[16] H. Hong. Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination. Technical Report 92-02.0, Research Institute for Symbolic Computation, Johannes Kepler University A-4040 Linz, Austria, 1992. To appear in the proceedings of *International Conference on Symbolic and Algebraic Computation* ISSAC-92.

[17] G. Huet and D. Oppen. Equations and rewrite rules. In *Formal Language Theory. Perspectives and Open Problems*, pages 349–405. Academic Press, New York, 1980.

[18] J. Jaffar and S. Michaylov. Methodology and implementation of a CLP system. In J.-L. Lassez, editor, Proceedings 4^{th} ICLP, pages 196–218, Cambridge, MA, May 1987. The MIT Press.

[19] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages, Munich, Germany*, pages 111–119. ACM, January 1987.

[20] M. Kalkbrenner and S. Stifter. Some examples for using quantifier elimination for the collision problem. Technical Report 87-22, Research Institute for Symbolic Computation, 1987.

[21] Dallas Lankford. On proving term rewriting systems are noetherian. Technical Report MTP-3, Mathematics, Louisiana Tech. Univ., Ruston, LA 71272, May 1979.

[22] R. G. K. Loos. The algorithm description language ALDES (Report). *ACM SIGSAM Bull.*, 10(1):15–39, 1976.

[23] S. McCallum. Solving polynomial strict inequalities using cylindrical algebraic decomposition. Technical Report 87-25.0, RISC-LINZ, Johannes Kepler University, A-4040 Linz, Austria (Europe), 1987.

[24] A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* Univ. of California Press, Berkeley, second edition, 1951.

[25] J. H. Wilkinson. The evaluation of the zeros of ill-conditioned polynomials. parts i and ii. *Numerische Mathematik*, 1:150–166,167–180, 1959.

Procs. ALP'92, LNCS 632, pp. 213-227

A General Scheme for Constraint Functional Logic Programming[1]

Francisco-Javier López-Fraguas
Dpt. Informática y Automática
Universidad Complutense de Madrid
Fac. Matemáticas Av. Complutense s/n
28040 MADRID SPAIN

Abstract

We present a general scheme CFLP(X) for first order constraint functional logic programming which plays, with respect to lazy functional logic languages with constructor discipline, a similar role to the well known of CLP(X) with respect to logic programming. In CFLP(X), over a base structure equipped with a set of predefined functions and predicates, we define new ones by means of "constrained conditional rewrite rules". We formulate a declarative semantics in a general setting, where base structures are Scott domains and functions and predicates are continuous, and we obtain a complete characterization of the minimal model of a program as the least fixpoint of an associated operator. Finally, we propose a sound operational semantics, lazy constrained narrowing, which is complete for semantically non ambiguous programs.

Contents

1. Introduction and Related Work.
2. Constraint Structures.
3. Defining Functions and Predicates over a Constraint Structure.
4. Declarative Semantics of CFLP(X) Programs.
5. Operational Semantics. Soundness and Completeness Results.
6. Conclusions and Future Work.
7. References.

1. Introduction and Related Work.

Combination of different programming paradigms is one of the main research topics in the field of declarative programming. Many works have been successfully developed since late 80's related to two concrete cases: logic programming (LP) + constraint programming (CP), and functional programming (FP) + logic programming. Our work involves these three paradigms.

The notion of constraint logic programming (CLP) became well established in the works of Jaffar and Lassez [Jaf&Las86,87]. They defined a general scheme, CLP(X), which can be instantiated to different logic languages by adopting different constraint systems X. The key idea of CLP is that the fundamental semantic properties of logic programs – existence of least model, and its least fixpoint characterization – are due to the fact that programs are made up from *definite* clauses, and not so much to the use of the Herbrand universe as the underlying structure for doing computations. Some other structures can be considered instead of Herbrand universe, unification being replaced by constraint solving in the base structure. The semantic properties of logic programs still remain within this generalization, which enhances the expressive power of LP, and can additionally profit from efficient constraint solvers known for concrete domains.

This approach was further abstracted in [Hoe&Smo88] where the process of defining a logic language is seen as a hierarchical construction of constraint systems and the conditions

[1]This work has been partially supported by the spanish projects TIC89/0104 and U.P.M. Acción Concertada "Diseño de un lenguaje simbólico numérico"

imposed on constraint structures are relaxed. Since then, a great number of works related to CLP have appeared, devoted to semantic questions, new generalizations, implementation of concrete systems, applications,...

On the other hand, the combination FP + LP has also awaken a great interest in recent years [Gro&Llo86] [Bel&Lev86]. Among the family of functional + logic languages we may cite K-LEAF [Lev&al87] and BABEL [Mor&Rod88,89] as the most related to our work from the point of view of declarative semantics. They are first order languages which use conditional rewriting rules and follow a constructor discipline. In the case of BABEL, lazy narrowing [Red85] is adopted as operational semantics.

The combination FP + CP has been far less investigated. In [Dar&Guo89] a modification of the [Hoe&Smo88] approach is proposed as a basis for constraint functional programming (CFP), but functions are nondeterministic and the use of functional syntax is quite restricted. Other recent approaches attempt to set up constrained versions of rewriting and equational deduction [Dar&Guo90] [Kir&al90], which might be also used as a basis for CFP, but they do not support some important features of modern functional languages, as lazy evaluation for programming with infinite objects or higher order functions. Furthermore, these approaches do not seem well suited for achieving that purposes in a natural way.

As far as we know, the only proposal for constraint functional logic programming (CFLP) is [Dar&al91a,b], where a scheme CFLP(X) for defining CFL languages over arbitrary constraint structures is presented. Roughly speaking, they define CFLP(X) as CLP(FP(X)), that is, as an instance of CLP(X) where the base structure X is enriched with new functions defined by means of a functional language, and where the constraints include equations between FP(X) expressions, to be solved by narrowing. This approach does not allow to constraint higher order functions or to use constraints in function definitions. Moreover, we feel that the semantic properties of the scheme are not definitely well established.

In a previous work [Lop&Rod91], we made a first approach to the combination FP + LP + CP, restricted to first order features. We intended to achieve a "smooth" extension of FLP as CLP(X) is with respect to LP. By "smooth" we mean that substantial changes in the semantic notions and the methodology to be followed are not needed. In fact - again as in the case of CLP - we think that many questions admit a clearer treatment in this setting, mainly due to the fact that many technical details related to unification are avoided. In that work we proposed a fixpoint declarative semantics for CFLP(X) programs, but the operational semantics was restricted to the case of flat cpo's as basic domains and strict functions and predicates over them, for which we proved completeness.

In this work we extend these results, the main novelty being the proposal of an operational semantics, lazy constrained narrowing, covering the general case of non flat domains and lazy functions and predicates.

In section 2 we introduce the basic notions about the base structures in which computations will be carried out. As in [Jaf&Las86,87] we require structures to be parameterized by a signature constituted by symbols for predefined functions and predicates. We also require carriers of structures to be Scott domains [Sco82], and predefined operations and predicates to be continuous functions (in the sense of domain theory).

In section 3 we establish how to define new functions and predicates, by means of *constrained conditional rewrite rules* resembling the conditional rewrite rules of functional logic languages. *Constraint solving* replaces unification, which makes no sense in this general framework. As we do not address any treatment of logical negation, the rules (clauses) for defining predicates are constrained conditional rules determining when they are true. At the end of this section, two examples show how FLP and CLP are expressible in our scheme.

Section 4 is devoted to declarative semantics of CFLP(X) programs. Functions and predicates are interpreted as continuous functions over the base domain X, and the notion of model is introduced. After this, we prove the most important result of this first part, in which we give a complete characterization of the least model of a program as the least fixpoint of

an associated operator. The standard approach known for LP [Apt90] and FLP [Lev&al87] [Mor&Rod88,90] is followed, except for one interesting detail: we do not require programs to satisfy any additional non ambiguity conditions in contrast to K-LEAF [Lev&al87] or BABEL [Mor&Rod88,90]. Only the hypothesis of consistence (i.e. existence of one model) is needed for obtaining this result.

In section 5 we present "constrained narrowing" as operational semantics for CFLP(X), which we prove sound. For proving completeness, we introduce a new signature whose terms can be understood as approximations of expressions of the original language. Over these terms we define a simplification ordering which allows us to characterize laziness of computations, and to prove completeness for this lazy narrowing.

This paper contains quite detailed proofs of all the included results. Additional technical details can be found in [Lop92a].

2. Constraint Structures.

For simplicity we will work with structures over a one-sorted signature. A *signature* is a set of symbols Σ, each of them with an associated *arity* $n \geq 0$. We assume that Σ is partitioned as $\Sigma = F \cup \Pi$, where F is the set of *function symbols* and Π the set of *predicate symbols*. We assume also a numerable set V of *variables*.

$\Sigma = F \cup \Pi$ and V define in the usual way the following syntactic domains

$t \in T_F$	ground terms (or ground base expressions)
$t \in T_F(V)$	terms (or base expressions)
$c \in B_{\Pi,F}$	ground base atoms, $c \equiv p(t_1,\ldots,t_n)$, $p \in \Pi$, $t_i \in T_F$
$c \in B_{\Pi,F}(V)$	base atoms, $c \equiv p(t_1,\ldots,t_n)$, $p \in \Pi$, $t_i \in T_F(V)$

Definition 2.1: Given a signature $\Sigma = F \cup \Pi$, a *Σ-constraint structure* R consists of

i) A Scott domain DR

ii) An assignment, for each $f \in F$ of arity n, of a continuous function

$f_R : DR \times \overset{(n)}{\ldots} \times DR \longrightarrow DR$

iii) An assignment, for each $p \in \Pi$ of arity n, of a continuous function

$p_R : DR \times \ldots \times DR \longrightarrow (true, \perp_{bool})$, where $true, \perp_{bool} \notin DR$.

DR should be understood as the base domain where computations are to be performed. Symbols of Σ are interpreted through f_R's and p_R's as *predefined* functions and predicates which we know how to evaluate in DR. Note that continuity of p_R means that $p_R(x_1,\ldots,x_n) = true$ iff $p_R(u_1,\ldots,u_n) = true$ for some finite $u_i \leq x_i$. We also say that p_R represents a *continuous relation*. Since we do not deal with negation, we can interpret $\perp_{bool}$ as the truth value false, in contrast to the 3-valued approach in [Kun87].

In the rest of the paper we assume a fixed signature Σ and a Σ-structure R over it, and therefore references to them will be frequently omitted.

A *valuation* α is a mapping $\alpha: V \longrightarrow DR$. Every valuation α can be extended to $T_F(V)$ and $B_{\Pi,F}(V)$. The set VAL of valuations can be equipped with the usual ordering ($\alpha_1 \leq \alpha_2$ iff $\alpha_1(X) \leq \alpha_2(X)$ for all $X \in V$), and becomes a Scott domain.

We say that a valuation α *satisfies* $c \in B_{\Pi,F}(V)$ if $\alpha[c] = true$.

A *base constraint* π is a finite multiset $c_1,\ldots,c_n$ of base atoms $c_i \in B_{\Pi,F}(V)$.

We say that a valuation α *satisfies* $\pi \equiv c_1,\ldots,c_n$ if α satisfies c_i for each i. A constraint π is *satisfiable* if there exists some α satisfying it.

We must assume, for the language we are defining being interesting, that satisfiability of constraints in R(Σ) can be effectively checked. We will not insist in the existence of "solved forms" for constraints, because it affects mainly to refinements of operational semantics that we will not consider here.

3. Defining new Functions and Predicates over a Constraint Structure.

Given Σ, R(Σ), let us consider another signature Σ'= F'∪ Π', where Σ ∩ Σ'= ∅. The elements of F' and Π' must be understood as symbols for *defined* functions and predicates. These new symbols enlarge the syntactic domains to

$e \in T_{F\cup F'}(V)$ expressions

$b \in B_{\Pi\cup\Pi',F\cup F'}(V)$ boolean expressions or atoms

If there is no confusion we will use *expression* for elements from $T_{F\cup F'}(V) \cup B_{\Pi\cup\Pi',F\cup F'}(V)$.

A *rule* for $f \in F'$ takes the form $f(X_1,...,X_n) = e \leftarrow \pi \square \delta$

where: • $X_1,...,X_n$ are distinct variables • $\pi = c_1,...,c_n$ where $c_i \in B_{\Pi,F\cup F'}(V)$

• $e \in T_{F\cup F'}(V)$ • $\delta = b_1,...,b_l$ where $b_i \in B_{\Pi',F\cup F'}(V)$

$f(X_1,...,X_n)$ is the *head* or *lefthand side* of the rule, e is the *righthand side* and $\pi \square \delta$ is the *constraint*. Both π and δ are meant to be multisets, and may be empty.

A *clause* for $p \in \Pi'$ takes the form $p(X_1,...,X_n) = \text{true} \leftarrow \pi \square \delta$

where Xi's, π, δ are like in the rules for function symbols.

A *program* P is a set of rules and clauses. As we will see later there is no need of imposing additional conditions to programs in order to prove the existence of a least model, if the program has models at all.

Example 3.1: Functional Logic Programming over Herbrand domains (CFLP(*H*))

Lazy functional programming as provided by, e.g., BABEL, may be expressed in this framework in the following way:

Given a set of constructors $CS = \bigcup CS^n$ (CS^n = set of constructors of arity n) we may consider the signature Σ = F ∪ Π, where

• $F = CS \cup \{c\_k: c \in CS^n, 1 \le k \le n, n \in \mathbb{N}\}$ • $\Pi = \{is\_c: c \in CS\} \cup \{==\}$

The base domain would be the (infinitary) Herbrand domain generated by CS and a bottom element ⊥ [Lev&al87] [Mor&Rod88,89]. The constructors $c \in CS$ should be interpreted as free functions, and the symbols *c_k* would act as "selector functions", so that

$$c\_k(x) = \begin{cases} t_k \text{ if } x = c(t_1,..,t_k,..,t_n) \\ \\ \bot \text{ otherwise} \end{cases}$$

The predicate symbol == should be interpreted as strict equality, i.e., is true for equal finite and total objects, and ⊥ for the rest.

The symbols *is_c* would be interpreted as the predicates

$$is\_c(x) = \begin{cases} \text{true if } x = c(t_1,\dots,t_n) \\ \\ \bot \quad \text{otherwise} \end{cases}$$

Remark that all the predefined operations are continuous.

The functions *c_k* and *is_c* correspond to two roles of unification: selection of alternatives (applicability conditions) and access to components.

For example, the function which returns the first N elements of a list, which might be written in a functional language as

```
first(0,Xs) = nil.
first(s(N),cons(X,Xs)) = cons(X,first(N,Xs)).
```

could be defined in CFLP(*H*) as

```
first(N,Xs) = nil ← is_0(N) □.
first(N,Xs) = cons(cons_1(Xs),first(suc_1(N),cons_2(Xs))
              ← is_suc(N) , is_cons(Xs) □.
```

We could apply this function to any list, even if it is an infinite one. Infinite lists can be obtained by means of function definitions as, for example, the infinite list of 0's defined by `l = cons(0,l) ← □.`

Example 3.2: Constraint Logic Programming (CLP).

CLP(X) programs, as considered in [Jaf&Las86,87], are also expressible in CFLP(X). In CLP(X) a base signature Σ = F∪Π (Π includes the symbol ==), and a base structure *R* are assumed, and only new predicates can be defined through clauses of the form $p(t_1,...,t_n) :- \pi \square \delta$.

We only need to lift *R* with a bottom element to get a flat cpo $R_\perp$, to rewrite the clauses to the form $p(X_1,...,X_n) = \text{true} \leftarrow X_1 == t_1,...,X_n == t_n , \pi \square \delta$, to interpret == as strict equality over $R_\perp$, and to replace all the predefined predicates and functions by their strict extensions. But observe that in our scheme we get more expressive power, since we are able to define not only new predicates, but also functions.

Note that this example shows in particular how to realize PROLOG II as an instance of our scheme. Observe that the constraint structure *T* of this instance contains finite and infinite regular trees (but not partial ones, with the exception of the bottom element added by lifting). The constraint structure *H* from example 3.1 cannot be used to realize PROLOG II, since strict equality over *H* never holds between infinite trees. Of course, CFLP(*T*) is more expressive than PROLOG II, because user defined functions are allowed.

Remark that in CFLP(*T*) all regular trees (even infinite ones) are finite elements of the carrier domain (in the sense of domain theory), and that constructors are strict. As a consequence, infinite trees can be defined only as solutions of constraints, but not as the result of function application. For example, the function `l= cons(0,l) ← □` of the example 1 would not denote in CFLP(T) an infinite list of 0's; since *cons* is strict in this case, the unfolding of l's recursive definition will remain ⊥. In CFLP(*T*) we can use an equality constraint for defining the infinite list l through a predicate, `l(L) ← L == [0|L] □.`

4. Declarative Semantics of CFLP(X) Programs.

4.1. Interpretations. Models of rules and programs

An *interpretation* I assigns to every $f \in F'$ ($p \in \Pi'$ resp.) a continuous function f_I (p_I resp.) with target DR ({true, $\perp_{bool}$} resp.). The set INT of all interpretations can be equipped with the usual ordering: $I_1 \le I_2$ iff $f_{I_1}(a_1,...,a_n) \le f_{I_2}(a_1,...,a_n)$ for all $f \in \Sigma'$, $a_i \in DR$. With this ordering INT becomes a Scott domain.

Given $I \in INT$, every valuation α can be extended to $T_{F\cup F'}(V)$ and $B_{\Pi\cup\Pi',F\cup F'}(V)$. We write $\alpha[e]_I$ for the value of e under I and α. Also the notion of satisfiability can be extended to cover the case of multisets of atoms $b \in B_{\Pi\cup\Pi',F\cup F'}(V)$.

We will use, sometimes implicitely, the following standard property .

Proposition 4.1 (Continuity of evaluation): for each expression e, the function

ev_e: VAL × INT ⟶ DR (or {true,$\perp_{bool}$}) defined by $ev_e(\alpha,I) = \alpha[e]_I$ is continuous.

Definition 4.1: I ∈ INT is a *model of a rule* $f(X_1,...,X_n) = e \leftarrow \pi \square \delta$ iff $f_I(\alpha(\bar{X})) \geq \alpha[e]_I$ for every valuation α which satisfies $\pi \square \delta$ in I. This definition also stands for clauses seen as functional rules. In this case f = p∈Π' and e = true, and so $p_I(\alpha(\bar{X}))$ = true is required.

Remark that with this definition rules are seen as "constrained inequations" and not as "constrained equations". This corresponds naturally to the intended operational reading of the rules as "constrained rewrite rules", which will be reflected in the definition of the operator T_P associated to a program (see def. 4.2 below) and in the formulation of the operational semantics (see section 5). The example 4.1 at the end of the section illustrates how the definition of models affects the declarative semantics of CFLP(X) programs.

In the rest of the section we do not need to distinguish functions and predicates, so we will use f for symbols of $\Sigma' = F' \cup \Pi'$.

An interpretation is a *model of a program* if it is model of all its rules. A program is *consistent* if it has some model.

The following notation will be used frequently in the sequel.

Notation: Given I ∈ INT, f ∈ Σ', $a_1,...,a_n$ ∈ DR, we write

$C(I,f,\bar{a}) = \{\ \alpha[e]_I : f(\bar{X}) = e \leftarrow \pi \square \delta \in P,\ \alpha|_{\bar{X}} = \bar{a},\ \alpha \text{ satisfies } \pi \square \delta \text{ in I}\ \}$.

Proposition 4.2.: I is a model of P iff for every f ∈ Σ', $\bar{a} \in DR^n$, there exists lub $C(I,f,\bar{a}) \leq f_I(\bar{a})$.

Proof: →) As I is a model of P, $f_I(\bar{a}) \geq x$ holds for each $x \in C(I,f,\bar{a})$. Hence $C(I,f,\bar{a})$ has a lub, because it is bounded (recall that DR is a Scott domain).

←) Even easier.■

We say that I is an *exact model* of P if $f_I(\bar{a})$ = lub $C(I,f,\bar{a})$ for all f, $\bar{a}$.

4.2. Existence of least model

We will prove here that every consistent program (the definition of programs does not ensure consistency) has a least model, which is reachable by iteration of an associated operator.

Definition 4.2 (T_P operator): given a program P, we define (partially over INT) the operator T_P: INT ⟶ INT as follows: given I ∈ INT, f ∈ Σ', $\bar{a} \in DR^n$,

$$f_{T_P(I)}(\bar{a}) = \text{lub } C(I,f,\bar{a}).$$

We regard $T_P(I)$ as defined if this lub exists for all f, $\bar{a}$. Note that T_P might be not defined for some interpretations. It is not worth to set $T_P(I) = \perp$ in this cases, since T_P would not even be monotonic.

Notation: $T_P\uparrow 0 = \perp_{INT}$, $T_P\uparrow k+1 = T_P(T_P\uparrow k)$, $T_P\uparrow\omega = \bigcup_k T_P\uparrow k$, if they are defined.

Lemma 4.1 (monotonicity of T_P): T_P is monotonic, in the following sense: if $I_1 \leq I_2$ and $T_P(I_2)$ is defined, then $T_P(I_1)$ is also defined and $T_P(I_1) \leq T_P(I_2)$.

Proof: it is easy to check that, for every f ∈ Σ', $\bar{a} \in DR^n$, each member of $C(I_1,f,\bar{a})$ is bounded by some member of $C(I_2,f,\bar{a})$. Therefore $C(I_1,f,\bar{a})$ is bounded (so has a lub) by lub $C(I_2,f,\bar{a})$ (which exists since $T_P(I_2)$ is defined), and moreover we have

$f_{T_P(I_1)}(\bar{a}) = \text{lub } C(I_1,f,\bar{a}) \leq \text{lub } C(I_2,f,\bar{a}) = f_{T_P(I_2)}(\bar{a})$.■

Lemma 4.2: I is a model of P iff $T_P(I)$ is defined and $T_P(I) \leq I$. In this case, $T_P(I)$ is also a model of P.

Proof: The first part is just another statement of proposition 4.2. For the rest, observe that $T_P(I) \leq I$ implies, by monotonicity, that $T_P(T_P(I))$ is defined and $T_P(T_P(I)) \leq T_P(I)$; by the first part of the lemma, $T_P(I)$ is a model of P.■

The main result of this section is the following

Theorem 4.1: P is consistent (i.e. has a model) iff $T_P\uparrow\omega$ exists. Moreover, in this case $I_M = T_P\uparrow\omega$ is the least model of P, the least fixpoint of T_P, and it is an exact model.

Proof: →) Let I any model of P (there exists someone by hypothesis). We first prove by induction that for any k ≥ 0, $T_P\uparrow k$ is defined and verifies $T_P\uparrow k \leq I$. The case k=0 is trivial, since $T_P\uparrow 0 = \perp_{INT}$. Now assume that $T_P\uparrow k \leq I$; since $T_P(I)$ is defined and $T_P(I) \leq I$ by lemma 4.2, we may conclude by monotonicity of T_P that $T_P\uparrow k+1 = T_P(T_P\uparrow k)$ is also defined and $T_P\uparrow k+1 \leq T_P(I) \leq I$, as we wanted.

Using again monotonicity it is very easy to prove that the iterations $T_P\uparrow k$ form a chain $T_P\uparrow 0 \leq T_P\uparrow 1 \leq ... \leq I$. Then there exists $I_M = T_P\uparrow\omega = \bigcup_k T_P\uparrow k$. Remark that I_M also verifies $I_M \leq I$.

←) We prove that P is consistent by proving that $I_M = T_P\uparrow\omega$ is itself a model of P.

Let $f(\bar{X}) = e \leftarrow \pi \square \delta$ a rule from P, and α a valuation satisfying $\pi \square \delta$ in I_M. By continuity of evaluation, there exists some k_0 such that α satisfies $\pi \square \delta$ in $T_P\uparrow k$ for all $k \geq k_0$, and by the definition of T_P, $\alpha[e]_{T_P\uparrow k} \leq f_{T_P\uparrow k+1}(\alpha(\bar{X})) \leq f_{I_M}(\alpha(\bar{X}))$. Hence, again by continuity of evaluation, $\alpha[e]_{I_M} = \bigcup_{k \geq k_0} \alpha[e]_{T_P\uparrow k} \leq f_{I_M}(\alpha(\bar{X}))$, that is, I_M is a model of the rule.

That I_M is the least model of P is clear as we have proved $I_M \leq I$, for every model I of P. I_M is also a fixpoint of T_P because $T_P(I_M) \leq I_M$ (since I_M is a model of P) and $I_M \leq T_P(I_M)$ (since $T_P(I_M)$ is also a model of P by lemma 4.2). But then I_M is the least fixpoint of T_P, because every fixpoint of T_P is obviously a model of P by lemma 4.2.

Finally, notice that $f_{I_M}(\bar{a}) = f_{T_P(I_M)}(\bar{a})$ = lub $C(I_M,f,\bar{a})$, hence I_M is an exact model of P.■

Consistency of a given program P (and henceforth, existence of $T_P\uparrow\omega$), can be proved as undecidable. Although the previous theorem is interesting, because it explains precisely the relation between existence of models and T_P, some restrictions must be imposed to programs if we want all of them to have models. The easiest choice is to force T_P to be totally defined. The following condition ensures that.

Definition 4.2.: A program P is said to be *non ambiguous* iff for any pair of variants of rules for f ∈ Σ', $f(X_1,...,X_n) = e \leftarrow \pi \square \delta$ and $f(X_1,...,X_n) = e' \leftarrow \pi' \square \delta'$, such that variables not in the head are renamed apart, the following condition holds: $\alpha[e]_I = \alpha[e']_I$ for every I ∈ INT and α ∈ VAL satisfying π,π' in I. The pair of variants might be of the same rule.

This condition implies that the sets $C(I,f,\bar{a})$ can only be of the form ∅, {x} or {x,⊥} for some x ∈ DR, and hence have a lub (a maximum, in fact). Remark that the stated condition is still quite ineffective. If we consider "logic" CFLP(X) programs, in which only clauses for defining predicates are used, it is obvious that every program is consistent. The theorem 4.1 particularizes in this case to the usual theorems for CLP [Jaf&Las86],[Ho&Smo88] of

existence of least model as least fixpoint, which were in turn generalizations of the classical one [Emd&Kow76] for LP.

The works from Jaffar and Lassez, and some other as [Gab&Lev91], present a wider range of operators associated to a program, which reflect different aspects related to the behavior of the program. It seems quite plausible that similar treatments will be possible in our framework.

Example 4.1:

This example shows how the inequational meaning of the rules for defining functions and predicates affects our declarative semantics.

Let us consider the following program over $\mathbf{Z}_\perp$

$f(X) = 1 \Leftarrow X = 0 \;\square\; .$
$f(X) = f(X-1) \Leftarrow \square\; .$

In the least model $I_M = T_P\uparrow\omega$ we have $f_{I_M}(z) = \begin{cases} 1 \text{ if } z \geq 0 \\ \perp \text{ if } z < 0 \end{cases}$

Note that the first two rules overlap for z = 0. In the fist one we have $f_{I_M}(0) = 1$; in the second one $1 = f_{I_M}(0) \geq f(0-1) = \perp$.

An alternative notion of model could be obtained by requiring each single rule to be satisfied as a conditional equation. Also with respect to this new notion of model, this program would have a least model I_M' given by

$$f_{I_M'}(z) = \begin{cases} 1 \text{ if } z \in \mathbf{Z} \\ \perp \text{ if } z = \perp \end{cases}$$

Note, however, that I_M' cannot be obtained as $T_P\uparrow\omega$.

5. Operational semantics

A *goal* takes the form $e \Leftarrow \pi \;\square\; \delta$ (where e is an expression or true), that is, like the righthand side of a rule. A goal is *partially finished* if δ is a multiset consisting only of true elements (we will say that δ is *solved* for this situation). G is *finished* if it is partially finished and e,π contain only predefined symbols.

Computations should be a generalization of those performed through narrowing, replacing unification by constraint satisfaction. We can use the term *constrained narrowing* for the computation mechanism to be defined.

In this section we will use the notation G[e'] for indicating that e' is a subexpression of either e,π or δ in the goal G. Also we will write $G \cup (\pi' \square \delta')$ for expressing the result of adding π' and δ' to the constraint part of G, that is, $(e \Leftarrow \pi \;\square\; \delta) \cup (\pi' \;\square\; \delta')$ denotes $e \Leftarrow \pi,\pi' \square \delta,\delta'$. Recall that π's and δ's are seen as multisets, and so the order of their components is irrelevant.

The *one-step constrained narrowing relation*, $e \Leftarrow \pi \;\square\; \delta \vdash e' \Leftarrow \pi' \square \delta'$, is defined by the rule

R1 (*Constrained narrowing*)

$G[f(e_1,\ldots,e_n)] \vdash G[e\sigma] \cup (\pi\sigma \;\square\; \delta\sigma)$
if $f \in F' \cup \Pi'$ (f is a defined function or predicate), and $\sigma = \{X_1 \leftarrow e_1,\ldots,X_n \leftarrow e_n\}$, where $f(X_1,\ldots,X_n) = e \Leftarrow \pi \;\square\; \delta$ is a variant of a rule (with new variables).

A *computation* starting by a goal G is a sequence of goals $G_0,G_1,\ldots,G_n$ such that $G = G_0 \vdash G_1 \vdash \ldots \vdash G_n$. We use $\vdash^*$ for the reflexive and transitive closure of $\vdash$.

Constrained narrowing, as stated here, seems to be a very unrestricted computation rule, leading then to very large search spaces. But we will be able, while studying completeness, to restrict the use of the rule R1 for reducing only on positions in a goal which really need to be reduced. We will give an (ineffective) characterization of such positions, and we will prove completeness for this "lazy constrained narrowing" for non ambiguous programs. But we turn first to the problem of soundness of constrained narrowing.

5.1. Soundness of constrained narrowing

A *solution* for a goal $e \Leftarrow \pi \;\square\; \delta$ in an interpretation I is a pair (α,x), where α is a valuation and $x \in DR$, such that α satisfies $\pi \;\square\; \delta$ in I and $\alpha[e]_I \geq x$. (α,x) is *finite* iff x is finite.

Remark that if (α,x) is a solution of a goal G in I, and α' verifies $\alpha'|_{var(G)} = \alpha|_{var(G)}$, then also (α',x) is solution of G in I.

We will also use the following easy consequence of the standard *substitution lemma*.

Lemma 5.1: if $\alpha(X_i) = \alpha[e_i]_I$, i=1,...,n, then for every expression e, $\alpha[e]_I = \alpha[e\sigma]_I$, where $\sigma \equiv \{X_i \leftarrow e_i\}$. As a consequence, α satisfies $\pi \;\square\; \delta$ in I iff α satisfies $\pi\sigma \;\square\; \delta\sigma$ in I.

Theorem 5.1. (Soundness of constrained narrowing): For any model I and any computation $G \vdash^* G'$, if (α,x) is a solution of G' in I, then (α,x) is a solution of G in I.

Proof: Obviously it suffices to prove soundness of one step computations. The complete proof for one step $G \vdash G'$ should distinguish also between different cases corresponding to the place where the subexpression to be reduced occurs. We include only the proof for reducing a subexpression in the e-part of the goal.

Let us consider a step $e[f(e_1,\ldots,e_n)] \Leftarrow \pi \;\square\; \delta \;\vdash\; e[e'\sigma] \Leftarrow \pi'\sigma,\pi \;\square\; \delta'\sigma,\delta$, where $f \in F'$ and there is a variant of a rule (with new variables)

$f(X_1,\ldots,X_n) = e' \Leftarrow \pi' \square \delta'$, with $\sigma \equiv \{X_1 \leftarrow e_1,\ldots,X_n \leftarrow e_n\}$

Let (α,x) be a solution of G' in a model I. Then

(1) α satisfies both $\pi \;\square\; \delta$ and $\pi'\sigma \;\square\; \delta'\sigma$ in I

(2) $\alpha[e[e'\sigma]]_I \geq x$

Now, let $\alpha'(X) = \begin{cases} \alpha[e_i]_I \text{ if } X = X_i \\ \alpha(X) \text{ otherwise} \end{cases}$

Remark that α and α' concide over any expression not involving the Xi's. Hence, by (1), α' satisfies $\pi'\sigma \;\square\; \delta'\sigma$ in I. As we also have $\alpha[e_i]_I = \alpha'[e_i]_I$, then α' satisfies $\pi' \square \delta'$, by lemma 5.1. Therefore, as I is a model of the rule, $f_I(\alpha'(X_1),\ldots,\alpha'(X_n)) \geq \alpha'[e']_I$.

But $f_I(\alpha'(X_1),\ldots,\alpha'(X_n)) = f_I(\alpha[e_1]_I,\ldots,\alpha[e_n]_I) = \alpha[f(e_1,..,e_n)]_I$, and again by lemma 5.1 $\alpha'[e']_I = \alpha'[e'\sigma]_I = \alpha[e'\sigma]_I$.

Then $\alpha[f(e_1,..,e_n)]_I \geq \alpha[e'\sigma]_I$, and hence $\alpha[e[f(e_1,..,e_n)]]_I \geq \alpha[e[e'\sigma]]_I \geq x$ by (2). This condition, together with (1), guarantees that (α,x) is a solution of G in I. ■

5.2 Completeness of constrained narrowing

Let us briefly discuss which kind of completeness result we can expect. First, since the semantic domain may contain infinite values, we should only look for computing finite approximations to a solution of a goal.

A different problem is the following one: the operator T_P is defined through the lub of sets of results obtained by applying different program rules for a defined function. In operational terms it means that several branches of the search space may contribute to the final denotation of an expression. Therefore, even a finite approximation to some denotation could not be reachable by a single computation. For this reason, we prove here completeness only for programs satisfying the additional following condition: T_P verifies $f_{T_P(I)}(\bar{a}) = \max C(I,f,\bar{a})$ (instead of lub, as in 4.1.). We say that these programs are *deterministic*. Note that non ambiguous programs in the sense of section 4 are deterministic.

We first introduce a notion which serves to express "how far" we have evaluated an expression.

Definition 5.1: The *shell* $|e|$ of an expression $e \in T_{\Sigma\cup\Sigma'}(V)$ is defined as

- $|X| = X$, if $X \in VAR$
- $|c(e_1,\ldots,e_n)| = c(|e_1|,\ldots,|e_n|)$, if $c \in \Sigma$
- $|f(e_1,\ldots,e_n)| = \perp$, if $f \in \Sigma'$

This notation extends to constraints in the natural way.

It is obvious that for any valuation α and any interpretation I it holds $\alpha[e]_I \geq \alpha[|e|]$, where $\alpha[|e|]$ is defined in a natural way as follows:

$\alpha[|X|] = \alpha(X)$, if $X \in VAR$

$\alpha[|c(e_1,\ldots,e_n)|] = c_R(\alpha[|e_1|],\ldots,\alpha[|e_n|])$, if $c \in \Sigma$

$\alpha[|f(e_1,\ldots,e_n)|] = \perp$, if $f \in \Sigma'$.

We can now give a first statement of the completeness result we are looking for.

Theorem 5.2 (Completeness of constrained narrowing): Given a deterministic program P, and a finite solution (α,x) of a goal G in the least model I_M, there exists a computation $G \vdash^* G'$ and a valuation α' such that

i) G' is partially finished, i.e., $G' \equiv e' \Leftarrow \pi' \square \delta'$, with δ' solved

ii) $\alpha'|_{var(G)} = \alpha|_{var(G)}$

iii) α' satisfies $|\pi'|$

iv) $\alpha'[|e'|] \geq x$.

We will prove indeed a much stronger result, where only a restricted kind of computations, namely lazy computations, are allowed. In order to define a suitable notion of laziness and to prove the theorem, we need to introduce some preliminary notions. We will use a well-founded ordering over a set of special terms. For standard notions about term orderings we refer e.g. to [Der&Jou90].

Given Σ,Σ' of predefined and defined symbols, we consider a new set of symbols $\Sigma'' = \{f_k : f \in \Sigma', k \geq 0\}$ (the indices k are introduced in order to reflect the iterations of the T_P operator) .

Over $\Sigma \cup \Sigma''$ we define a *precedence ordering* $>$ as follows:

- $f_k > c$ for all $f_k \in \Sigma''$, $c \in \Sigma \cup \{true\}$
- $f_k > g_{k'}$ for all $f,g \in \Sigma'$, $k > k'$.

$>$ is a well-founded ordering that induces an ordering ($<_{RPO}$) in $T_{\Sigma\cup\Sigma''}(V)$, defined by

i) $X <_{RPO} t$, for all $X \in V$, $t \in T_{\Sigma\cup\Sigma''}(V)$, $t \notin V$.

ii) $s \equiv f(s_1,\ldots,s_n) <_{RPO} g(t_1,\ldots,t_m) \equiv t$ iff one of the following conditions hold:

a) $s \leq_{RPO} t_i$, for some i

b) $f < g$, and $s_i <_{RPO} t$, for all $i=1,\ldots,m$

c) $f \equiv g$, and $\{s_1,\ldots,s_n\} \ll \{t_1,\ldots,t_m\}$, where $\ll$ is the *multiset extension* of $<_{RPO}$, defined as the smallest ordering satisfying $S \cup \{t_1,\ldots,t_n\} \ll S \cup \{s\}$, for every multiset S of terms $\in T_{\Sigma\cup\Sigma''}(V)$, every $s \in T_{\Sigma\cup\Sigma''}(V)$, and every multiset $\{t_1,\ldots,t_n\}$ with $n \geq 0$ and $t_i <_{RPO} s$ for all $i = 1,\ldots,n$.

This ordering is a slight variant of a standard simplification ordering, the multiset recursive path ordering [Der&Jou90]. Notice that we require variables to be minimal elements with respect to this ordering.

We call *labeled expressions* to the elements of $T_{\Sigma\cup\Sigma''}(V)$.

A *labeled goal* is of the form $e \Leftarrow \pi \square \delta$, where e and each of the components of π and δ are labeled expressions. For the purpose of providing an ordering for labeled goals, we consider a labeled goal $e \Leftarrow \pi_1,\ldots,\pi_n \square \delta_1,\ldots,\delta_n$ as the multiset $\{e,\pi_1,\ldots,\pi_n,\delta_1,\ldots,\delta_n\}$. With this reading, the ordering for labeled goals is the relation $\ll$ introduced above.

We will use labeled expressions and labeled goals for expressing approximations to expressions and goals. More precisely, we give the following

Definition 5.2:

a) Let $e \in T_{\Sigma\cup\Sigma'}(V)$ (an expression) and $e' \in T_{\Sigma\cup\Sigma''}(V)$ (a labeled expression). We say that e and e' have *the same shape* , or that e' is an *approximation* to e, if they are identical when ignoring the indices in e'. The definition extends naturally to goals.

b) The *value* $\alpha[e]$ of a labeled expression $e \in T_{\Sigma\cup\Sigma''}(V)$ under a valuation α is

$\alpha[X] = \alpha(X)$, if $X \in VAR$

$\alpha[c(e_1,\ldots,e_n)] = c_R(\alpha[e_1],\ldots,\alpha[e_n])$, if $c \in \Sigma$

$\alpha[f_k(e_1,\ldots,e_n)] = f_{T_P\uparrow k}(\alpha[e_1],\ldots,\alpha[e_n])$, if $f \in \Sigma'$.

c) We say that a valuation α *satisfies* a labeled constraint $\pi_1,\ldots,\pi_n \square \delta_1,\ldots,\delta_m$ iff

$\alpha[\pi_i] = \alpha[\delta_j] = true$, for all $i = 1,\ldots,n$, $j = 1,\ldots,m$.

d) (α,x) is a *solution* of a labeled goal $e \Leftarrow \pi \square \delta$ iff α satisfies $\pi \square \delta$ and $\alpha[e] \geq x$.

e) Given a goal G and a solution (α,x) of G in I_M, we say that an approximation G' of G is *sufficient for witnessing* (α,x) iff (α,x) is also solution of G'.

Notation: • Σ''_k denotes the set $\{f_j \in \Sigma'' : j \leq k\}$

• Let $e \in T_{\Sigma\cup\Sigma'}(V)$; we note as $e^{(k)}$ the term of $T_{\Sigma\cup\Sigma''_k}(V)$ obtained by replacing each $f \in \Sigma'$ occurring in e by f_k. Obviously e and $e^{(k)}$ have the same shape. We use this notation also for constraints and goals. Remark that the minimal approximation to e in the sense of $<_{RPO}$ is $e^{(0)}$.

We next summarize in two lemmas some properties about the orderings $<_{RPO}$, $\ll$ and about approximations, which will be used for proving the completeness theorem. Most of them are quite standard results, and the rest are easily provable.

Lemma 5.2:

a) $<_{RPO}$ and $\ll$ are well-founded orderings

b) If $e <_{RPO} e'$, then $e_0[e] <_{RPO} e_0[e']$, for all $e_0,e,e' \in T_{\Sigma\cup\Sigma'',(V)}$

c) If $e <_{RPO} e' \equiv f(X_1,\ldots,X_n)$, then $e\sigma <_{RPO} e'\sigma$, for any substitution σ of labeled expressions for variables such that $dom(\sigma) \subseteq \{X_1,\ldots,X_n\}$

d) If $t \in T_{\Sigma\cup\Sigma''_k,(V)}$ and $k < k'$, then $t <_{RPO} f_{k'}(t_1,\ldots,t_n)$, for all $f \in \Sigma'$ and all $t_1,\ldots,t_n \in T_{\Sigma\cup\Sigma'',(V)}$

e) If t,t_i are all $<_{RPO} t'$ then $(s[t'],s_1,\ldots,s_m) \gg (s[t],t_1,\ldots,t_n,s_1,\ldots,s_m)$, for arbitrary $s_i \in T_{\Sigma\cup\Sigma'',(V)}$.

Lemma 5.3:

a) $\alpha[e]_{I_M} \geq \alpha[e']$ for all approximations e' of e. As a consequence, if (α,x) is a solution of an approximation G' to a goal G, then (α,x) is also solution of G in I_M (and of course, G' is sufficient for witnessing (α,x)).

b) If (α,x) is a finite solution of a goal G in I_M, then G has an approximation G' sufficient for witnessing (α,x).

c) $\alpha[|e|] = \alpha[e^{(0)}]$. As a consequence, α satisfies $|\pi|$ iff α satisfies $\pi^{(0)}$.

d) Let $G \equiv e \leftarrow \pi \sqcap \delta$ a goal and (α,x) a solution of G in I_M. Then $G^{(0)}$ is sufficient for witnessing (α,x) iff δ is solved, α satisfies $|\pi|$, and $\alpha[|e|] \geq x$.

We will also use the extension of lemma 5.1 for the case of labeled expressions, omitting in this case the references to I.

The main result for characterizing what "lazy constrained narrowing" should mean, and for subsequently proving completeness of lazy narrowing is the following

Lemma 5.4 (Complexity reduction): Let P be a deterministic program. Let $G_0[f(e_1,\ldots,e_n)]$ be a goal, (α,x) a solution in I_M and $G'_0[f_k(e'_1,\ldots,e'_n)]$ some minimal approximation of G_0 sufficient for witnessing (α,x), with $k > 0$. Then it is possible to apply some rule for f for reducing G_0 to a new goal G_1 verifying

i) (α',x) is a solution of G_1 for some α' coinciding with α over $var(G_0)$

ii) G_1 admits an approximation G'_1 sufficient for witnessing (α',x), such that $G'_1 \ll G'_0$

Proof: $\alpha[f_k(e'_1,\ldots,e'_n)]$ $(= f_{T_P\uparrow k}(\alpha[e'_1],\ldots,\alpha[e'_n])$ must be $\neq \perp$, since otherwise $G_0[f_0(e'_1,\ldots,e'_n)]$ (which is $\ll G'_0$) would also be a sufficient approximation for G_0 witnessing (α,x), and hence G'_0 would not be minimal. Then, by the definition of $T_P\uparrow k$, there must be a rule (with fresh variables) $f(X_1,\ldots,X_n) = e \leftarrow \pi \sqcap \delta$, and a valuation α' (which can be chosen to additionally verify $\alpha'|_{var(Go)} = \alpha|_{var(Go)}$), such that:

(1) $\alpha'(X_i) = \alpha[e'_i] (= \alpha'[e'_i])$

(2) α' satisfies $\pi \sqcap \delta$ in $T_P\uparrow k-1$

(3) $\alpha'[e]_{T_P\uparrow k-1} = \alpha[f_k(e'_1,\ldots,e'_n)]$. (we use the determinacy condition)

(2) and (3) may be reformulated as

(2') α' satisfies $\pi^{(k-1)} \sqcap \delta^{(k-1)}$

(3') $\alpha'[e^{(k-1)}] = \alpha[f_k(e'_1,\ldots,e'_n)]$.

By (1),(2'),(3') and lemma 5.1, we obtain

(4) α' satisfies $\pi^{(k-1)}\sigma' \sqcap \delta^{(k-1)}\sigma'$, where $\sigma' \equiv \{X_1 \leftarrow e'_1,\ldots,X_n \leftarrow e'_n\}$

(5) $\alpha'[e^{(k-1)}\sigma'] = \alpha[f_k(e'_1,\ldots,e'_n)]$.

Now, lemma 5.2d ensures $e^{(k-1)} <_{RPO} f_k(X_1,\ldots,X_n)$, and then, by lemma 5.2c, we get

(6) $e^{(k-1)}\sigma' <_{RPO} f_k(e'_1,\ldots,e'_n)$.

The same stands for all the components of $\pi^{(k-1)}\sigma' \sqcap \delta^{(k-1)}\sigma'$.

Therefore $G'_1 \equiv G'_0[e^{(k-1)}\sigma'] \cup \pi^{(k-1)}\sigma' \sqcap \delta^{(k-1)}\sigma'$ is a witness for (α',x) by (4),(5), and $G'_1 \ll G'_0$ by (6) and lemma 5.2.e.

To conclude the proof, just remark that applying the same rule to G_0 we get $G_1 \equiv G_0[e\sigma] \cup \pi\sigma \sqcap \delta\sigma$, where $\sigma \equiv \{X_1 \leftarrow e_1,\ldots,X_n \leftarrow e_n\}$, and hence G'_1 is an approximation of G_1. Since G'_1 witnesses (α',x), then (α',x) is a solution for G_1 in I_M, by lemma 5.3a.■

The lemma justifies the following

Definition 5.3:

a) Let $G[f(e_1,\ldots,e_n)]$ be a goal, (α,x) a solution of in I_M and $G'[f_k(e'_1,\ldots,e'_n)]$ a minimal approximation to G sufficient for witnessing (α,x). Then $f(e_1,\ldots,e_n)$ is a position which *demands evaluation* with respect to (α,x),G' iff $k > 0$.

b) We say that a constrained narrowing step $G_0 \vdash G_1$ is *lazy* with respect to some solution (α,x) of G_0 iff it has been performed by reducing in a position in G_0 which demands evaluations wrt some G'_0 witnessing for (α,x). A computation $G \vdash^* G'$ is lazy iff all its steps are lazy.

Therefore, the previous lemma simply states that we can perform a lazy constrained narrowing step on a goal G_0 to obtain a new goal G_1 which still admits the desired solution but is "closer to it". The lemma also reveals the *don't care indeterminism* of lazy constrained narrowing: no matter which position demanding evaluation we choose, we can reduce on it. We will use the previous lemma assuming that we choose a minimal G'_1 among the sufficient approximations to G_1.

We can prove now a completeness theorem stronger than the stated above.

Theorem 5.2 (Completeness of lazy constrained narrowing): Given a deterministic program P, and a finite solution (α,x) of a goal G in the least model I_M, there exists a lazy computation $G \vdash^* G'$ and a valuation α' which fulfil the same conditions stated in Th.5.1.

Proof

Let $G_0 \equiv G \equiv e \leftarrow \pi \sqcap \delta$. Since x is finite, there must be a k such that α satisfies $\pi\sqcap\delta$ in $T_P\uparrow k$, and $\alpha[e]_{T_P\uparrow k} \geq x$. Then G_0 admits $e^{(k)} \leftarrow \pi^{(k)} \sqcap \delta^{(k)}$ as a sufficient approximation, and hence G_0 has a minimal sufficient approximation G'_0. If G'_0 is $G_0^{(0)}$, then we have finished, since then lemma 5.3d ensures that δ is solved, α satisfies $|\pi|$ and $\alpha[|e|] = \alpha[e^{(0)}] \geq x$.

Otherwise, we can repeatedly apply the previous lemma to α,G_0,G'_0 to obtain a sequence α_1,G_1,G'_1, α_2,G_2,G'_2, ...,correspondig to a lazy computation $G_0 \vdash G_1 \vdash G_2$..., and verifying

$G'_0 \gg G'_1 \gg G'_2 \gg \ldots$. As $\gg$ is well-founded, we must reach some α_n, G_n, G'_n to which the lemma 5.4 is not applicable, that is, when $G'_n = G^{(0)}_n$. Again by lemma 5.3d, the desired computation is now finished.■

6. Conclusions and Future Work

We have developed a general framework for first order constraint functional logic programming, CFLP(X), which plays, with respect to functional logic languages as BABEL [Mor&Rod88,90], a similar role to the well known of CLP(X) [Jaf&Las86,87] with respect to logic programming. Our scheme includes the expressive power of both FLP and CLP.

We have formulated a declarative semantics for CFLP(X) programs over Scott domains as carriers of base structures, obtaining a complete characterization of the consistency and existence of least model for a program in terms of the least fixpoint of an associated operator.

We have presented an operational semantics, constrained narrowing, which can be seen as a generalization of narrowing. We have proved soundness of constrained narrowing. For deterministic programs we have formulated the notion of completeness that can be expected. We have defined a simplification ordering over approximations of the denotation of expressions in the least model, which allowed us to characterize laziness of narrowing steps, and to give a simple proof of completeness of lazy constrained narrowing, thus generalizing the completeness results of lazy narrowing for functional logic languages.

Of course, there is still much to do about CFLP, at least in our approach. Some further developments we have in mind are:

- Improvement of the operational semantics in order to have a more "practical" approach. Notions of "solved forms" (of constraints and expressions) and "failed computations" should be introduced.

- Implementation of some interesting instance of the scheme, like CFLP(H) over Herbrand domains with equality and disequality, based on existing works about abstract narrowing machines [Kuc&al90] [Loo91] [Mor&al90]. Some works of this kind are in progress [Kuc&al92][Lop92b].

- Extension of CFLP(X) to include higher order functions. It seems to be the hardest task because higher order features are not yet well integrated within FLP. We hope that some works along this line [Gon&al90,91] will contribute to progress also in CFLP.

Acknowledgements
We would like to thank Mario Rodríguez-Artalejo for his unvaluable help, sugestions and encouragement, without which this work surely would not exist. We also thank to the anonymous referee who detected and kindly indicated us some errors in a first version of Lemma 5.2.

7. References

[Apt90] K.R.Apt: Logic Programming, in J.van Leeuwen (ed.) Handbook of Theoretical Computer Science, Vol. B,pp. 495-574, Elsevier Science Pub. 1990.
[Bel&Lev86] M.Bellia, G.Levi: The Relation between Logic and Functional Languages, Journal of Logic Programming, Vol.3, 1986, 217-236.
[Dar&al91a] J.Darlington, Y.K.Guo, H. Pull: Introducing Constraint Functional Logic Programming, Tech. Rep., Imperial College, Feb. 1991.
[Dar&al91b] J.Darlington, Y.K.Guo, H. Pull: A New Perspective on the Integration of Functional and Logic Languages, Tech. Rep., Imperial College, Sep. 1991.
[Dar&Guo89] J.Darlington, Y.K.Guo: Constraint Functional Programming, Tech. Rep., Imperial College, November 1989.
[Dar&Guo90] J.Darlington, Y.K.Guo: Constraint Equational Deduction, Tech. Rep., Imperial College, March 1990.
[Der&Jou90] N.Dershowitz, J.P. Jouannaud: Rewrite Systems, in J.van Leeuwen (ed.) Handbook of Theoretical Computer Science, Vol. B,pp. 243-320, Elsevier Science Pub. 1990.

[Emd&Kow76] M.H. van Emden, R.A.Kowalski: The semantics of predicate logic as a programming language, Jour. ACM, Vol. 23 (4), 1976, pp. 733-742.
[Gab&Lev91] M. Gabrielli, G. Levi: Modelling Answer Constraint in Constraint Logic Programming, Tech. Rep. TR-4/91, Dip. Informatica, Univ. Pisa, March 1991.
[Gon&al90] J.C.González-Moreno, T.Hortalá-González, M.Rodríguez-Artalejo: A Functional Logic Language with Higher Order Logic Variables, Tech. Rep. DIA 90/6, October 1990.
[Gon&al91] J.C.González-Moreno, T.Hortalá-González, M.Rodríguez-Artalejo: Denotational versus Declarative Semantics for Functional Languages, Tech. Rep. DIA 91/3, September 1991.
[Gro&Lin86] D. de Groot, G.Lindstrom (eds): Logic Programming: Functions, Relations and Equations, Prentice Hall, 1986.
[Hoe&Smo88] M.Hoefeld, G.Smolka: Definite Relations over Constraint Languages, LILOG Report 53, IBM Deutchland, Germany, Oct. 1988.
[Kir&al90] C.Kirchner, H.Kirchner, M. Rusinowitch: Deduction with symbolic constraints, Revue Française d'Intelligence Artificielle, Vol. 4, N. 3, pp. 9-52.
[Jaf&Las86] J.Jaffar, J.L.Lassez: Constraint Logic Programming, Tech. Rep., Dep. of Computer Science, Monash Univ., June 1986.
[Jaf&Las87] J.Jaffar, J.L.Lassez: Constraint Logic Programming, Procs. 14th ACM Symp. on Princ. of Prog. Lang., 1987, pp. 114-119.
[Kuc&al90] H.Kuchen, R.Loogen, J.J.Moreno-Navarro, M.Rodríguez-Artalejo: Graph-based Implementation of a Functional Logic Language, Procs. ESOP'90, Springer LNCS 432, 1990, pp. 271-290.
[Kuc&al92] H.Kuchen, F.J.López-Fraguas, J.J.Moreno-Navarro, M.Rodríguez-Artalejo: Implementing Disequality in a Lazy Functional Logic Language, Tech. Rep. (in preparation).
[Kun87] K.Kunen: Negation in Logic Programming, Journal of Logic Programming, 4(3), pp. 289-308, 1987.
[Lev&al87] G. Levi et alter: A complete semantical characterization of K-LEAF, a logic language with partial functions, Procs. 4th. Symp. on Logic Progr., 1987, pp. 1-27.
[Lop&Rod91] F.J. López-Fraguas, M. Rodríguez-Artalejo: An Approach to Constraint Functional Logic Programming, Tech. Rep. DIA 91/4, October 1991.
[Lop92a] F.J. López-Fraguas: A General Scheme for Constraint Functional Logic Programming, Tech. Rep. (in preparation).
[Lop92b] F.J. López-Fraguas: Implementing Disequality in a Strict Functional Logic Language, Tech. Rep. (in preparation).
[Loo91] R. Loogen: From Reduction Machines to Narrowing Machines. In CCPSD, TAPSOFT 91, LNCS 494, 1991, pp. 438-454.
[Mor&al90] J.J.Moreno-Navarro, H.Kuchen, R.Loogen, M.Rodríguez-Artalejo: Lazy Narrowing in a Graph Machine, Procs. 2nd. Int. Conf. on Algebraic and Logic Programming, Springer LNCS 463, 1990, pp. 298-317.
[Mor&Rod88] J.J.Moreno-Navarro, M.Rodríguez-Artalejo: BABEL: A functional and logic language based and constructor discipline and narrowing. Procs. 1st. Int. Conf. on Algebraic and Logic Progr., Springer LNCS 343, 1989, pp. 223-232.
[Mor&Rod89] J.J.Moreno-Navarro, M.Rodríguez-Artalejo: Logic Programming with Functions and Predicates: the language BABEL, to appear in J. Logic Programming.
[Red85] U.S.Reddy: Narrowing as the Operational Semantics of Functional Languages. Procs. Int. Symp. on Logic Programming, IEEE Comp. Soc. Press 1985, pp. 138-151.
[Red87] U.S.Reddy: Functional Logic Languages, Part I, Procs. of a Workshop on Graph Reduction, Springer LNCS 279, 1987, pp. 401-425.
[Sco82] D.S. Scott: Domains for Denotational Semantics, Procs. ICALP'82, Springer LNCS 140, 1982, pp. 577-613.