# On the semantics of Optimization Predicates in CLP Languages

François FAGES

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

LIENS - 93 - 12

May 1993

# On the Semantics of Optimization Predicates in CLP languages

François Fages

LIENS-CNRS,
Ecole Normale Supérieure,
45 rue d'Ulm,
75005 Paris, France
fages@dmi.ens.fr

**Abstract**

The Constraint Logic Programming systems which have been implemented include various higher-order predicates for optimization. In CLP(FD) systems, several optimization predicates, such as `minimize(G(X),f(X))`, `minimize-maximum(G(X),[f1(X),...,fn(X)])`, are implemented by using branch and bound algorithms. In CLP(R) systems, the Simplex algorithm used for satisfiability checks can also be used for linear optimization through the predicate `rmin(f(X))` which adds to the constraints on $X$ the ones defining the space where the linear term $f(X)$ is minimized. These optimization constructs do not belong however to the formal CLP scheme of Jaffar and Lassez, and they lack a declarative semantics. In this paper we propose a general definition for optimization predicates, for which one can provide both a logical and a fixpoint semantics based on Kunen-Fitting's semantics of negation. We show that the branch and bound algorithm can be derived as a refinement of the implementation of the semantics using CSLDNF-resolution, and that the branch and bound algorithm can be lifted to a full first-order setting with constructive negation.

## 1 Introduction

The Constraint Logic Programming systems which have been implemented include various higher-order predicates for optimization. In CLP(FD) systems as CHIP, defined over finite domains, several optimization predicates, such as `minimize(G(X),f(X))`, or `minimize-maximum(G(X),[f1(X),...,fn(X)])`, are implemented with branch and bound algorithms [8]. In CLP(R) systems, defined over real numbers, the Simplex algorithm used for satisfiability checks can also be used for linear optimization through the predicate `rmin(f(X))` which adds to the constraints on $X$ the ones defining the space where the linear term $f(X)$ is minimized. These optimization constructs do not belong however to the formal CLP scheme of Jaffar and Lassez [5], and they lack a declarative semantics.

The first problem to solve is the dependence of the result on the ordering of the goals. In many systems indeed the constraints on the variables appearing in the optimization goal are passed to the optimization process, producing the following problematical behavior:

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

? X>=1 , minimize(p(X),X).
X=1

? minimize(p(X),X) , X>=1.
no
```

Clearly the optimization process should be localized to the goal given as argument, and the other constraints inherited from the other goals should not change the optimality condition. Therefore the correct answer in the previous example is *no*. If $X = 1$ was the intended answer, one should write:

```
? minimize((X>=1,p(X)),X).
X=1
```

With this provision one can give a declarative reading to CLP programs containing optimization predicates. We show that Kunen's three-valued semantics of logic programs with negation is all we need to do so, and that optimization predicates can thus be treated as higher-order constraints. After reviewing completeness results in the CLP scheme, we show that the well-known branch and bound algorithm can be derived as a refinement of the implementation of the semantics using CSLDNF-resolution, and that the branch and bound algorithm can be lifted to a full first-order setting with constructive negation.

## 2 The declarative semantics of optimization predicates

**Definition 1** *Let $(A, \leq)$ be a totally ordered structure. The* minimization *higher-order predicate*

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

*is defined as a notation for the formula:*

$$G(X,Y) \wedge \forall Z\ (G(X,Z) \supset f(X,Z) \not< f(X,Y))$$

*An optimization constraint logic program (OCLP) (resp. goal) is a CLP program (resp. goal) which may contain occurrences of the minimization predicate in rules bodies (resp. in goals).*

The second argument to the predicate $min$ is a possibly empty list of "protected" variables, $[X]$. Only the variables of the goal, away from $X$, are affected by the optimality condition.

As is well known, general first-order formulas can be normalized [6]. An OCLP program $P$ can be transformed into an equivalent normal CLP program containing negations, by replacing each occurrence of the atom

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

by the conjunction of atoms

$$G(X,Y), \neg p(X,Y)$$

where $p$ is a new predicate symbol, and by adding to the program the rule:

$$p(X,Y) \leftarrow f(X,Z) < f(X,Y) | G(X,Z).$$

In the following, $\overline{P}$ denotes the normal CLP program obtained by repeatedly applying this transformation to $P$, and $P^*$ denotes the Clark's completion [6] of $P$ (without Clark's equality axioms as symbols are interpreted in $A$).

**Definition 2** *The semantics of an OCLP program $P$ is the set of 3-valued consequences of $\overline{P}^* \wedge th(A)$. A correct answer to an OCLP query $G$ and a program $P$ is a set of constraints $c$ such that*

$$\overline{P}^* \wedge th(A) \models_3 \forall(c \rightarrow G) \wedge \exists(c)$$

Going back to the example of the introduction, we can check that the correct answer to

$$X \geq 1, min(p(X),[\,],X)$$

is *no*, that $X \geq 1$ is a correct answer to

$$X \geq 1, min(p(X),[X],X),$$

and that $X = 1$ is the correct answer to the goal

$$min(X \geq 1 | p(X),[\,],X).$$

In general the answers of an OCLP query are non ground, and can be arbitrary sets of constraints. Considering the rule

$$p(X,Y) \leftarrow 0 \leq X, X \leq Y.$$

$X = Y$ is a correct answer to the query

$$min(p(X,Y),[\,],Y-X).$$

The definition of OCLP programs does not exclude recursion though optimization predicates. A similar problem is discussed in [1] and [2] where the stratified semantics of aggregates are generalized using the well-founded and stable model semantics of logic programs. In the context of OCLP programs, the natural theory to apply is Fitting-Kunen's 3-valued semantics, which does not coincide with stratified and well-founded semantics. From a practical point of view one can notice that definite OCLP programs usually don't contain recursion through optimization. In general however, we have:

**Proposition 1** *Any normal logic program is equivalent to a definite OCLP program.*

**Proof:** Let us consider the OCLP program over the natural numbers obtained from the normal logic program 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 rules

$$q(X,0).$$
$$q(X,y) \leftarrow p(X).$$

We have $\exists X \exists y\ max(q(X,y),[X],y)$ iff $\exists X \exists y \forall z\ q(X,y) \wedge \neg(q(X,z) \wedge z > y)$
iff $\exists X \exists y\ (y = 0 \wedge \neg p(X)) \vee (p(X) \wedge \forall z \neg(\alpha(X,z) \wedge z > y))$
iff $\exists X \exists y\ y = 0 \wedge \neg p(X)$
iff $\exists X \neg p(X)$. []

## 3 Completeness results

The completeness result of SLDNF-resolution w.r.t. to the three-valued semantics of logic programs [4] relies on the properties of the finite powers of Fitting's operator $\Phi^A_P$. These properties generalize to normal CLP programs:

**Theorem 1** *[3] [7] Let A be a structure and P a normal CLP program. Then the following are equivalent:*

- $\Phi^A_P \uparrow n(c|G) = t$ *for some finite n,*
- $th(A) \wedge \overline{P}^* \models_3 \forall(c \rightarrow G) \wedge \exists(c)$

In this way Stuckey [7] proved the completeness of CSLDCN-resolution (i.e. constraint SLD-resolution with constructive negation) for normal CLP programs. CSLDCN-resolution is based on the CSLD inference rule for positive goals, and on the CSLDCN inference rule for negative goals:

$$CSLD \frac{\leftarrow c|A_1,...,A_i,...,A_n \quad th(A) \models \exists(c')}{\leftarrow c'|A_1,...,A_{i-1},B_1,...,B_m,A_{i+1},...,A_n}$$

where $c' = c \wedge A_i = B$, $(B \leftarrow B_1,...,B_m) \in P$.

$$CSLDCN \frac{\leftarrow c|A_1,...,\neg A_i,...,A_n \quad th(A) \models \exists(c')}{\leftarrow c'|A_1,...,A_{i-1},A_{i+1},...,A_n}$$

where $c' = c \wedge \neg\exists^{\sim}(c_1) \wedge ... \wedge \neg\exists^{\sim}(c_n)$ (the existential closure being over variables not in $c$), and $\{c \wedge c_1,...,c \wedge c_n\}$ are the successful derivations of the goal $\leftarrow c \wedge A_i$.

**Theorem 2** *[7] Let P be a normal OCLP program over a structure A.*
*If* $th(A) \wedge \overline{P}^* \models_3 \forall(c \rightarrow G) \wedge \exists(c)$ *then the CSLDCN-derivation tree for* $(c|G)$ *contains successful derivations with constraints* $c_1,...,c_n$*, such that* $A \models c \supset \exists^{\sim}c_1 \vee ... \vee \exists^{\sim}c_n$.
*If* $th(A) \wedge \overline{P}^* \models_3 \forall(c \rightarrow \neg G) \wedge \exists(c)$ *then the CSLDCN-derivation tree for* $(c|G)$ *is finitely failed.*

In particular the completeness of CSLDNF-resolution (i.e. constraint SLD-resolution with negation by failure) follows under the non-floundering assumption.

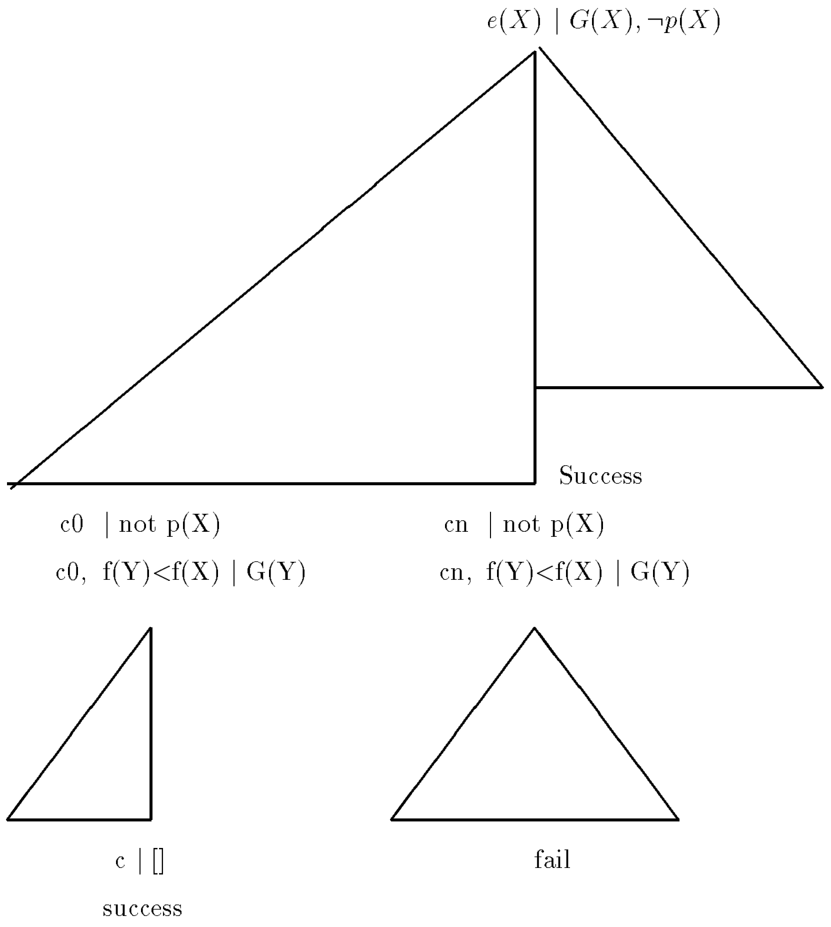
## 4 The branch and bound algorithm as negation by failure

In this section we study the application of the optimization predicate over a goal $G(X, Y)$, such that all the successful CSLD-derivations of $G(X, Y)$ instanciate the arguments $X$ and $Y$ to some values. This is typically the case of optimization in CLP(FD), where enumeration is mixed with constraint propagation in order to palliates the incompleteness of the constraint solvers [8].

Under these assumptions, it is clear that the negative goals introduced by the optimization predicates (in $G(X, Y), \neg p(X, Y)$) never flounder. Thus CSLDNF-resolution is complete w.r.t. the (3-valued) semantics of such OCLP programs. For simplicity, let us consider the goal

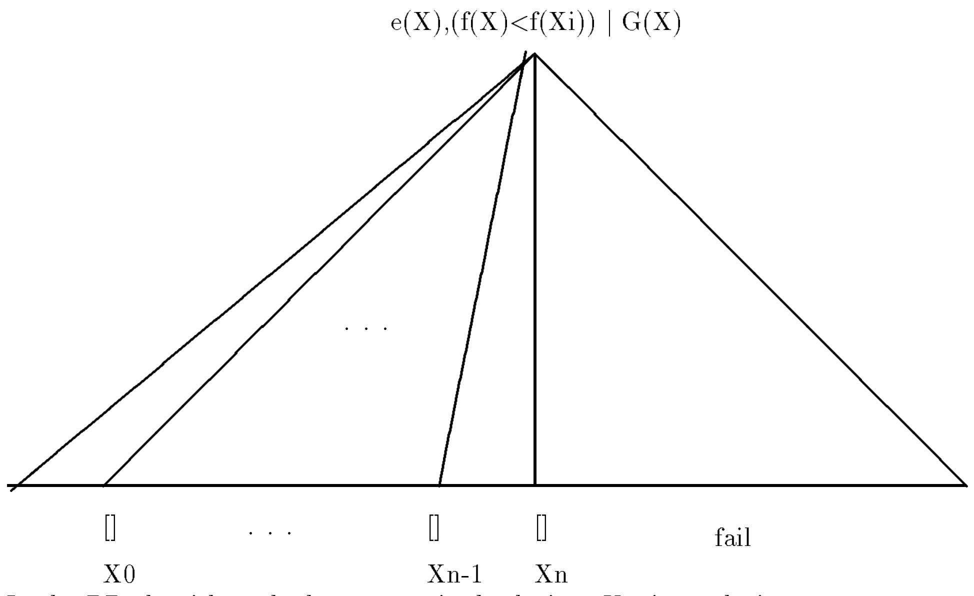
$$e(X)|min(G(X), [\,], f(X)),$$

and its CSLDNF-derivation tree.



For the moment let us consider the search for only one successful derivation of the optimization goal, not all successful derivations. Under the normal left-right order traversal of the tree, when a successful derivation is obtained for $e(X)|G(X)$ with constraint $c_i$, then $c_i|\neg p(X)$ remains to be shown, and for this, another derivation tree is developed for the goal $c_i, f(Y) < f(X)|G(Y)$. If this subtree is finitely failed then we obtain a successful derivation for the optimization goal. Otherwise if the derivation subtree contains a successful derivation, then it is a failure for the optimization goal, and the successful subderivation is lost. Therefore a derivation subtree for $G$ is developed for each successful derivation of $e(X)|G(X)$.

The well-known branch and bound algorithms can be presented as optimized versions of CSLDNF-resolution procedures, that exploit the successful derivations found in the refutation of the optimality of a solution. In the *backtracking version* of the branch and bound algorithm (BB), a *single* derivation tree for $G$ is developed. When a successful derivation is found (under the left-right order traversal), the corresponding solution $X_i$ is memorized, and the search by backtracking continues with the additional constraint $f(X) < f(X_i)$. The additional constraint is used to prune the search space and explore only a portion of the derivation tree:



In the BB algorithm, the last memorized solution, $X_n$, is a solution to:

$$min(e(X)|G(X), [\,], f(X))$$

To show that $X_n$ is indeed a solution to

$$e(X)|min(G(X), [\,], f(X))$$

it suffices to check that the goal

$$e(X), f(X) < f(X_n)|G(X)$$

fails. If it is not the case, then the goal $e(X)|min(G(X), [\,], f(X))$ must fail. The gain in efficiency over CSLDNF-resolution is obvious as only two derivation trees are thus developed in this way.

Therefore two algorithms are possible for finding the answers to the goal

$$e(X)|min(G(X), [\,], f(X))$$

depending whether the branch and bound algorithm is applied initially to $e(X)|G(X)$ or to $G(X)$:

**Algorithm 1** *BB algorithm with environment constraints.*

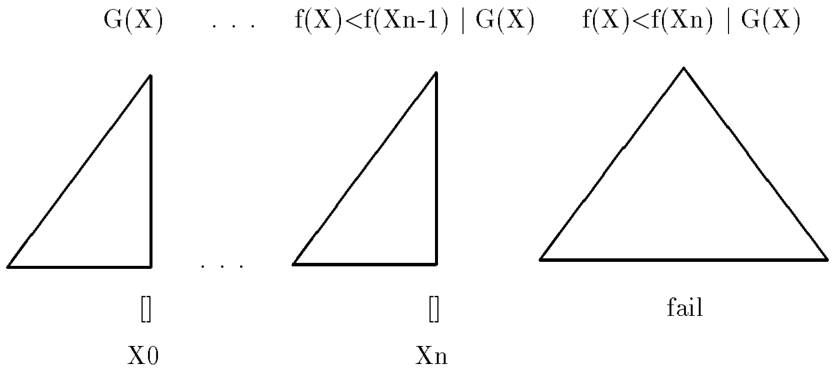
1. *compute one solution $X_n$ to $min(e(X)|G(X), [\,], f(X))$ by using BB algorithm,*
2. *check by CSLDNF-resolution that $e(X), f(X) < f(X_n)|G(X)$ admits no solution, otherwise fail,*
3. *return $X_n$, or if all solutions are needed, return the answers to $e(X), f(X) = f(X_n)|G(X)$ computed by CSLDNF-resolution.*

**Algorithm 2** *BB algorithm without environment constraints.*

1. *compute one solution $X_n$ to $min(G(X), [\,], f(X))$ by using BB algorithm,*
2. *return the solutions to $e(X), f(X) = f(X_n)|G(X)$ computed by CSLDNF-resolution.*

In actual CLP(FD) systems with optimization predicates, algorithm 1 without step 2 is generally implemented, hence the difficulties mentioned in the introduction concerning the declarative semantics and the possibility to treat optimization predicates as higher-order constraints. Algorithm 2 does not use the constraints inherited from the environment to prune the search space for finding the optimal cost of a solution (step 1). Note however that, under termination assumptions, step 1 in algorithm 2 can be done at compile time.

Note also that other versions than the backtracking version of the branch and bound algorithm can be preferred for implementation in a CLP system. In the *iterative version* of the branch and bound algorithm, once a successful derivation for $G(X)$ is found, the corresponding solution $X_0$ is memorized, and another derivation tree is developed for $f(X) < f(X_0) \mid G(X)$. When the derivation tree is finitely failed, the last memorized solution is optimal.



When heuristic search techniques are used in combination with constraint propagation, the iterative version of the branch and bound algorithm makes it possible to change the order in which goals are selected, according to the new constraints added and to the heuristic. For this reason the iterative version can be practically more efficient.

## 5 The branch and bound algorithm lifted to the full first-order setting with constructive negation

CSDLCN-resolution provides a complete procedure for general OCLP programs without the non-floundering assumption. Let us consider the goal $min(G(X), [\,], f(X))$. On a successful derivation of $G(X)$ with constraint $c_i(X)$, constructive negation for the remaining goal

$$c_i(X) \mid \neg p(X)$$

consists in developing a complete derivation tree for

$$c_i(X), f(Y) < f(X) \mid G(Y)$$

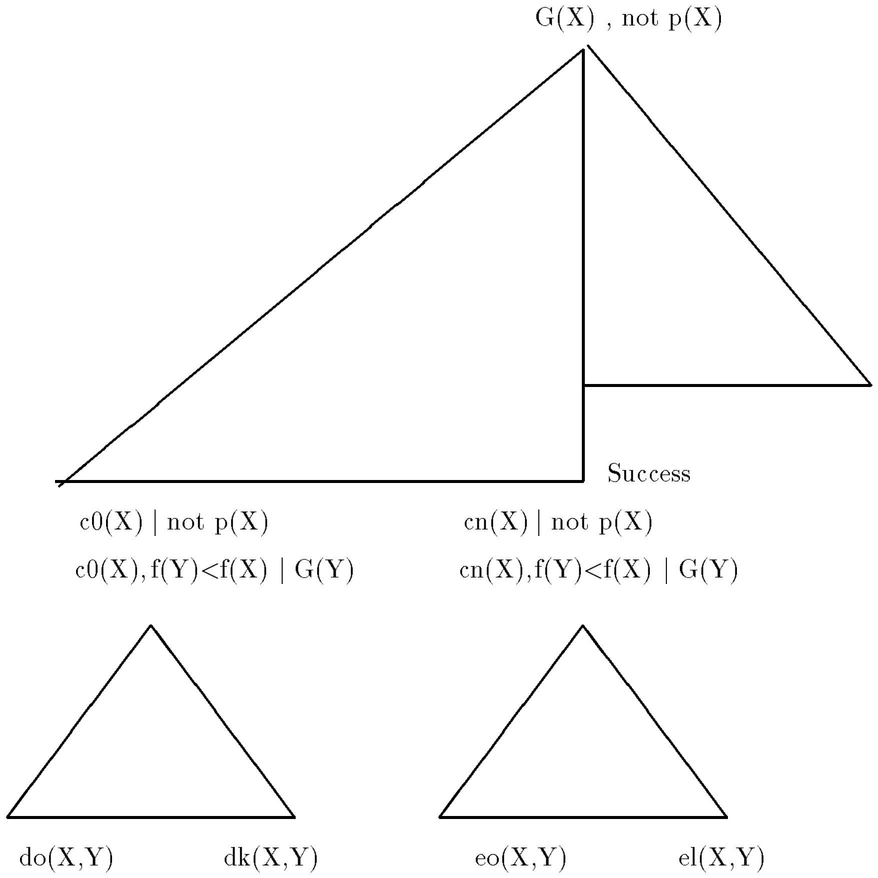
If $c_i(X) \wedge d_0(X,Y), ..., c_i(X) \wedge d_k(X,Y)$ are the constraints associated to the successful derivations of this tree, then the negative goal is successful if the constraint

$$\forall Y \;\; c_i(X) \wedge \neg d_0(X,Y) \wedge ... \wedge \neg d_k(X,Y)$$

is satisfiable[1].

Therefore a complete derivation tree for $G$ is developed for each successful derivation of $G(X)$ not satisfying that condition:

[1]Note that if the structure is admissible [7] this condition is equivalent to a conjunction of existentially quantified disjunctions of conjunctions of admissible constraints.



Now the transformations described in the previous section can be applied in a similar fashion here in order to generalize the branch and bound algorithm to a full first-order setting. For instance the iterative version of the generalized branch and bound algorithm consists in finding a successful derivation for $G(X)$, say with constraint $c_0(X)$, then iterate finding a successful derivation for the goal

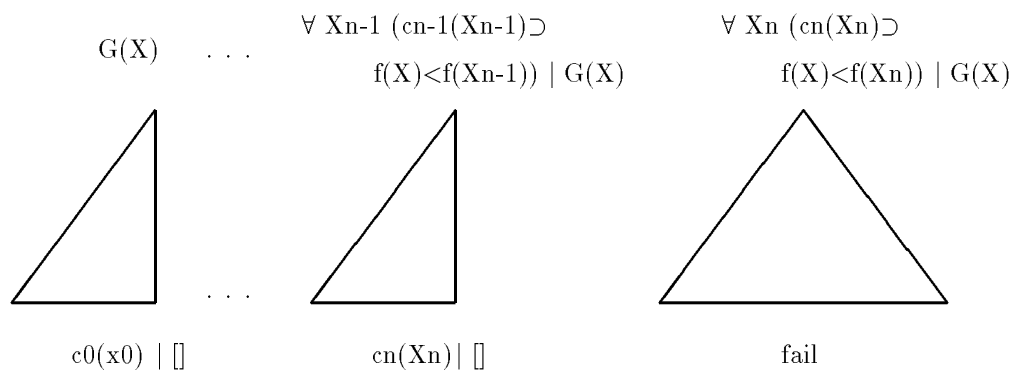
$$\neg\exists X_0(c_0(X_0) \wedge f(X_0) \leq f(X))|G(X)$$

which is equivalent to

$$(\forall X_0\ c_0(X_0) \supset f(X) < f(X_0))|G(X).$$

Note that as the structure is a total order, the constraint $(\forall X_0\ c_0(X_0) \supset f(X) < f(X_0))$ is equivalent to the constraint without universal quantifier $f(X) < k_0$ where $k_0 = min_{c_0(X_0)} f(X_0)$, when it exists. In particular in CLP(R), linear programming algorithms permit to decide efficiently the constraints involved in that restricted form of constructive negation, without having to rely on the admissibility of the structure $R$ result [7] which is based on generally unpractical quantifier elimination techniques.

The derivation trees developed in the iterative first-order branch and bound procedure are thus the followings:



The procedure stops when the derivation tree is finitely failed, in which case the last memorized solution, say $c_i(X)$, is such that

$$\overline{P}^* \wedge th(A) \models_3 \forall X\ c_i(X) \supset G(X)$$

$$\overline{P}^* \wedge th(A) \models_3 \neg\exists Y(\forall X\ c_i(X) \supset f(Y) < f(X))|G(Y)$$

that is

$$\overline{P}^* \wedge th(A) \models_3 \forall Y G(Y) \supset (\exists X\ c_i(X) \wedge f(Y) \not< f(X))$$

hence

$$c_i(X) \wedge \forall Y \neg(c_i(Y) \wedge f(Y) < f(X))$$

is satisfiable, and is an optimal solution.

In this way both algorithms 1 and 2 of the previous section can be generalized to a full first-order setting:

**Algorithm 3** *CLP-BB procedure with environment constraints.*

1. *compute one answer constraint $c_n(X)$, to min$(e(X)|G(X), [\,], f(X))$ by using BB algorithm,*
2. *check by CSLDNF-resolution that $e(X), c_n(X_n), f(X) < f(X_n)|G(X)$ admits no successful derivation, otherwise fail,*
3. *return $c_n(X)$, or if all solutions are needed, return the answers to $e(X), c_n(X_n), f(X) = f(X_n)|G(X)$ computed by CSLDNF-resolution.*

**Algorithm 4** *CLP-BB procedure without environment constraints.*

1. *compute one answer constraint $c_n(X)$ to min$(G(X), [\,], f(X))$ by using BB algorithm,*
2. *return the solutions to $e(X), c_n(X_n), f(X) = f(X_n)|G(X)$ computed by CSLDNF-resolution.*

## 6 Conclusion

Optimization higher-order predicates in CLP systems can be given a logical semantics based on the three-valued consequences of logic programs with negation. We have shown that the well-known branch and bound algorithms can be presented in this framework as specific optimizations of CSLDNF-resolution procedures. Applying the same optimizations to CSLDCN-resolution, which is based on constructive negation, we obtained a powerful generalization of the branch and bound algorithms to a full first-order setting, including linear programming as a deterministic particular case.

**Acknowledgement**

It is a pleasure to acknowledge fruitful discussions with my colleagues at LCR and at LIENS, and to thank Peter Stuckey for providing me with the references to the results on aggregates.

## References

[1] S. Ganguly, S. Greco, C. Zaniolo, “Minimum and maximum predicates in logic programming”. Proc. of PODS’91, Denver, pp. 154-163. 1991.

[2] D.B. Kemp, P.J. Stuckey, “Semantics of logic programs with aggregates”, Proc. of ILPS’91, San Diego, pp.387-401. 1991.

[3] K. Kunen, “Negation in logic programming”, Journal of Logic Programming, 4(3), pp.289-308, 1987.

[4] K. Kunen, “Signed data dependencies in logic programming”, Journal of Logic Programming, 7(3), pp.231-245, 1989.

[5] J. Jaffar, J.L. Lassez, “Constraint Logic Programming”, Proc. of POPL’87, Munich. 1987.

[6] J.W. Lloyd, “Foundations of Logic Programming”, Springer Verlag. 1987.

[7] P. Stuckey, “Constructive negation for constraint logic programming”, Proc. LICS’91, 1991.

[8] P. Van Hentenryck : ”Constraint Satisfaction in Logic Programming”, MIT Press 1989.