non-downward closed constraint based version of Fitting's operator. So a constrained atom will be true (resp. false) in $T_P(I)$ if the constraint in the constrained atom is a combination of constraints in a finite part of I which validates the body of a program clause for the atom (resp. invalidates the body of all program clauses for the atom).

Definition 4.2. Let P be a $CLP(\mathcal{A})$ program. T_P is an operator over $2^{\mathcal{B}} \times 2^{\mathcal{B}}$ defined by $T_P(I) = < T_P^+(I), T_P^-(I) >$ where:

$T_P^+(I) = \{c|p(X) \in \mathcal{B}$ *: there exists a clause in P with local variables Y,*
$p(X) \leftarrow d|A_1, ..., A_m, \neg A_{m+1}, ..., \neg A_n$
for all $1 \le i \le m$ there exists $c_i|A_i \in I^+$
for all $m+1 \le j \le n$ there exists $c_j|A_j \in I^-$
such that $c = \exists Y\ d \wedge \bigwedge_{i=1}^{n} c_i$ is $\mathcal{A}$-satisfiable$\}$

$T_P^-(I) = \{c|p(X) \in \mathcal{B}$ *: for each clause defining p in P with local variable Y_k,*
$p(X) \leftarrow d_k|A_{k,1}, ..., A_{k,m_k}, \alpha_k$*, where $m_k \ge 0$,*
for all $1 \le i \le m_k$ there exists $e_{k,i}|A_{k,i} \in I^-$,
there exists $n_k \ge m_k$ and for all $m_{k+1} \le j \le n_k$
there exists $e_{k,j}|A_{k,j} \in I^+$ with $\neg A_{k,j}$ occurring in α_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$\}$

Note that in the definition of T_P^+, for each literal in the body of a program clause defining p, exactly one constrained atom is taken in I. In the definition of T_P^-, if p is not defined in P then we have $c = true$, otherwise for each clause defining p, a finite number of constrained atoms are taken in I to invalidate the body of the clause. Note that for each atom in the body at most one constrained atom is taken in I^- (α_k may contain both positive and negative literals), whereas for each negative literal a finite number of constrained atoms can be taken in I^+. This is crucial for the completeness w.r.t. the logical semantics. For instance, with the program P:

$p \leftarrow \neg q(X).$
$q(X) \leftarrow X \ge 0.$
$q(X) \leftarrow X < 0.$

We have $T_P^+(\emptyset) = \{X \ge 0|q(X),\ X < 0|q(X)\}$ and $T_P^-(T_P(\emptyset)) = \{true|p\}$. If in the definition of T_P^- only one constrained atom was taken in I^+ for a negative literal in the clause, then p would not be false in the iteration of T_P. Allowing to take similarly a finite number of constrained atoms in I^- for a same positive literal, instead of at most one, would not change the definition of T^- as we shall see that the finite powers of T_P are closed by disjunction on false atoms (proposition 5.6).

Example 4.3. Let us consider an example over the Herbrand domain formed with a constant 0 and an unary function symbols s. Clark's equational theory CET augmented with the domain closure axiom DCA is a complete theory for that structure [19], in particular we have $CET + DCA \models (\forall y\ x \ne s(y)) \leftrightarrow x = 0$. The following program is a classical example that shows that Fitting's operator $\Phi_P^{\mathcal{A}}$ is not continuous:

$p(x) \leftarrow x = s(y)|p(y).$
$q \leftarrow p(x).$

No atom is true in the powers of $\Phi_P^{\mathcal{A}}$ and T_P. At ordinal ω, all ground instances of $p(x)$ are false both in $\Phi_P^{\mathcal{A}} \uparrow \omega$ and $[T_P \uparrow \omega]$, whereas the atom q becomes false *in* $\Phi_P^{\mathcal{A}} \uparrow \omega + 1$ *and stays undefined in* $[T_P \uparrow \omega + 1]$*:*

α	$(\Phi_P^{\mathcal{A}} \uparrow \alpha)^-$	$(T_P \uparrow \alpha)^-$
0	$\emptyset$	$\emptyset$
1	$\{p(0)\}$	$\{x = 0 \mid p(x)\}$
2	$\{p(0), p(s(0))\}$	$\{x = 0 \mid p(x), x = 0 \vee x = s(0) \mid p(x)\}$
...	...	...
ω	$\{p(s^i(0) \mid i \geq 0\}$	$\{x = 0 \vee ... \vee x = s^i(0) \mid p(X) \mid i \geq 0\}$
$\omega + 1$	$\{q\} \cup \{p(s^i(0) \mid i \geq 0\}$	$\{x = 0 \vee ... \vee x = s^i(0) \mid p(X) \mid i \geq 0\}$

The definition of $\Phi_P^{\mathcal{A}}(I)$ *based on valuations allows to infer that* q *is false in* $\Phi_P^{\mathcal{A}} \uparrow \omega + 1$ *whilst the definition of* T_P *based on finite subsets of* I *does not.*

Proposition 4.4. T_P *is an operator over partial interpretations.*

Proof: We just have to prove that if I is a partial interpretation, then $[T_P^+(I)] \cap [T_P^-(I)] = \emptyset$.
Let $c|p(X) \in T_P^+(I)$ and θ be any valuation of the variables in X such that $c\theta$ is true. There exists a clause in P, $p(X) \leftarrow d|A_1, ..., A_m, \neg A_{m+1}, ..., \neg A_n$, such that for all $1 \leq i \leq m$ there exists $c_i|A_i \in I^+$, for all $m+1 \leq j \leq n$ there exists $c_j|A_j \in I^-$, such that $c = \exists Y(d \wedge \bigwedge_{i=1}^n c_i)$. As $c\theta$ is true in $\mathcal{A}$ let ρ be a valuation extending θ to the variables in Y such that $(d \wedge \bigwedge_{i=1}^n c_i)\rho$ is true.

Let us suppose that there exists $e|p(X) \in T_P^-(I)$ such that $e\theta$ is true. Then for the previous clause defining p there exists $p \leq m$ and $\{e_1|A_1, ..., e_p|A_p\} \subseteq I^-$, there exists $\{e_{p+1}|B_{p+1}, ..., e_q|B_q\} \subseteq I^+$, where for all $p+1 \leq j \leq q$ $B_j \in \{A_{m+1}, ..., A_n\}$, such that $e = \forall Y(\neg d \vee \bigvee_{j=1}^q e_j)$ is satisfied by θ. Hence $(\neg d \vee \bigvee_{j=1}^q e_j)\rho$ is true.

Now as $d\rho$ is true, $e_j\rho$ must be true for some $j \in [1, q]$, with $e_j|A_i \in I^-$ for some $i \in [1, m]$ (or $e_j|A_i \in I^+$ for some $i \in [m+1, n]$). Hence we have $c_i|A_i \in I^+$ (or $c_i|A_i \in I^-$) with $c_i\rho$ true, so we get a contradiction: $c_i \wedge e_j$ is satisfied by ρ and we have $c_i|A_i \in I^+$ and $e_j|A_i \in I^-$ (or $c_i|A_i \in I^-$ and $e_j|A_i \in I^+$), i.e. I is not a partial interpretation. □

Proposition 4.5. T_P *is monotonic in the semi-lattice* $(\mathcal{I}, \subseteq_3)$.

Proof: If $I \subseteq_3 J$ then $I^+ \subseteq J^+$ and $I^- \subseteq J^-$, so it is straightforward to verify that by definition of T_P we have both $T_P^+(I) \subseteq T_P^+(J)$ and $T_P^-(I) \subseteq T_P^-(J)$, thus $T_P(I) \subseteq_3 T_P(J)$. □

Proposition 4.6. T_P *is continuous in the semi-lattice* $(\mathcal{I}, \subseteq_3)$.

Proof: The result follows from the fact that an operator f over a powerset, monotonic w.r.t. set inclusion, is continuous if it is finitary, i.e. $\forall x, y \; x \in f(y) \Rightarrow \exists y' \subseteq y$ finite s.t. $x \in f(y')$. From its definition T_P is clearly finitary. □

As T_P is continuous we can take the least fixpoint of T_P as the fixpoint semantics of the program. We then show a strong equivalence theorem with the operational semantics which shows that the fixpoint semantics fully characterizes the operational behavior of normal CLP programs w.r.t. answer constraints.

Definition 4.7. (Fixpoint semantics) $\mathcal{F}(P) = lfp(T_P) = T_P \uparrow \omega$.

Main theorem 4.8. (Full abstraction for answer constraints computed by uniform derivations) $\mathcal{O}(P) = \mathcal{F}(P)$.

PROOF:

$\subseteq_3$: We show more generally that if $c|\alpha \lhd F$ where $\alpha \neq \Box$ then, let $V = V(c|\alpha)$,

1) if $d|\Box \in F$ then for each occurrence of an atom A_i in α, $1 \leq i \leq m$, there exists $c_i|A_i \in \mathcal{F}(P)^+$, for each occurrence of a negative literal $\neg A_j$ in α, $m+1 \leq j \leq n$, there exists $c_j|A_j \in \mathcal{F}(P)^-$, such that $\exists Y d = c \wedge \bigwedge_{i=1}^{n} c_i$ where $Y = V(d) \setminus V$.
2) if $\neg_V F$ is $\mathcal{A}$-satisfiable then there exist occurrences of atoms of atoms in α, $A_1, ..., A_m$, $m \geq 0$, and for all $1 \leq i \leq m$ there exists $c_i|A_i \in \mathcal{F}^-$, there exists $n \geq m$ and for all $m+1 \leq j \leq n$ there exists $c_j|A_j \in \mathcal{F}^+$ where $\neg A_j$ is a negative literal in α, such that $\neg_V F = \neg c \vee \bigvee_{i=1}^{n} c_i$.

Therefore taking $c|\alpha = true|p(x)$ in 1) we get $\mathcal{O}(P)^+ \subseteq \mathcal{F}(P)^+$, and taking $c|\alpha = true|\neg p(x)$ in 1) we get $\mathcal{O}(P)^- \subseteq \mathcal{F}(P)^-$. The proof is by structural induction on a proof tree for $c|\alpha \lhd F$.

TRIV: 1) We have $F = \{c|\alpha \ : \ \mathcal{A} \models \exists c\}$ and $\alpha \neq \Box$ hence $\{d|\Box\} \notin F$.

2) We have $F = \{c|\alpha \ : \ \mathcal{A} \models \exists c\}$ and $\neg_V F = \neg c$.

RES: We have $\alpha = p(X)$ and $F = \bigcup_{i=1}^{k} F_i$ where $\{p(X) \leftarrow c_i|\alpha_i\}_{1 \leq i \leq k}$ is the set of renamed apart clauses defining $p(X)$ in P with local variables Y_i such that $c \wedge c_i$ is $\mathcal{A}$-satisfiable, and $c \wedge c_i|\alpha_i \lhd F_i$ for all $1 \leq i \leq k$.

1) Let us suppose $d|\Box \in F$, then $d|\Box \in F_i$ for some i. By the induction hypothesis applied to $c \wedge c_i|\alpha_i \lhd F_i$ we get that for each atom's occurrence A_j (resp. negative literal's occurrence $\neg A_j$) in α_i, $1 \leq j \leq n$, there exists $e_j|A_j \in \mathcal{F}(P)^+$ (resp. $e_j|A_j \in \mathcal{F}(P)^-$) such that $\exists Z d = c \wedge c_i \wedge \bigwedge_{j=1}^{n} e_j$ where $Z = V(d) \setminus V(c \wedge c_i|\alpha_i)$.
Now let $e = \exists Y_i(c_i \wedge \bigwedge_{j=1}^{n} e_j)$, then by the definition of T_P^+ we get that $e|p(X) \in \mathcal{F}(P)^+$, thus $\exists Y d = \exists Y_i \exists Z d = c \wedge e$.

2) Let us suppose $\neg_V F$ is $\mathcal{A}$-satisfiable. As $\neg_V F = \bigwedge_{i=1}^{k} \neg_V F_i$ by eq. 6, $\neg_V F_i$ is $\mathcal{A}$-satisfiable for all $1 \leq i \leq k$. Let $V_i = V(c \wedge c_i|\alpha_i)$, we have $\neg_V F = \bigwedge_{i=1}^{k} \forall Y_i \neg_{V_i} F_i$, hence for all $1 \leq i \leq k$, $\neg_{V_i} F_i$ is $\mathcal{A}$-satisfiable as well. By applying the induction hypothesis to $c \wedge c_i|\alpha_i \lhd F_i$, we get that for all $1 \leq i \leq k$,

$$\neg_{V_i} F_i = \neg c \vee \neg c_i \vee \bigvee_{j=1}^{n_i} c_j^i$$

where $n_i \geq 0$ and for all $1 \leq j \leq n_i$, $c_j^i|A_j^i \in \mathcal{F}^-$ (resp. $c_j^i|A_j^i \in \mathcal{F}^+$) where A_j^i is an atom's occurrence in α_i (resp. $\neg A_j^i$ is a literal in α_i).
Now let $d = \bigwedge_{i=1}^{k} \forall Y_i(\neg c_i \vee \bigvee_{j=1}^{n_i} c_j^i)$, we get from the definition of T_P^- that $d|p(X) \in \mathcal{F}(P)^-$. Thus $\neg_V F = \bigwedge_{i=1}^{k} \forall Y_i \neg_{V_i} F_i = \neg c \vee d$.

FRT: We have $\alpha = \alpha_1, \alpha_2$, $c|\alpha_1 \lhd F_1$, $c|\alpha_2 \lhd F_2$ and $F = F_1 \times F_2$. Let $V_1 = V(c|\alpha_1)$ and $V_2 = V(c|\alpha_2)$.

1) If $d|\Box \in F$ then there exist $d_1|\Box \in F_1$ and $d_2|\Box \in F_2$ such that $d = d_1 \wedge d_2$. Let $Y = V(d) \setminus V(c|\alpha)$, $Y_1 = V(d_1) \setminus V(c|\alpha_1)$ and $Y_2 = V(d_2) \setminus V(c|\alpha_2)$. By the hypothesis on variable clashes in the FRT rule we have $\exists Y d_1 = \exists Y_1 d_1$ and $\exists Y d_2 = \exists Y_2 d_2$, therefore $\exists Y d = \exists Y_1 d_1 \wedge \exists Y_2 d_2$. Now the induction hypothesis 1) applied to $c|\alpha_1 \lhd F_1$ and $c|\alpha_2 \lhd F_2$ immediately concludes the proof.

2) By eq. 7 we have $\neg_V F = \neg_V F_1 \vee \neg_V F_2$. Furthermore by the hypothesis on variable clashes in the FRT rule we have $\neg_{V_1} F_1 = \neg_V F_1$ and $\neg_{V_2} F_2 = \neg_V F_2$, therefore $\neg_V F = \neg_{V_1} F_1 \vee \neg_{V_2} F_2$. Now the induction hypothesis 2) applied to $c|\alpha_1 \lhd F_1$ and $c|\alpha_2 \lhd F_2$ also immediately concludes the proof.

PRN: We have $\alpha = \neg p(X)$ and $F = c \times \{\neg_V S|\neg p(X),\ \neg_V F'|\Box\}$ with $c|p(X) \lhd F'$ and $S \subseteq \mathcal{S}(F')$.

1) If $d = c \wedge \neg_V F'$ is $\mathcal{A}$-satisfiable, then $\neg_V F'$ is $\mathcal{A}$-satisfiable, hence by the induction hypothesis 2) applied to $c|p(X) \lhd F'$ we get that there exists $c_1|p(X) \in \mathcal{F}(P)^-$ such that $\neg_V F' = \neg c \vee c_1$. Thus $\exists Y d = c \wedge (\neg c \vee c_1) = c \wedge c_1$.

2) Let us suppose that $\neg_V F$ is $\mathcal{A}$-satisfiable. Let $S = \{s_1|\Box, ..., s_m|\Box\}$, proposition 3.12 gives $\neg_V F = \neg c \vee \bigvee_{i=1}^m \exists Y_i s_i$ where $Y_i = V(s_i) \setminus V$. Now by the induction hypothesis 1) applied to $c|p(X) \lhd F'$ we get that there exists $\{c_1|p(X), ..., c_m|p(X)\} \subseteq \mathcal{F}(P)^+$ such that for all $1 \leq i \leq m$, $\exists Y_i s_i = c_i$. Thus $\neg_V F = \neg c \vee \bigvee_{i=1}^m c_i$ with $c_i|p(X) \in \mathcal{F}(P)^+$.

$\supseteq_3$: We prove by induction on n that $\mathcal{O}^+(P) \supseteq T_P \uparrow n^+$ and $\mathcal{O}^-(P) \supseteq T_P \uparrow n^-$ for all $n \geq 0$. The base case $n = 0$ is trivial. Let us consider the induction step.

Let $c|p(X) \in (T_P \uparrow n)^+$. There exists a clause with local variables Y

$$p(X) \leftarrow d|A_1, ..., A_m, \neg A_{m+1}, ..., \neg A_n$$

for all $1 \leq i \leq m$ there exists $c_i|A_i \in (T_P \uparrow n-1)^+$ for all $m+1 \leq j \leq n$ there exists $c_j|A_j \in (T_P \uparrow n-1)^-$ such that $c = \exists Y d \wedge \bigwedge_{i=1}^n c_i$.
By induction we get that c_i is a computed answer to the goal $true|A_i$ for all $1 \leq i \leq m$ and to the goal $true|\neg A_i$ for all $m+1 \leq i \leq n$.
Hence by corollary 3.9, $\bigwedge_{i=1}^n c_i$ is a computed answer to

$$true|A_1, ..., A_m, \neg A_{m+1}, ..., \neg A_n.$$

By the instanciation lemma 3.5, we get that c is a computed answer to the goal $d|A_1, ..., A_m, \neg A_{m+1}, ..., \neg A_n$, hence by the RES rule we get $c|p(X) \in \mathcal{O}^+(P)$.

Let $c|p(X) \in (T_P \uparrow n)^-$. For any clause defining p in P, with local variables Y_k,

$$p(X) \leftarrow d_k|A_{k,1}, ..., A_{k,m_k}, \alpha_k$$

there exists $\{e_{k,1}|A_{k,1},...,e_{k,m_k}|A_{k,m_k}\} \subseteq (T_P \uparrow n-1)^-$,
there exists $n_k \geq m_k$ and $\{e_{k,m_k+1}|A_{k,m_k+1},...,e_{k,n_k}|A_{k,n_k}\} \subseteq (T_P \uparrow n-1)^+$, where for all $m_k+1 \leq j \leq n_k$, $\neg A_{k,j}$ is a negative literal in α_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.

By induction we have that for all $1 \leq i \leq m_k$, $e_{k,i}$ is a c.a.c. for the goal $true|\neg A_{k,i}$. As the PRN rule is necessarily applied at the root, we have

$$true|A_{k,i} \lhd F_{k,i}$$

with $e_{k,i} = \neg_{V(A_{k,i})} F_{k,i}$.
Similarly by induction we have that for all $m_{k+1} \leq j \leq n_k$, $e_{k,j}$ is c.a.c. for the goal $true|A_{k,j}$. Hence by the PRN rule, taking the singleton $\{e_{k,j}|\Box\}$ as success set, we have

$$true|\neg A_{k,j} \lhd F_{k,j}$$

with $\neg e_{k,j}|\neg A_{k,j} \in F_{k,j}$. By proposition 3.12 we get $\neg_{V(A_{k,j})} F_{k,j} = e_{k,j}$.
Let $\beta_k = \alpha_k \setminus \bigcup_{j=m_k+1}^{n_k} \neg A_{k,j}$, by lemma 3.7 and rule TRIV we have

$$d_k|A_{k,1},...,A_{k,m_k},\neg A_{k,m_k+1},...,\neg A_{k,n_k},\beta_k \lhd F_k$$

where $F_k = d_k \times \times_{i=1}^{n_k} F_{k,i} \times \{d_k|\beta_k\}$. Note that by equation 7 we have $\neg_X F_k = \forall Y_k(\neg d_k \vee \bigvee_{i=1}^{n_k} e_{k,i}) = c_k$.
Now by applying the RES rule we get $true|p(X) \lhd F$ where $F = \bigcup_{k\in K} F_k$. By equation 6 we have $\neg_X F = \bigwedge_k c_k = c$, hence by the PRN rule we get $c|p(X) \in \mathcal{O}^-(P)$.

$\Box$

Full abstraction does not hold for non uniform derivations, however the full abstraction theorem 4.8, together with corollaries 3.16 and 3.20 show that non-uniform derivations are nevertheless sound and complete w.r.t. the fixpoint semantics.

Theorem 4.9. (Soundness and completeness of non-uniform derivations) $\tilde{\mathcal{O}}(P) \sqsubseteq \mathcal{F}(P)$ *and* $\mathcal{F}(P) \subseteq \tilde{\mathcal{O}}(P)$.

5. Three-valued logical semantics

The main theorem of [15], extended to CLP programs in [23], characterizes the three-valued logical consequences of the Clark's completion with the finite powers of Fitting's operator $\Phi_P^{\mathcal{A}}$:

Theorem 5.1 ([15][23]). Let P *be a normal* $CLP(\mathcal{A})$ *program and* ψ *be a* Π, Σ, V*-formula, then* $P^*, th(\mathcal{A}) \models_3 \phi$ *iff* ψ *is true in* $\Phi_P^{\mathcal{A}} \uparrow n$ *for some integer* n.

Corollary 5.2. $[\mathcal{L}(P)]_{\mathcal{A}} = \Phi_P^{\mathcal{A}} \uparrow \omega$.

In this section we show that the finite powers of T_P coincide with those of Fitting's operator $\Phi_P^{\mathcal{A}}$ as in [23], and thus, by the previous theorem, that the fixpoint and operational semantics are correct and complete w.r.t. the three-valued logical consequences of the program's completion.

Proposition 5.3. If I is a finite partial interpretation then $T_P(I)$ is finite.

PROOF: Obvious from the definition of T_P. □

Corollary 5.4. For all $n \geq 0$, $T_P \uparrow n$ is finite.

Definition 5.5. A constrained interpretation I is closed by disjunction if whenever $c|p(X) \in I$, $c'|p(X) \in I$ then there exists $d|p(X) \in I$ such that $\mathcal{A} \models (c \vee c') \rightarrow d$.

Proposition 5.6. Let I be a partial constrained interpretation. If I^- is closed by disjunction then so is $T_P^-(I)$.

PROOF: Let $c|p(X), c'|p(X) \in T_P^-(I)$. 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 $m_k \geq 0$,
there exist $\{e_{k,r}|A_{k,r}\}_{r\in R} \subseteq I^-$, and $\{e'_{k,r'}|A_{k,r'}\}_{r'\in R'} \subseteq I^-$, where R and R' are subsets of $\{1, ..., m_k\}$,
there exist finite sets $\{e_{k,s}|A_{k,s}\}_{s\in S} \subseteq I^+$ and $\{e'_{k,s'}|A_{k,s'}\}_{s'\in S'} \subseteq I^+$ where $R \cap S = \emptyset$, $R' \cap S' = \emptyset$ and for all $j \in S \cup S'$, $\neg A_{k,j}$ is a negative literal in α_k,
such that $c_k = \forall Y_k(\neg d_k \vee \bigvee_{i\in R\cup S} e_{k,i})$, $c'_k = \forall Y_k(\neg d_k \vee \bigvee_{j\in R'\cup S'} e'_{k,j})$ $c = \bigwedge_k c_k$, and $c' = \bigwedge_k c'_k$ are $\mathcal{A}$-satisfiable ($c = c' = true$ if p is not defined in P)$\}$.

Now for any clause defining p in P, let

$$\{f_{k,l}\}_{l\in L} = \{e_{k,s}\}_{s\in S} \cup \{e'_{k,s'}\}_{s'\in S'} \cup \{e_{k,r}\}_{r\in R\setminus R'} \cup \{e'_{k,r'}\}_{r'\in R'\setminus R} \cup \{g_{k,r}\}_{r\in R\cap R'}$$

where, as I is a partial interpretation closed by disjunction on false atoms, we can define $g_{k,r}$ for all $r \in R \cap R'$ by choosing $g_{k,r}|A_{k,r} \in I^-$ such that $\mathcal{A} \models e_{k,r} \vee e'_{k,r} \rightarrow g_{k,r}$.

Let $f_k = \forall Y_k(\neg d_k \vee \bigvee_{l\in L} f_{k,l})$, we have $\mathcal{A} \models (c_k \vee c'_k) \rightarrow f_k$, hence f_k is $\mathcal{A}$-satisfiable. Let $f = \bigwedge_k f_k$, we have $\mathcal{A} \models c \vee c' \rightarrow f$, hence f is $\mathcal{A}$-satisfiable. Therefore by definition of T_P^- we conclude that $f|p(X) \in T_P^-(I)$ with $\mathcal{A} \models c \vee c' \rightarrow f$.
□

Corollary 5.7. For all $n \geq 0$, $T_P \uparrow n$ is closed by disjunction on false atoms.

Corollary 5.8. *$\mathcal{F}(P)$ (and $\mathcal{O}(P)$) are closed by disjunction on false constrained atoms.*

Lemma 5.9. *$[T_P(I)]_{\mathcal{A}} = \Phi_P^{\mathcal{A}}([I]_{\mathcal{A}})$ for all finite partial interpretation closed I by disjunction on false atoms.*

PROOF: We consider both inclusions on positive and negative parts separately.

$\subseteq^+$: Let $c|p(X) \in T_P^+(I)$, and θ be any $\mathcal{A}$-valuation of X such that $c\theta$ is true. From the definition of T_P^+ there exists a clause in P with local variables Y,

$$p(X) \leftarrow d|A_1, ..., A_m, \neg A_{m+1}, ..., \neg A_n$$

such that for all $1 \leq i \leq m$ there exists $c_i|A_i \in I^+$,, for all $m+1 \leq j \leq n$ there exists $d_j|A_j \in I^-$ such that $c = \exists Y(d \wedge \bigwedge_i c_i \wedge \bigwedge_j d_j)$ is $\mathcal{A}$-satisfiable. Therefore there exists an $\mathcal{A}$-valuation ρ which extends θ to an $\mathcal{A}$-valuation of the variables in Y such that $(d \wedge \bigwedge_i c_i \wedge \bigwedge_j d_j)\rho$ is true. Hence $d\rho$ is true, $A_i\rho \in [I^+]_{\mathcal{A}}$ for all i, $1 \leq i \leq m$, and $A_j\rho \in [I^-]_{\mathcal{A}}$ for all j, $m+1 \leq j \leq n$. Hence by definition of $(\Phi_P^{\mathcal{A}})^+$ we have $p(X)\theta \in \Phi_P^{\mathcal{A}+}([I]_{\mathcal{A}})$ for all θ such that $c\theta$ is true.

$\subseteq^-$: Let $c|p(X) \in T_P^-(I)$, and θ be any $\mathcal{A}$-valuation of X such that $c\theta$ is true.
For any clause in P defining p, with local variable Y_k,
$p(X) \leftarrow d_k|A_{k,1}, ..., A_{k,m}, \alpha_k$,
there exists $\{e_{k,1}|A_{k,1}, ..., e_{k,m}|A_{k,m}\} \subseteq I^-$,
there exists $\{e_{k,m+1}|A_{k,m+1}, ..., e_{k,n}|A_{k,n}\} \subseteq I^+$, where for all $m+1 \leq j \leq n$, $\neg A_{k,j}$ is a negative literal in α_k,
such that $c_k = \forall Y_k(\neg d_k \vee \bigvee_{i=1}^n e_{k,i})$ is $\mathcal{A}$-satisfiable, and $c = \bigwedge_k c_k$ is $\mathcal{A}$-satisfiable.
Therefore for any clause defining p in P, and any $\mathcal{A}$-valuation ρ_k extending θ to an $\mathcal{A}$-valuation for the variables in Y_k, we have
either $d_k\rho_k$ false,
or $e_{k,i}\rho_k$ true for some $1 \leq i \leq m$, in which case $A_i\rho_k \in [I^-]_{\mathcal{A}}$,
or $e_{k,j}\rho_k$ true for some $m+1 \leq j \leq n$, in which case $A_j\rho_k \in [I^+]_{\mathcal{A}}$.
Hence by definition of $(\Phi_P^{\mathcal{A}})^-$ we have $p(X)\theta \in \Phi_P^{\mathcal{A}-}([I]_{\mathcal{A}})$ for all θ such that $c\theta$ is true.

$\supseteq^+$: Let $p(X)\theta \in \Phi_P^{\mathcal{A}+}([I]_{\mathcal{A}})$. There exists a clause in P with local variables Y, $p(X) \leftarrow d|A_1, ..., A_m, \neg A_{m+1}, ..., \neg A_n$ such that $d\theta$ is true and for all $1 \leq i \leq m$, $m+1 \leq j \leq n$, $A_i\theta \in [I^+]_{\mathcal{A}}$ and $A_j\theta \in [I^-]_{\mathcal{A}}$.
Hence for all $1 \leq i \leq m$, $m+1 \leq j \leq n$, there exist $c_i|A_i \in I^+$, $d_j|A_j \in I^-$, such that $c_i\theta$ and $d_j\theta$ are true. Hence $c = \exists Y(d \wedge \bigwedge_i c_i \wedge \bigwedge_j d_j)$ is $\mathcal{A}$-satisfiable (by θ), and from the definition of T_P^+ we get $c|p(X) \in T_P^+(I)$.

$\supseteq^-$: Let $p(X)\theta \in \Phi_P^{\mathcal{A}-}([I]_{\mathcal{A}})$. From the definition of $\Phi_P^{\mathcal{A}}$, for any clause in P defining p, with local variable Y_k, $p(X) \leftarrow d_k|\alpha_k$, and for any $\mathcal{A}$-valuation θ_k extending θ to the variables in Y_k, we have
either $d_k\theta_k$ false,
or $A\theta_k \in [I^+]_{\mathcal{A}}$ for some negative literal $\neg A$ in α_k, in which case there exists $c|A \in I^+$ with $c\theta_k$ true,

or $A'\theta_k \in [I^-]_\mathcal{A}$ for some positive literal A' in α_k, in which case there exists $c'|A' \in I^-$ with $c'\theta_k$ true.
Let us consider the classes of all constrained atoms taken in I^- and I^+, for all $\mathcal{A}$-valuations extending θ to the variables in Y_k. By hypothesis I is finite, thus these classes are finite sets, say $\{c_{k,1}|A_{k,1}, ..., c_{k,n}|A_{k,n_k}\} \subseteq I^+$ where $\neg A_{k,1}, ..., \neg A_{k,n_k}$ are negative literals in α_k, and $\{c'_{k,1}|A'_{k,1}, ..., c'_{k,n'}|A'_{k,n'}\} \subseteq I^-$ where $A'_{k,1}, ..., A'_{k,n'}$ are positive literals in α_k.
Let $\{A_{k,n_k+1}, ..., A_{k,m_k}\} = \{A'_{k,i} \mid 1 \leq i \leq n'\}$. As I is closed by disjunction on false atoms, there exists $\{c_{k,n_k+1}|A_{k,n_k+1}, ..., c_{m_k}|A_{k,m_k}\} \subseteq I^-$ such that for all $c'_{k,i}|A'_{k,i}$, $1 \leq i \leq n'$ there exists a j, $n_k + 1 \leq j \leq m_k$ such that $A'_{k,i} = A_{k,j}$ and $\mathcal{A} \models c'_{k,i} \rightarrow c_{k,j}$.
Now let $c_k = \forall Y_k(\neg d_k \vee \bigvee_{i=1}^{n_k} c_{k,i})$. For all k, $c_k\theta$ is true, hence $c = \bigwedge c_k$ is $\mathcal{A}$-satisfiable and from the definition of T_P^- we get $c|p(X) \in T_P^-(I)$.

□

Theorem 5.10. For all $n \geq 0$, $[T_P \uparrow n]_\mathcal{A} = \Phi_P^\mathcal{A} \uparrow n$.

PROOF: By induction on n. The base case $n = 0$ is trivial. For the induction step, we have $[T_P \uparrow n]_\mathcal{A} = [T_P(T_P \uparrow n-1)]_\mathcal{A}$. By corollary 5.4 and 5.7, $T_P \uparrow n-1$ is finite and closed by disjunction on false atoms, hence we get by lemma 5.9, $[T_P \uparrow n]_\mathcal{A} = \Phi_P^\mathcal{A}([T_P \uparrow n-1]_\mathcal{A})$. Therefore by the induction hypothesis we conclude $[T_P \uparrow n]_\mathcal{A} = \Phi_P^\mathcal{A}(\Phi_P^\mathcal{A} \uparrow n-1) = \Phi_P^\mathcal{A} \uparrow n$. □

Corollary 5.11. For all $n \geq 0$, $\Phi_P^\mathcal{A} \uparrow n$ has a finite cover.

Corollary 5.12. $[\mathcal{F}(P)]_\mathcal{A} = \Phi_P^\mathcal{A} \uparrow \omega$.

Theorem 5.13. (Correctness and completeness of the fixpoint semantics w.r.t. the logical semantics) $\mathcal{F}(P) \subseteq \mathcal{L}(P)$, $\mathcal{L}^+(P) \sqsubseteq_f \mathcal{F}^+(P)$ and $\mathcal{L}^-(P) \sqsubseteq \mathcal{F}^-(P)$.

PROOF: By corollaries 5.12 and 5.2 we get $[\mathcal{F}(P)]_\mathcal{A} = [\mathcal{L}(P)]_\mathcal{A}$, thus by definition of $\mathcal{L}$ we have $\mathcal{F}(P) \subseteq \mathcal{L}(P)$.

Conversely, let $c|p(X) \in \mathcal{L}^+(P)$, by theorem 5.1, $\forall X(c \rightarrow p(X))$ is true in $\Phi_P^\mathcal{A} \uparrow n$ for some n, thus by theorem 5.10, it is true in $T_P \uparrow n$ for some n. Now as $T_P \uparrow n$ is finite (corollary 5.4), there exists $\{d_1|p(X), ..., d_k|p(X)\} \subseteq T_P \uparrow n$ such that $\mathcal{A} \models c \rightarrow \bigvee_{i=1}^k d_i$, so $\mathcal{L}^+(P) \sqsubseteq_f \mathcal{F}^+(P)$. We prove similarly that $\mathcal{L}^-(P) \sqsubseteq_f \mathcal{F}^-(P)$, yet by corollary 5.8 we get $\mathcal{L}^-(P) \sqsubseteq \mathcal{F}^-(P)$. □

Theorem 5.14. (Correctness and completeness of the operational semantics w.r.t. the logical semantics) $\mathcal{O}(P) \subseteq \mathcal{L}(P)$, $\mathcal{L}^+(P) \sqsubseteq_f \mathcal{O}^+(P)$ and $\mathcal{L}^-(P) \sqsubseteq \mathcal{O}^-(P)$. Similarly $\tilde{\mathcal{O}}(P) \subseteq \mathcal{L}(P)$, $\mathcal{L}^+(P) \sqsubseteq_f \tilde{\mathcal{O}}^+(P)$ and $\mathcal{L}^-(P) \sqsubseteq \tilde{\mathcal{O}}^-(P)$.

PROOF: By theorems 5.13 and 4.8 (resp. 4.9). □

6. Comparison with other schemes for constructive negation

The constructive negation scheme of Chan [6] for logic programs, and Stuckey [23] for constraint logic programs, relies on a transition relation over explicitly quantified complex goals. The transition relation is defined by the usual SLD resolution rule for positive subgoals and by the following constructive negation rule CN for complex subgoals:

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

for each $j \in J$ where $\bigvee_{j\in J} c_j \wedge \beta'_j$ is a disjunctive normal form of $\bigwedge_{k\in K} \neg\exists Z_k(c \wedge d_k \wedge \beta_k)$ and where $\{c \wedge d_k|\alpha_k\}_{k\in K}$ is a frontier in a SLDCN derivation tree for $c|\beta$.

Not only the constraints but also the goals in the frontier of an auxiliary derivation tree are thus transformed into disjunctive normal form and reinjected in the resolvant at each resolution step with a negative subgoal. 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. The practical advantage of constructive negation by pruning is that it relies on standard SLD derivation trees for definite goals only. The only extra machinery to handle negation is a concurrent pruning mechanism over standard SLD derivation trees. It is remarkable that the exploitation of concurrency in the development of SLD derivation trees is sufficient to build a complete scheme for negation. This is the case also for the fail answers approach proposed recently by Drabent in [7] for normal logic programs. Drabent's execution model is essentially equivalent to constructive negation by pruning in that case, the success by pruning rule is a special case of the fail answer approach, we believe that both schemes define in fact the same set of computed answer substitutions for normal logic programs.

If we look at the nesting of negation, we can see that the effect of doubly negating a goal is to collect in a single answer constraint all the successes found for the positive goal. Corollary 5.8 shows that the computed answer constraints for negative goals are closed by disjunction, thus a simple way to obtain a strong completeness result w.r.t. the logical semantics (i.e. $\mathcal{L}(P) \sqsubseteq \mathcal{O}(P)$ instead of $\mathcal{L}(P) \sqsubseteq_f \mathcal{O}(P)$) is to put double negations on positive goals. On the other hand, in the intensional negation scheme double negations are eliminated by simplification. In this respect our scheme is nearer to the one of Chan and Stuckey.

The closure by disjunction property for negative literals can be seen also as a drawback as at some point in the execution all the current information on a negative literal need to be handled by the constraint solver. A general solution to this problem is to exploit the trade-off there is between the constraint solver and the non-deterministic derivation system. This is possible if the structure $\mathcal{A}$ is admissible [23] (cf. section 2.1), in that case the language of constraints need not even be closed by negation. Constructive negation by pruning can be adapted mainly by changing the definition of $\neg_V F$. The negation of the projection of the constraints in a frontier $F = \{c_1|\alpha_1, ..., c_n|\alpha_n\}$ over a set of variables V is then no longer a constraint but a frontier defined as:

$$\neg_V F = \{d_{1,1}|\Box, ..., d_{1,l_1}|\Box\} \times ... \times \{d_{n,1}|\Box, ..., d_{n,l_n}|\Box\}$$

where $\mathcal{A} \models \forall V(\neg\exists Y_i c_i(V, Y_i) \leftrightarrow \exists Z_i(d_{i,1} \vee ... \vee d_{i,l_i}))$ for all $1 \leq i \leq n$. This change amounts to replace in the procedural interpretation the pruning by success rule by a check of satisfiability with at least one of the disjunct, and the success by pruning rule by the creation of a success for each satisfiable disjunct.

Another possible drawback of constructive negation by pruning is that once a derivation tree is developed for a negative literal it receives no more information from the resolution of the positive part of the goal. This is the price to pay for having a single derivation tree for a negative literal instead of duplicating resolution steps at all its occurrences. Many optimizations can nevertheless be imagined, such as sending back for pruning in the auxiliary tree the constraints which are entailed by the frontiers in the main tree.

On the theoretical side constructive negation was proved correct and complete w.r.t. Kunen's logical semantics by Stuckey for consistent fair computation rules [23]. Similar results were obtained for intensional negation [5] and by Drabent [7]. None of these schemes however were provided with a fixpoint semantics. The full abstraction theorem given for constructive negation by pruning allows to analyze and transform normal $CLP(\mathcal{A})$ programs by reasoning at the fixpoint semantics level of abstraction while preserving the program equivalence based on the observation of computed answer constraints. Note that a similar result is conjectured in [3]. Note also that the full abstraction result has been obtained without fixing a resolution strategy, it holds w.r.t. the computation of all frontiers in uniform derivation trees. This left open the problem of giving a fully abstract fixpoint semantics under specific strategies, such as breadth-first [23].

7. Variations on a scheme for optimization predicates

7.1. *Query optimization by pruning*

Because of their importance in real-life applications, most constraint logic programming systems, such as CHIP, CLP(R) and Prolog III, include various constructs for optimization w.r.t. an objective function. The optimization of the top-level query [20] [24] can be achieved with a simple pruning mechanism on the derivation tree of the query, assuming the constraint solver can deal with constraint minimization. By constraint minimization we mean, given a constraint $c(X, Y)$ and a term $f(Y)$, determine the minimum value, noted $min_{c(X,Y)}f(Y)$, of $f(Y)$ under constraint $c(X, Y)$, when it exists, fail otherwise.

The main procedure is a variant of the branch-and-bound procedure. Once a successful derivation is found for the query $G(X)$, say with answer constraint c, then the optimal value v of the objective function f is computed for that derivation, $v = min_c f(X)$, and the tree is pruned by adding the constraint $f(X) \leq v$. If v doesn't exist it is a failure, otherwise whenever the tree gets finite after pruning, the optimal solutions to the query are given by the remaining successful derivations.

There are some problems however to use that procedure recursively for optimization goals, noted $min(G(X), f(X))$ where $G(X)$ is a goal and $f(X)$ a term, because of the following non-logical behavior well-known in current CLP systems:

```
p(X):- X>=0.
q(X):- X>=1.
?  q(X), min(p(X),X).
X=1
```

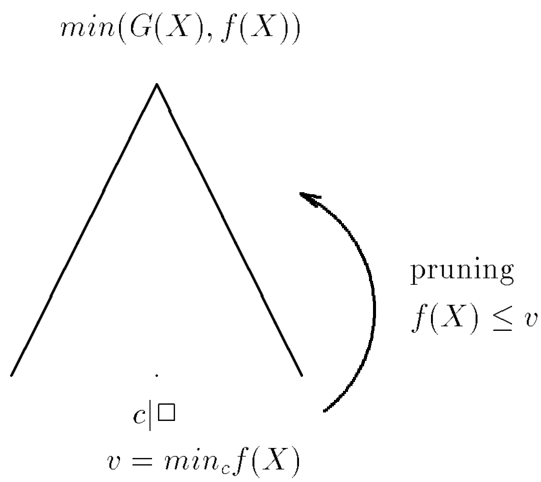


Figure 4. Query optimization by pruning.

```
? min(p(X),X), q(X).
fail
```

In [8] and [21], it is shown that optimization higher-order predicates can be provided with a faithful logical semantics based on constructive negation. The correct answer in the example is fail, whereas $X = 1$ is a correct answer to the goal $min((p(X), q(X)), X)$. In the next section we show how constructive negation by pruning specializes to a correct and complete concurrent branch and bound like procedure for optimization predicates.

7.2. *Optimization higher-order predicates*

Definition 7.1. 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 allow for the arbitrary composition of optimization subgoals and for the recursive definition of predicates through their optimal solutions, as used for instance in dynamic programming. We shall show the completeness of the following concurrent branch and bound like procedure for μCLP programs.

To resolve a goal of the form $c|\alpha, min(G(X), f(X)), \alpha'$, two SLD derivation trees are developed, one Ψ_1 for $c|\alpha, G(X), \alpha'$, and one Ψ_2 for $c \wedge f(Y) < f(X)|G(Y)$. Once a successful derivation is found in Ψ_2, say with answer constraint d, then Ψ_1 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_1, say with answer constraint e, then Ψ_1 and Ψ_2 are pruned by adding the constraint $f(X) \leq w$ if $w = min_e f(X)$ exists, false otherwise. We get a successful derivation for the minimization goal when we get a successful derivation in Ψ_1 and Ψ_2 is finitely failed. The minimization goal gets finitely failed if Ψ_1 gets finitely failed after pruning. Figure 5 illustrates the mutual pruning mechanism.

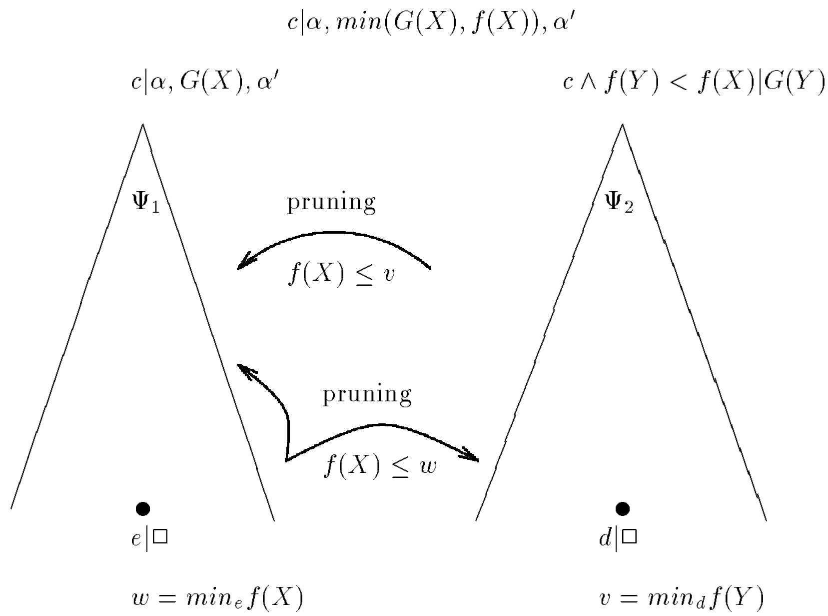


Figure 5. Subgoal optimization by pruning.

Note that in the previous case of query optimization the context is empty, $c = true$, $\alpha = \alpha' = \Box$, hence both CSLD trees Ψ_1 and Ψ_2 can be taken identical up to variable renaming. The mutual pruning mechanism of the optimization scheme can thus be simplified into a single pruning in Ψ_1 with constraint $f(X) \leq w$, as described in the previous section. This is no longer possible if the goal contains a constraint or an atom outside the minimization predicate.

Example 7.2. Consider the $\mu CLP(R)$ program P

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

and the goal `X>1|min(p(X),X)`*.*
The first SLD tree for `X>1|p(X)` *is infinite. The second SLD tree for* `X>1,Y<X|p(Y)` *contains a success with answer constraint $Y = 0$. The first tree is thus pruned with the constraint $X \leq 0$, hence it 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 [24] and [8] 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 (see [21] for a different procedure).

Completeness and full abstraction can be proved by specializing the principle of constructive by pruning to optimization goals in μCLP programs. As is well known, a general first-order formula can be normalized into a normal logic program [17]. μCLP programs can thus 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:

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

As the negative literals in the translation of a μCLP program come all from optimization predicates we can equivalently replace the PRN rule by the following OPT rule for μCLP programs:

$$\text{OPT:} \quad \frac{c|G(X) \ \triangleleft \ F_1 \qquad\qquad c, f(Y) < f(X)|G(Y) \ \triangleleft \ F_2}{c|min(G(X), f(X)) \ \triangleleft \ c \times F_1 \times \{\neg_V S|min(G(X), f(X)), \ \neg_V F_2|\Box\}}$$

where $S \subseteq \mathcal{S}(F_2)$, $V = V(c) \cup X$ and $Y \cap V = \emptyset$.

Now it is easy to see that as the variables X and Y are related by the constraint $f(Y) < f(X)$ solely, the negation of constraints involved in the OPT rule amount to a simple form of constraint minimization:

Proposition 7.3. (negation of constraints as constraint minimization) In the OPT rule for all $d|\alpha \in F_2$, let $Z = V(d) \setminus V$, if $v = min_d f(Y)$ exists then $\mathcal{A} \models (c \wedge \neg\exists Z\ d) \leftrightarrow (c \wedge f(X) \leq v)$, otherwise $c \wedge \neg\exists Z\ d$ is $\mathcal{A}$-unsatisfiable.

Proof: As $Y \cap V = \emptyset$ we have $Y \subseteq Z$ and $d = c \wedge f(Y) < f(X) \wedge d'(Z)$. If $v = min_d f(Y)$ exists then we have $v = min_{d'(Z)} f(Y)$ thus $\mathcal{A} \models (c \wedge \neg\exists Z d) \leftrightarrow (c \wedge f(X) \leq v)$. If $min_{d(X,Y,Z)} f(Y)$ doesn't exist then for any value v, the constraint $f(Y) < v \wedge d'(Z)$ is $\mathcal{A}$-satisfiable, thus $c \wedge \neg\exists Z d$ is $\mathcal{A}$-unsatisfiable. $\Box$

The OPT rule can thus be interpreted procedurally with both a pruning by success rule (PBS) that prunes the main tree with the constraint $f(X) \leq v$ where v is the optimal value of the objective function for a success in the auxiliary tree (prune with false if v doesn't exist), and with a success by pruning rule (SBP) that negates frontiers in the auxiliary tree once a successful derivation is found in the main tree. It is worth noting however that the computation of frontiers is not necessary in this context, the following proposition shows that the SBP rule can be replaced by a reversed pruning operation and by a check for finite failure in the auxiliary tree.

Proposition 7.4. In the OPT rule, suppose $d|\Box \in \mathcal{S}(F_1)$*, if* $w = min_d f(X)$ *exists and* $(f(X) \leq w) \times F_2 = \emptyset$ *then* $\mathcal{A} \models (d \wedge \neg_X F_2) \leftrightarrow (d \wedge f(X) \leq w)$ *otherwise* $d \wedge \neg_X F_2$ *is* $\mathcal{A}$*-unsatisfiable.*

PROOF: Remark first that due to the similarity of the goals in the premises of the OPT rule, if $f(X)$ can take two values $v < v'$ under constraint d, then $f(Y)$ can take value v under constraint e for some $e|\alpha \in F_2$.

If $w = min_d f(X)$ exists and $(f(X) \leq w) \times F_2 = \emptyset$ then for all $e|\alpha \in F_2$ $\mathcal{A} \models e \rightarrow f(Y) \geq w$, thus $\mathcal{A} \models (d \wedge f(X) \leq w) \rightarrow (d \wedge \neg_X F_2)$. Furthermore by (the contrapositive of) the previous remark we have $\mathcal{A} \models (d \wedge \neg_X F_2) \rightarrow (d \wedge f(X) = w)$.

Otherwise, either $min_d f(X)$ doesn't exist, in which case by the previous remark $d \wedge \neg_X F_2$ is $\mathcal{A}$-unsatisfiable, or $(f(X) \leq w) \times F_2 \neq \emptyset$ in which case there exists $e|\alpha \in F_2$ such that $e \wedge f(X) \leq w$ is $\mathcal{A}$ satisfiable, thus $d \wedge \neg \exists Z\ e$ where $Z = V(e) \setminus X$ is $\mathcal{A}$-unsatisfiable, and so is $d \wedge \neg_X F_2$. $\Box$

Note finally that given a successful derivation with constraint d in the main tree such that $w = min_d f(X)$ exists, even if the auxiliary tree doesn't get finitely failed by pruning, both the main tree and the auxiliary tree can be pruned with the constraint $f(X) \leq w$ as we already know the there will be a similar successful derivation in the auxiliary tree with $f(Y) = w$. This provides evidence that the procedure given in the introduction of this section is a sound procedural interpretation of the principle of constructive negation by pruning. As the transformations preserve the equivalence with the general scheme, the completeness results of the previous sections continue to hold:

Theorem 7.5. Let P *be a* μCLP *program. The fixpoint semantics* $\mathcal{F}(P)$ *is fully abstract w.r.t. the answer constraints computed by rules TRIV, RES, FRT and OPT. The operational semantics based on rules TRIV, SLD and OPT is sound and complete w.r.t. the logical semantics* $\mathcal{L}(P)$*.*

7.3. Optimization predicates with protected variables

The optimization predicates defined in [8] or [21] are more general than those considered in the previous section 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 7.6. 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 7.7. Local optimization predicates can be used to express the min-max method of game theory with the following goal:

```
? 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.

The previous execution model for optimization predicates is not correct for local optimization predicates. This is not surprising as it is easy to see that any normal logic program can be encoded as a definite CLP program with local optimization predicates encoding negations. Therefore there is no hope to fundamentally improve a general scheme for negation in the context of local optimization predicates.

Proposition 7.8. Any normal logic program is equivalent to a CLP program containing local optimization predicates in place of negative literals.

PROOF: Given a normal logic program P and a normal goal G let us consider the CLP goal $\overline{G}$ and the CLP program $\overline{P}$ over the Herbrand domain and the natural numbers obtained by replacing each negative literal $\neg p(X)$ by $max(q(X,y),[X],y)$ where q is a new predicate symbol, and by adding the clauses

$$q(X,0).$$

$$q(X,y) \leftarrow p(X).$$

One easily checks that $P^* \models \exists G$ iff $\overline{P}^*, \mathcal{N} \models \exists \overline{G}$. □

Constructive negation by pruning can be used to interpret local optimization predicates, it can be used as well to interpret directly preference predicates over solutions defined by CLP programs, that is to evaluate goals of the form

$$G(X) \wedge \neg \exists Y (G(Y) \wedge better(Y,X)).$$

where *better* is a user-defined preference predicate. This form of optimization, called relational optimization, doesn't need to encode preferences by objective functions, it is discussed and illustrated by one application in [10].

8. Conclusion

The principle of constructive negation by pruning (CNP) provides a correct and complete operational semantics for normal CLP programs w.r.t. Kunen's three-valued logical semantics. CNP is the first scheme to receive a fixpoint semantics which fully characterizes the operational behavior of normal CLP programs w.r.t. answer constraints. Furthermore, that fixpoint semantics is based on a continuous finitary version of Fitting's operator which is interesting to study in its own right.

We have shown that CNP provides a simple operational semantics to normal CLP programs: there is no complex subgoals with explicit quantifiers, no formula transformation at run-time or compile-time, only pruning over concurrent standard SLD derivation trees. It is remarkable that exploiting concurrency in the formation of standard SLD trees is sufficient to build a complete scheme for negation. This is an example of the potential power of concurrency in proof theory.

We believe that constructive negation by pruning can lead to a practical scheme for handling negation in CLP systems. Of prime importance is the study of efficient constraint solvers for constraint systems with negation over finite and infinite trees, linear arithmetic, finite domains, order-sorted domains, etc. In the context of optimization higher-order predicates we have shown that constraint minimization is the only required extension of the solvers, and that CNP specializes to a concurrent branch and bound like procedure, without frontier computation in SLD trees.

Ongoing work concerns on the one hand the natural extension of the class *cc* of concurrent constraint programming languages [22] with constructive negation by pruning and optimization higher-order agents [9], and on the other hand the bottom-up abstract interpretation of normal CLP programs based on operator T_P.

Acknowledgements

The author is grateful to Annalisa Bossi, Alberto Bottoni, Patrick Cousot, Roberto Giacobazzi and Francesco Ranzato for valuable discussions of this material, as well as to the anonymous referees for many helpful comments and suggestions on an earlier version of this paper. This work has been partially supported by MESR contract 93S0051 PRC AMN.

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, H.A. Blair, A. Walker, "Towards a theory of declarative knowledge", in Foundations of Deductive Databases and Logic Programming, J. Minker Ed., Morgan Kaufmann Pub., pp.89-148, 1987.
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", Acta Informatica, 32:1, pp.27-59, 1995.
8. F. Fages, "On the semantics of optimization predicates in CLP languages", 13th FSTTCS conference, Bombay, LNCS 761, Springer-Verlag, pp. 193-204, 1993.
9. F. Fages, "Constructive negation by pruning and optimization higher-order predicates for CLP and CC languages", in Constraint Programming: Basics and Trends, Ed. A. Podelski, Springer-Verlag LNCS 910, pp.68-89. 1995.
10. F. Fages, J. Fowler, T. Sola, "Handling preferences in constraint logic programming with relational optimization", Proc of PLILP'94, Madrid, Springer Verlag, LNCS 844, pp.261-276, 1994.
11. M. Fitting, "A Kripke/Kleene semantics for logic programs", Journal of Logic Programming, 2(4), pp.295-312, 1985.

12. 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.
13. J. Jaffar, J.L. Lassez, "Constraint Logic Programming", Proc. of POPL'87, Munich, pp.111-119, 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.J. Maher, P.J. Stuckey, "Expanding query power in constraint logic programming languages", Proc. of NACLP'89, MIT Press, pp.20-36, 1989.
21. K. Marriott, P.J. Stuckey, "Semantics of constraint logic programs with optimization", Technical report, Univ. of Melbourne, 1993.
22. V. Saraswat, "Concurrent constraint programming", MIT Press, 1993.
23. P. Stuckey, "Constructive negation for constraint logic programming", Proc. LICS'91, pp.328-339, 1991.
24. P. Van Hentenryck : "Constraint Satisfaction in Logic Programming", MIT Press 1989.