The Semantics of Equational Logic Programming as an instance of CLP *

María Alpuente [‡] *Moreno Falaschi* [§] *Maurizio Gabbrielli* [§]
Giorgio Levi [§]

Abstract

We show how functional (i.e. equational) and logic programming can be integrated within the Constraint Logic Programming paradigm. The resulting language CLP($\mathcal{H}/\mathcal{E}$) is specialized in solving equations with respect to a Horn equational theory $\mathcal{E}$. CLP($\mathcal{H}/\mathcal{E}$) inherits all the semantic properties of the CLP scheme, including a new semantics which models answer constraints. The operational semantics of CLP($\mathcal{H}/\mathcal{E}$) is defined by a constraint solver based on conditional narrowing. Several strategies to obtain an incremental constraint solver are considered.

1. Introduction

In this paper we show how equational and logic programming can be integrated within the constraint logic programming paradigm thus inheriting all the CLP semantic properties. In the last few years several approaches to the integration of logic and equational programming have been developed [4, 9, 16, 22, 24, 29, 32, 7, 45, 46]. One relevant approach [29, 35] defines equational logic programs as logic programs augmented by Horn equational theories. Interpreted function symbols can appear as arguments of relations and existentially quantified variables can occur as arguments of functions. Function definition and evaluation are thus embedded in a logical framework. To properly cope with the equational theory, the conventional SLD-resolution mechanism based on a (syntactical) unification algorithm of logic programs has to be modified. The operational semantics of an equational logic language is based on

*This work was supported by ESPRIT Basic Research Action P3020 "Integration", by CICYT under grant TIC 91-0425, and by "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo of C.N.R." under grant n.9100880.PF69

[‡]Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n, Apdo. 22012, 46020 Valencia, Spain.

[§]Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy

some special form of equational resolution (such as SLDE resolution [5, 20, 24, 29]), where SLD-resolution is usually kept as the only inference rule but the syntactical unification algorithm is replaced by equational (semantic) unification ($\mathcal{E}$-*unification* [10, 51, 52]). $\mathcal{E}$-*unification* means the general unification w.r.t. an equational theory $\mathcal{E}$, which is usually described by a set of equations. The set of $\mathcal{E}$-*unifiers* of a pair of terms is only semidecidable, so that the process of semantic unification may run forever even if the equational theory is unconditional and canonical. Moreover, there is in general no single most general $\mathcal{E}$-unifier. Infinitely many incomparable mgu's over $\mathcal{E}$ of a pair of terms may exist. Three approaches are relevant to the problem of computing the set of $\mathcal{E}$-*unifiers* of two terms, namely flat SLD resolution [6, 7, 53, 54], complete sets of transformations [21, 30, 43] and paramodulation [17, 48] or some special form of it, such as superposition [16] or narrowing [6, 8, 23, 31, 32, 38, 47, 50].

The *Constraint Logic Programming* paradigm CLP($\mathcal{X}$) has been proposed by Jaffar and Lassez [34, 33] in order to integrate a generic computational mechanism based on constraints with the logic programming framework. The benefits of such an integration are several. From a pragmatic point of view, CLP($\mathcal{X}$) allows to use a specific constraints domain $\mathcal{X}$ and a related constraint solver within the declarative paradigm of logic programming. From the theoretical viewpoint, CLP($\mathcal{X}$) provides a unified view of several extensions to pure logic programming within a framework which preserves the unique semantic properties of logic programs, in particular the existence of equivalent operational, model theoretic and fixpoint semantics [34]. Moreover, since the computation is performed over the particular domain of computation $\mathcal{X}$, CLP($\mathcal{X}$) programs have an equivalent "algebraic" semantics [34] directly defined on the algebraic structure of $\mathcal{X}$.

In this paper we are concerned with an instance of the CLP scheme specialized in solving equations with respect to a Horn equational theory $\mathcal{E}$ [29, 35]. The intended structure is given by the finest partition induced by $\mathcal{E}$ on the Herbrand Universe $\mathcal{H}$ over a finite one sorted alphabet Σ. $=$ is the only predicate symbol for constraints (equational constraints) and is interpreted as semantic equality over the domain. In the following, we will refer to such a structure as $\mathcal{H}/\mathcal{E}$. The advantages of this approach are that, since the language is an instance of the scheme, all the above mentioned fundamental properties are automatically inherited by it. Besides, if an efficient algorithm to solve the constraints is developed it can easily be embedded into a general CLP interpreter and can cooperate with other constraint solvers.

Let us notice that for the language to be formally based on the semantics of the scheme, the structures must be *solution compact* as defined in [34, 33]. Informally, solution compactness means that every element of the domain can be described by a constraint and that the language of constraints must be precise enough to distinguish any object which does not satisfy a given constraint from those ones which do. Any structure which has no limit elements is trivially *solution compact*. This includes, in particular, the structure $\mathcal{H}/\mathcal{E}$.

A narrowing algorithm or some other $\mathcal{E}$-*unification* procedure can be considered the kernel of the constraint solver which semidecides the solvability of the constraints in the structure $\mathcal{H}/\mathcal{E}$. Solvability has to be tested but the equations do not need to be completely solved at every computation step. This paper first deals with an abstract description of an *incremental constraint solver* for equational logic programming which relies on a narrowing calculus. Our constraint solver not only checks the solvability but also simplifies the constraints. Then we describe a calculus for a narrowing procedure which is heuristically guided from the discarded substitutions, while looking for solutions to new constraints. Finally we present another calculus which builds incrementally the search tree of a constraint. For the incremental construction of this tree we follow an approach which is similar to [25] and our results rely on the innermost selection narrowing defined in [6]. The main issues are how to achieve incrementality in the computation process and how to use finitely failed derivations as a heuristic for optimizing the algorithms in order to achieve an intelligent narrowing.

We mentioned the fact that defining a new language as an instance of CLP($\mathcal{X}$) allows to inherit all the semantic properties of the CLP scheme. This is also the case for some new semantics which have recently be proposed with the aim of modeling observational equivalences [18]. These semantics extend the results in [13, 14] to the case of constraint logic programs. The construction is based on a new notion of interpretation (set of constrained atoms), on a natural extension of the standard notion of truth and on the definition of various immediate consequences operators, whose least fixpoints on the lattice of interpretations are models corresponding to various observable properties. Within such a framework, the equivalence between the operational and the declarative semantics can be fully achieved. For example, the least fixpoint of one immediate consequences operator fully characterizes the ability to compute answer constraints. This semantics will be considered in this paper. Its instantiation to the case of the structure $\mathcal{H}/\mathcal{E}$ is the semantics to be taken as the basis of a correct notion of program equivalence to be preserved by program transformation techniques, such as fold/unfold. Moreover, suitable abstract immediate consequence operators can be used for bottom-up abstract interpretation (i.e. fixpoint computation of the abstract model).

The paper is organized as follows. Section 2. considers the answer constraint semantics for CLP($\mathcal{X}$). In Section 3. we briefly recall some basic concepts on conditional term rewriting systems and universal unification. In Section 4. we define CLP($\mathcal{H}/\mathcal{E}$) logic programs and an incremental constraint solver as a kernel of an operational semantics for them. Section 5. is devoted to the heuristic narrowing calculus. Section 6. presents the calculus which incrementally builds the search tree of a constraint. We assume the reader to be familiar with logic programming [42], constraint logic programming [18, 34, 33], equations and conditional rewriting systems [37]. A preliminary short paper on CLP($\mathcal{H}/\mathcal{E}$) appeared in [1]. The proofs of the results mentioned in this paper can be found in [2] and [19].

2. An answer constraint semantics for CLP($\mathcal{X}$)

We will first recall the basic CLP concepts as defined in [34]. The CLP framework is defined using a many-sorted first order language, where $SORT = \bigcup SORT_i$ denotes a finite set of sorts. A *signature* of an n-ary function, predicate or variable symbol f is a sequence of respectively $n+1, n, 1$ elements of SORT. By the term *sort of* f we mean the last element in the signature of the function symbol f. By Σ , Π and V (possibly subscripted) we denote possibly denumerable collections of function symbols, predicate symbols and variables with their signatures and typical elements $f, g, \ldots, p, q \ldots$ and $X, Y, \ldots$ respectively. We assume there is a denumerable number of variable symbols for each sort and that each sort contains at least two symbols. $\tau(\Sigma \cup V)$ and $\tau(\Sigma)$ denote the set of terms and ground terms (i.e. terms without variables) built on Σ and V. $\tau(\Sigma)$ is usually called the Herbrand Universe ($\mathcal{H}_\Sigma$) over Σ and will be called $\mathcal{H}$ when the signature is clear from the context. A (Π, Σ)-atom is an element $p(t_1, \ldots, t_n)$ where $p \in \Pi$ is n-ary and $t_i \in \tau(\Sigma \cup V)$, $i = 1, \ldots, n$. A (Π, Σ)-constraint is a possibly empty or infinite conjunction of (Π, Σ)-atoms. The empty constraint will be denoted by *true*. The symbol $\tilde{}$ will denote a finite sequence of symbols. For a syntactic object O, by $var(O)$ we denote the set of variables occurring in O.

Definition 2.1 (CLP programs) *[34] Let* $\Pi = \Pi_C \cup \Pi_B$ *and* $\Pi_C \cap \Pi_B = \emptyset$. *A* (Π, Σ)*-program is a set of clauses of the form*

$$H : -c \Box. \qquad or \qquad H : -c \Box B_1, \ldots, B_n.$$

where c *is a finite* (Π_C, Σ)*-constraint,* H *(the head) and* $B_1, \ldots, B_n$ *(the body) are* (Π_B, Σ)*-atoms. A* goal *is a program clause with no head and with a non-empty body.*

Definition 2.2 *Let* $\Pi = \Pi_C \cup \Pi_B$ *and* $\Pi_C \cap \Pi_B = \emptyset$. *A* (Π_B, Σ)*-constrained atom is an object of the form* $c \Box p(\tilde{X})$*, where* c *is a* (Π_C, Σ)*-constraint and* $p(\tilde{X})$ *is a* (Π_B, Σ)*-atom.*

In the CLP($\mathcal{X}$) framework, $\mathcal{X}$ stands for a specific constraint domain on which the computation is performed. The notion of *structure* gives the semantic interpretation of such a domain and is the key element in the algebraic semantics.

Definition 2.3 *[34] Let* $\Pi = \Pi_C \cup \Pi_B$ *and* $\Pi_C \cap \Pi_B = \emptyset$. *A* structure $\Re(\Pi, \Sigma)$ *is defined over (sorted) alphabets* Π *and* Σ *of predicate and function symbols, where* Π *contains the equality symbol (which needs no signature).* $\Re(\Pi, \Sigma)$ *consists of*

1. *a collection* $D\Re$ *of non-empty sets* $D\Re_s$*, where* s *ranges over the sorts in SORT.*

2. *an assignment to each n-ary function symbol* $f \in \Sigma$ *of a function* $D\Re_{s_1} \times \ldots \times D\Re_{s_n} \rightarrow D\Re_s$ *where* $(s_1, s_2, \ldots, s_n, s)$ *is the signature of* f,

3. *an assignment to each n-ary predicate symbol* $p \in \Pi_C$, *apart from* $=$ *(interpreted as identity) of a function* $D\Re_{s_1} \times \ldots \times D\Re_{s_n} \rightarrow \{True, False\}$, *where* $(s_1, s_2, \ldots, s_n)$ *is the signature of* p.

A $\Re(\Pi, \Sigma)$*-valuation for a* (Π, Σ)*-expression is a mapping* $\theta : V \rightarrow D\Re$, *where* $V = \bigcup_{s \in SORT} V_s$ *is the set of all variables, and* $\theta(X_s) \in D\Re_s$, *where* s *is the sort of the variable* X_s.

(Π, Σ)-programs, (Π_C, Σ)-constraints and (Π_B, Σ)-constrained atoms will be called programs, constraints and constrained atoms. Moreover $\Re(\Pi, \Sigma)$ will be denoted by $\Re$. The notion of $\Re$-valuation is extended in the obvious way to terms and constraints. If C is a possibly infinite set of atomic constraints, we write $\Re \models C\theta$ iff $\forall c \in C\ \Re \models c\theta$ ($c\theta$ is $\Re$-equivalent to true) holds. If $A = p(t_1, \ldots, t_n)$ is a (Π_B, Σ)-atom and θ is a $\Re$-valuation on $t_1, t_2, \ldots, t_n$ then $A\theta$ denotes $p(t_1\theta, \ldots, t_n\theta)$.

Definition 2.4 (solvability)*[34] A constraint* c *is* $\Re$*-solvable iff there exists a* $\Re$*-valuation* θ *such that* $\Re \models c\theta$. θ *is called an* $\Re$*-solution of* c. *A constrained atom* $c \Box p(\tilde{X})$ *is* $\Re$*-solvable iff* c *is* $\Re$*-solvable.*

The structures considered in CLP are the "solution compact" ones as defined in [34, 33]. Since we are not concerned with negation, we will require the first condition only, namely

- $\forall$ sort s and $\forall d \in D\Re_s$, there exists a possibly infinite set C of constraints on the variable X such that for each $\Re$-solution θ of C, $X\theta = d$ holds [34].

2.1. n-interpretations and n-models

We introduce the new notion of n-model and show its relation with the standard notion of $\Re$-model as introduced in [34]. Since the structure $\Re$ is the intended domain of computation for CLP, interpretations for CLP programs can be based on $\Re$. All the following definitions are related to a given $\Re$ (and therefore to a given (Π, Σ)). Let us first introduce an operator on constrained atoms which returns the set of the "ground" instances.

Definition 2.5 *[34] The set of "domain instances"* $[c \Box p(\tilde{X})]$ *of a constrained atom* $c \Box p(\tilde{X})$ *is defined as*

$$[c \Box p(\tilde{X})] = \{p(\tilde{X})\theta \mid \theta \text{ is an } \Re\text{-solution of } c\}.$$

This definition can be extended to sets of constrained atoms. Let S be a set of constrained atoms. Then $[S] = \bigcup_{A \in S}[A]$

Let us now define a preorder on constrained atoms which represents the relation "being more constrained". Since we are not concerned with a specific domain of computation, we cannot give a syntactic characterization of the preorder. Hence we give the following definition, in terms of solutions of the constraints.

Definition 2.6 *Let $\mathcal{A}$ be the set of the constrained atoms for program P. We define the relation $\sqsubseteq$ on $\mathcal{A}$ as follows*

$$c_1 \Box p(\tilde{X}) \sqsubseteq c_2 \Box p(\tilde{X}) \text{ iff } [c_1 \Box p(\tilde{X})] \subseteq [c_2 \Box p(\tilde{X})].$$

We denote by $\equiv$ the equivalence induced by the preorder $\sqsubseteq$ on the set of constrained atoms, ($c_1 \equiv c_2$ iff $c_1 \sqsubseteq c_2$ and $c_2 \sqsubseteq c_1$).

Note that in case $var(c_1) \subseteq var(c_2) \subseteq \tilde{X}$, $c_1 \Box p(\tilde{X}) \sqsubseteq c_2 \Box p(\tilde{X})$ iff c_1 implies c_2. The following definitions introduce the notions of interpretations base, interpretations and models for CLP programs.

Definition 2.7 (n-$\Re$-base) *Let P be a (Π, Σ)-program and let $\mathcal{A}$ be the set of all the $\Re(\Pi, \Sigma)$-solvable constrained atoms for P. The n-$\Re$-base of interpretations $\mathcal{B}$ is the quotient set of $\mathcal{A}$ w.r.t. the equivalence relation $\equiv$. The ordering induced by $\sqsubseteq$ on $\mathcal{B}$ will still be denoted by $\sqsubseteq$. For the sake of simplicity A will represent the equivalence class of the constrained atom A.*

Definition 2.8 (n-interpretation) *An n-interpretation is any subset of $\mathcal{B}$. The set of all the n-interpretations is denoted by $\mathcal{I}$.*

Definition 2.9 (truth) *Let I be an interpretation. A constrained atom $c \Box p(\tilde{X})$ is true in I iff $[c \Box p(\tilde{X})] \subseteq [I]$. A clause $H \leftarrow c \Box B_1, \ldots, B_n$ is true in I iff for each $\Re$-valuation θ such that θ is a $\Re$-solution of c, $\{B_1\theta, \ldots, B_n\theta\} \subseteq [I]$ implies $H\theta \in [I]$.*

Definition 2.10 (n-models) *An n-model of a CLP program P is any n-interpretation in which all the clauses of P are true.*

Let us now show the relation between our concepts and the concepts of $\Re$-base, $\Re$-interpretations and $\Re$-models defined in [34]. The $\Re$-base $\mathcal{B}_\Re$ can be defined in terms of our base as $\mathcal{B}_\Re = [\mathcal{B}]$. $\Re$-interpretations of [34] are defined as usual as subsets of $\mathcal{B}_\Re$ and are a special case of our n-interpretations. Indeed, given the solution compactness

of the $\Re$ structure, every "domain instance" $p(d_1, \ldots, d_n)$ can be represented by the constrained atom $c\Box p(X_1, \ldots, X_n)$ where for each solution θ of the (possibly infinite) constraint c, $p(d_1, \ldots, d_n) = p(X_1, \ldots, X_n)\theta$. $\Re$-models are $\Re$-interpretations in which all the clauses of the program are true. It is worth noting that our notion of truth is consistent with the truth defined in [34] on $\Re$-interpretations, since it is defined on sets of "domain instances" of the form $p(\tilde{X})\theta$ where θ is an $\Re$-valuation. Hence also $\Re$-models are just a special case of n-models.

A parallel with the logic programming case can clarify the picture. Pure logic programming can be obtained from the general CLP framework by considering the structure $\mathcal{H}(\{=\} \cup \Pi_B, \Sigma)$ where $=$ is the only predicate symbol for constraints, the only $\mathcal{H}$-domain is the Herbrand universe over a finite one sorted alphabet Σ and $=$ is interpreted as syntactic equality over the domain. In the following we will refer to such a structure as $\mathcal{H}$. The logic programs standard semantics defines models as sets of ground atoms and this is exactly what is obtained by $\mathcal{H}$-models defined according to [34]. The new Herbrand base defined in [13] contains non-ground atoms in the models, where a non-ground atom stands for an implicit definition of the (possibly infinite) set of all its ground instances. This mechanism is generalized in the CLP case by allowing constrained atoms in the n-$\Re$-base, each constrained atom $c\Box p(\tilde{X})$ describing the set of elements $p(\tilde{X})\theta$ where θ is any $\Re$-solution for the constraint c. In order to abstract from the particular syntactic representation of atoms, in the logic programming case the new Herbrand base is defined modulo variance. Indeed, the $\mathcal{H}$ structure has the important property of the existence of a unique (up to variable renaming) canonical form of $(\{=\} \cup \Pi_B, \Sigma)$-constraints which allows to syntactically identify $\mathcal{H}$-equivalent constraints, namely the solved form of [39]. This property allows to compute on the Herbrand universe in a purely syntactic way using unification theory. In particular the unification algorithm which computes the idempotent mgu can be considered the constraint solver which decides the solvability of constraints and, if possible, returns the unique (up to variable renaming) canonical form.

When considering just the $\mathcal{H}$ structure, our definitions boil down to those of the s-semantics [13], since Theorem 2.1 is a straightforward consequence of the results in [39].

Theorem 2.1 *Let $A_1 = e_1\Box p(\tilde{X})$ and $A_2 = e_2\Box p(\tilde{Y})$ be two $\mathcal{H}$-constrained atoms. Then if the Herbrand universe is not trivial the following propositions hold*

1. $[A_1] = [A_2] = \emptyset$ *iff* e_i *is not unifiable,* $i = 1, 2$,

2. $[A_1] = [A_2] \neq \emptyset$ *iff* $p(\tilde{X})\vartheta_1 \equiv_t p(\tilde{X})\vartheta_2$ *where* ϑ_i *is an idempotent mgu of the set of equations* e_i *for* $i = 1, 2$, *and* $\equiv_t$ *is the variance equivalence.*

2.2. Operational semantics

We first show the CLP computational rule as defined in [34]. Then an operational semantics given in terms of transition systems is defined. The next step is the formalization of the success set $SS(P,\Re)$, which induces an equivalence $\approx$ on programs, which will be shown to characterize the operational behaviour of programs in terms of answer constraints.

Definition 2.11 (($P,\Re$)-derivation step)*[34] Let P be a program. A $(P,\Re)$-derivation step of a goal $:-(c\Box A_1,\ldots,A_n)$ results in a goal of the form $:-(\tilde{c}\Box\tilde{B}_1,\ldots,\tilde{B}_n)$, and is denoted by $(c\Box A_1,\ldots,A_n) \overset{(P,\Re)}{\rightarrow} (\tilde{c}\Box\tilde{B}_1,\ldots,\tilde{B}_n)$, if there exist n variants of clauses in P, $H_i :- c_i\Box\tilde{B}_i, i = 1,\ldots,n$, with no variables in common with G and with each other, such that $\tilde{c} = \{c, c_1, c_2, \ldots, c_n, A_1 = H_1, \ldots, A_n = H_n\}$ is $\Re$-solvable. A* $(P,\Re)$-derivation *of a goal G is a finite or infinite sequence of goals such that every goal, apart from G, is obtained from the previous one by means of a $(P,\Re)$-derivation step. A* successful *$(P,\Re)$-derivation of a goal G is a finite sequence whose last element is a goal of the form $(c\Box.)$ where c is the* answer constraint *of the derivation. All other finite $(P,\Re)$-derivation are* finitely failed. *The successful derivation of a goal G yielding the answer constraint c is denoted by $G \overset{(P,\Re)\star}{\longrightarrow} (c\Box.)$*

The above derivation step can be specified as a transition relation, following the operational approach to semantics based on transition systems [49]. A structural operational approach to the semantics of CLP is also taken in [26]. We need to specify the set Γ of configurations (states), the transition relation $\rightarrow_{CLP}$ and the set of terminal configurations.

Definition 2.12 (CLP($\mathcal{X}$) configurations) *Let P be a program and $G = \leftarrow c\Box A_1,\ldots,A_n.$ be a goal. We define a CLP($\mathcal{X}$) configuration $\mathcal{C}$ as a pair*

$$\mathcal{C} \;=\; \langle \leftarrow s[c] \diamond A_1,\ldots,A_n\rangle$$

where $s[c]$ denotes a constraint solver state, whose structure is left unspecified as it depends on the specific constraint solver, but which includes at least the constraint c. When the constraint c is clear from the context $s[c]$ will simply be denoted by s.

Definition 2.13 (CLP($\mathcal{X}$) transition relation $\rightarrow_{CLP}$) *The rule describing a $(P,\Re)$-computation step from a configuration $\langle \leftarrow s_i \diamond A_1,\ldots,A_n\rangle$ where s_i includes c_i is given by*

$$\frac{s_i \;\overset{\tilde{c}}{\hookrightarrow}\; s_{i+1}}{\langle \leftarrow s_i\diamond A_1,\ldots,A_n\rangle \;\rightarrow_{CLP}\; \langle \leftarrow s_{i+1}\diamond\tilde{B}_1,\ldots,\tilde{B}_n\rangle}$$

if there exist n variants of clauses in P, $H_j \leftarrow c'_j \Box \tilde{B_j}., j = 1, \ldots, n$, with no variables in common with $\leftarrow c_i \Box A_1, \ldots, A_n.$ and with each other and $\tilde{c} = \{c'_1, c'_2, \ldots, c'_n, A_1 = H_1, \ldots, A_n = H_n\}$. The condition $s_i \overset{\tilde{c}}{\hookrightarrow} s_{i+1}$ means that some constraint solver can make a move verifying the $\Re$-solvability of the constraint $c_i \cup \tilde{c}$ and returning the new constraint solver state s_{i+1} which includes (a possibly simplified version of) this constraint, c_{i+1}.

Definition 2.14 (CLP($\mathcal{X}$) initial configuration ($\mathcal{C}_0$)) *Let $G_0 = \leftarrow c_0 \Box \tilde{B}.$ be a goal and s_0 be the "empty" CLP constraint solver state. If c_0 is not an empty constraint and $s_0 \overset{c_0}{\hookrightarrow} s$ then $\mathcal{C}_0 = \langle \leftarrow s \diamond \tilde{B} \rangle$ is the CLP($\mathcal{X}$) initial configuration. If c_0 is empty then $\mathcal{C}_0 = \langle \leftarrow s_0 \diamond \tilde{B} \rangle$.*

Definition 2.15 (CLP($\mathcal{X}$) terminal configurations ($\mathcal{C}$)) *A terminal configuration $\mathcal{C}$ has the form $\mathcal{C} = \langle \leftarrow s[c] \diamond \quad \rangle$, where c represents the answer constraint.*

Let us notice that when the constraint solver is designed for simply testing the $\Re(\Pi, \Sigma)$-*Solvability* of the constraint $c_i \cup \tilde{c}$, the above defined transition relation becomes the standard $(P, \Re)$-derivation step as defined in [34]:

$$\frac{\Re(\Pi, \Sigma) \models (\tilde{\exists})(c_i \cup \tilde{c})}{\langle \leftarrow c_i \Box A_1, \ldots, A_n. \rangle \rightarrow_{CLP} \langle \leftarrow c_i \cup \tilde{c} \ \Box \tilde{B}_1, \ldots, \tilde{B}_n. \rangle}$$

In this case, the concepts of (P,$\Re$)-derivation and successful and finitely failed (P,$\Re$)-derivations can be defined in the usual way [34]. The constraint in the last goal of a successful derivation is the answer constraint of the derivation.

Definition 2.16 *Let P be a program. The success set $SS(P, \Re)$ is defined as follows*

$$SS(P, \Re) = \quad \{c \Box p(\tilde{X}) \quad \mid c \Box p(\tilde{X}) \in \mathcal{B}, \ (true \Box p(\tilde{X})) \overset{(P,\Re)}{\longrightarrow} (c \Box .) \quad \}.$$

Let us show an example of the semantics $SS(P, \mathcal{R})$ for a $CLP(\mathcal{R})$ program ($CLP(\mathcal{R})$ is defined on an arithmetical domain $\mathcal{R}$ of reals numbers [12]).

Example 2.1 *[12] The following program P computes the mortgage repayments. P is the principal, T is the mortgage lifetime in months, I is the annual percentage interest rate, R is the monthly repayment and B is the outstanding balance.*

$$\begin{aligned} P = \{ \quad & mortgage(P, T, I, R, B) :- \quad T = 1, B + R = P * (1 + I/1200). \\ & mortgage(P, T, I, R, B) :- \quad T > 1 \Box \\ & \qquad\qquad mortgage(P * (1 + I/1200) - R, T - 1, I, R, B). \quad \} \end{aligned}$$

Program P can be queried in several different way. For example the goal true$\Box$ mortgage(100000,360,12,R,0) *computes the answer constraint* $R = 1028.61$ *while the goal* true $\Box$ mortgage(P,120,12,R,B) *has answer constraint* $P = 0.303 * B + 69.7 * R$.

The $SS(P, \mathcal{R})$ semantics of program P in the example 2.1 is the following set, where m stands for $mortgage(P, T, I, R, B)$

$$\begin{array}{ll} \{ & T = 1, B + R = P * (1 + I/1200)\Box m, \\ & T = 2, B + R = P * (1 + I/1200)2 - R * (1 + I/1200)\Box m, \\ & T = 3, B + R = P * (1 + I/1200)3 - R * (1 + I/1200)2 + \\ & \qquad\qquad -R * (1 + I/1200)\Box m, \\ & \vdots \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \}. \end{array}$$

Let us now show the soundness and completeness results for $SS(P, \Re)$. As resulting from the following Theorem 2.2 $SS(P, \Re)$ can fully characterize the operational behaviour of programs. The proof of the theorem is based on the observation that if a goal $G = c_0 \Box p_1(\tilde{t}_1), \ldots, p_n(\tilde{t}_n)$ has a successful derivation with answer constraint c_{ans}, then defining $\overline{c} = c_0 \cup (\bigcup_{i=1,\ldots,n} \tilde{X}_i = \tilde{t}_i, c_i)$ where the c_i are the answer constraints of the goals $c_0 \Box p_i(\tilde{X}_i)$, $\overline{c}$ is equivalent (up to variables in G) to c_{ans} (and analogously for the converse).

Theorem 2.2 *Let P be a CLP program and $G = (c_0 \Box A_1, \ldots, A_n)$ any goal. Then $G \stackrel{(P,\Re)}{\rightarrow} (c_{ans} \Box .)$ iff there exists n constrained atoms $(c_i \Box B_i) \in SS(P, \Re)$, $i = 1, \ldots n$, which share no variables with G and with each other, such that $c_0 \cup c_1 \cup \ldots \cup c_n \cup A_1 = B_1 \cup \ldots \cup A_n = B_n$ and c_{ans} have the same solutions for the variables in G.*

2.3. Fixpoint semantics

Let us now introduce an immediate consequence operator, which allows to characterize the $SS(P, \Re)$ operational semantics.

Definition 2.17 *Let P be a CLP program and let $J \subseteq \mathcal{B}$.*

$$\begin{array}{lll} T_{(P,\Re)}(J) = & \{ & c \Box p(\tilde{X}) \in \mathcal{B} \mid \\ & & \exists \text{ a renamed clause } p(\tilde{t}) : -c_0 \Box p_1(\tilde{t_1}), \ldots, p_n(\tilde{t_n}) \text{ in } P \\ & & \exists \ c_i \Box p_i(\tilde{X_i}) \in J, 1 \leq i \leq n \text{ which share no variables} \\ & & c' = c_0 \cup \{\tilde{X_i} = \tilde{t_1}, \ldots, \tilde{X_n} = \tilde{t_n}\} \cup \{c_1, \ldots, c_n\} \\ & & c' \text{ is } \Re\text{-solvable, finite and } c = c' \cup \{\tilde{X} = \tilde{t}\} \qquad \}. \end{array}$$

Note that previous operator is defined in terms of the CLP derivation rule. Indeed $T_{(P,\Re)}(J)$ can be seen as the result of the unfolding of program P wrt the set of unit clauses extracted from J (see later). This is the key property which allows to model answer constraints by means of a bottom-up fixpoint semantics. Moreover, note that the standard immediate consequence operator defined in [34] can be obtained by considering the "domain instances" $[T_{(P,\Re)}(J)]$ of the defined operator.

Theorem 2.3 *The mapping $T_{(P,\Re)}$ is continuous on the cpo $(\mathcal{I}_P, \subseteq)$. There exists a least fixpoint $lfp(T_{(P,\Re)})$ of $T_{(P,\Re)}$ which is $T_{(P,\Re)} \uparrow \omega$.*

Definition 2.18 *Let P be a program. Then the fixpoint semantics $Fix(P,\Re)$ of P is defined as $Fix(P,\Re) = T_{(P,\Re)} \uparrow \omega$.*

2.4. Unfolding semantics and equivalence results

The equivalence between the operational and the fixpoint semantics can be proved in a concise and elegant way by introducing the intermediate notion of *unfolding semantics* $Unf(P,\Re)$ [40, 41]. $Unf(P,\Re)$ is obtained as the limit of the unfolding process. The proof of the equivalence between $Unf(P,\Re)$ and $SS(P,\Re)$ is straightforward since these semantics are based on the same inference rule (applied to clauses and goals respectively). On the other side, the equivalence between $Unf(P,\Re)$ and $Fix(P,\Re)$ can be based on the strict relation existing between $T_{(P,\Re)}(I)$ and the unfolding rule (see below).

Definition 2.19 *Let P and Q be CLP programs. Then the unfolding of P w.r.t. Q is defined as*

$$\begin{aligned} unf_P(Q) = \ \{(A :- c_0, c_1, \ldots, c_n \Box \tilde{L}_1, \ldots, \tilde{L}_n) \mid \\ \exists\ A :- c_0 \Box p_1(\tilde{t}_1), \ldots, p_n(\tilde{t}_n) \in P \\ \exists\ p_i(\tilde{m}_i) :- c'_i \Box \tilde{L}_i \in Q \text{ for } i = 1, \ldots, n \\ c_0, c_1, \ldots, c_n \text{ is } \Re\text{-solvable where } c_i = c'_i \cup \tilde{t}_i = \tilde{m}_i \ \ \} \end{aligned}$$

The following definition allows to extract from a CLP program P an n-interpretation and vice versa.

Definition 2.20 *Let P be an CLP program and I be an n-interpretation. Then we define*

$$\alpha(P) = \{c \cup \tilde{X} = \tilde{t} \Box p(\tilde{X}) \in \mathcal{B} \mid p(\tilde{t}) :- c \Box. \in P\}$$

and

$$\gamma(I) = \{p(\tilde{t}) :- c \Box. \mid c \cup \tilde{X} = \tilde{t} \Box p(\tilde{X}) = \mu(c_a) \text{ and } c_a \in I\}.$$

where $\mu(c_a)$ returns one element of the the equivalence class c_a.

Definition 2.21 *Let P be an CLP program. Then we define the collection of programs*

$$P_1 = P$$
$$P_i = unf_{P_{i-1}}(P)$$

The unfolding semantics $Unf(P, \Re)$ of the program P is defined as

$$Unf(P, \Re) = \bigcup_{i=1,2,\ldots} \alpha(P_i).$$

The following theorem states the equality of the unfolding and the operational semantics. Its proof is easy since both the semantics are top-down.

Theorem 2.4 *Let P be an Ω-open program. Then $Unf(P, \Re) = SS(P, \Re)$.*

Note that $T_{(P,\Re)}(I) = \alpha(unf_P(\gamma(I)))$ holds. This relation allows to prove the following theorem.

Theorem 2.5 *Let P be a program. Then $Fix(P, \Re) = Unf(P, \Re)$.*

Corollary 2.1 (equivalence of the operational and the fixpoint semantics) *Let P be a program. Then $SS(P, \Re) = Fix(P, \Re)$.*

2.5. The model-theoretic semantics

In this section we discuss the models introduced by the fixpoint and the operational semantics. Since we are concerned with the algebraic semantics, i.e. the semantics on the particular structure $\Re$ considered as domain of computation, we will consider the notion $\models_\Re$ of logical consequence in $\Re$-models as opposed to the standard notion of logical consequence $\models$.

Definition 2.22 *Let P be a program and G be a constrained atom. $P \models_\Re G$ iff G is true in every $\Re$-model of P, i.e. according to definition 2.9, if $\forall M$ $\Re$-model of P, $[G] \subseteq M$.*

An $\Re$-interpretation I, can be regarded as a particular n-interpretation I_n. Indeed, since the structure $\Re$ is solution compact, we can represent each "domain instance" $p(\tilde{d})$ by a constrained atom $c \Box p(\tilde{X})$ such that the solutions of the (possibly infinite) constraint c define the elements $\tilde{d}$. Formally we can give the following definition.

Definition 2.23 *Let I be an $\Re$-interpretation. Then we define the corresponding n-interpretation $\eta(I)$ as follows*

$$\eta(I) = \{c\Box p(\tilde{X}) \in \mathcal{B} \mid \exists\ p(\tilde{d}) \in I\ \wedge\ \forall\ \theta\ \textit{solution of}\ c,\ p(\tilde{X})\theta = p(\tilde{d})\}.$$

The following theorem shows that $\eta(I)$ and I are equivalent from the model theoretic point of view.

Theorem 2.6 *Let I be an $\Re$-interpretation and c a clause. Then c is true in I iff c is true in $\eta(I)$.*

In general, since we have one concept of truth only (definition 2.9, being true in every $\Re$-model is equivalent to being true in every n-model, i.e. from the model theoretic point of view, n-models and $\Re$-models are equivalent. Note that the intersection of a set of n-models is not always an n-model. For example, consider the program $P = \{p(X) : -X = a\Box\}$ on the Herbrand domain. Clearly both $M_1 = \{X = a\Box p(X)\}$ and $M_2 = \{X = Y\Box p(X)\}$ are n-model of P, but $M \cap M_2 = \emptyset$ is not an n-model of P. Therefore in general there exists no minimal n-model with respect to set inclusion. We will then introduce an ordering $\lhd$ and a lattice structure on n-interpretations which allow to restore the model intersection property. The model theoretic semantics of a program P is then defined as the greatest lower bound of the set of all n-models of P. Theorem 2.9 shows that this model theoretic semantics definition is the same of the usual one given in [34] in terms of $\Re$-models.

Definition 2.24 (ordering on n-interpretations) *We define a relation $\lhd$ on $\mathcal{I}$ as follows. Let $I_1, I_2 \in \mathcal{I}$ Then*
$I_1 \leq I_2$ iff $\forall A_1 \in I_1 \exists A_2 \in I_2$ such that $A_1 \sqsubseteq A_2$ and
$I_1 \lhd I_2$ iff $(I_1 \leq I_2)$ and $(I_2 \leq I_1$ implies $I_1 \subseteq I_2)$.

The intuitive meaning of the above defined relations is the following. $I_1 \leq I_2$ means that every contrained atom verified by I_1 is also verified by I_2 (I_2 contains more positive information). $I_1 \lhd I_2$ means that either I_2 strictly contains more positive information than I_1 or (if the amount of positive information is the same), that I_1 expresses it by less elements than I_2 (I_2 is more redundant).

Definition 2.25 *Let P be a program. Its* model theoretic semantics *$\mathcal{M}_P$ is defined as $\mathcal{M}_P = glb\{N \mid N$ is a n-model of $P\}$.*

The previous definitions extend to the CLP case results given in [14] for logic programs. In the particular case of $\Re$-interpretations the $\lhd$ ordering is set-inclusion, and the following theorem shows that there exists the minimal R-model w.r.t. set inclusion.

Theorem 2.7 *[34] Let P be a CLP and let M be the set of all the $\Re$-models of P. Then $\mathcal{M}_\Re = \bigcap_{m \in M} m$ is the least $\Re$-model of P w.r.t. set inclusion.*

$\mathcal{M}_\Re$ can be considered as the CLP counterpart of the standard least Herbrand model for logic programs. The following theorems show that indeed the $\mathcal{M}_\Re$ minimal model is the "same" (up to the η transformation) of the least n-model defined according to the $\lhd$ ordering on n-interpretations. The proofs are based on the observation that $[M] = [\eta([M])]$, on the monotonicity of η and on the definition of the ordering $\lhd$.

Theorem 2.8 *Let M be an n-model of a program P. Then $\eta([M])$ is an R-model of P and $\eta([M]) \lhd M$.*

Theorem 2.9 *Given a program P let $\mathcal{M}_\Re$ be its least $\Re$-model. Then $\eta(\mathcal{M}_\Re) = \mathcal{M}_P$.*

Analogously to the logic programming case, the least model characterizes the set of the "ground" $\Re$-logical consequences of the program and is equal to the "ground" fixpoint and operational semantics defined in [33]. The answer constraint semantics is also an n-model, as shown by the following theorem. Note that $SS(P, \Re)$ is not a minimal model.

Lemma 2.2 *Let P be a program. Then $SS(P, \Re)$ is an n-model of P and if $c \Box p(\tilde{X}) \in SS(P, \Re)$ then $c \Box p(\tilde{X}) \equiv c' \Box p(\tilde{X})$ where c' is finite.*

Finally let us mention that, as shown in [34], it is possible to have a complete correspondence between the algebraic and logical semantics, by considering a theory $\Im$ which corresponds to $\Re$, in the sense that
(i) $\Re \models \Im$, i.e. $\Re$ models $\Im$
(ii) $\Re \models \exists C$ implies $\Im \models \exists C$ for all constraints C.
Therefore considering the algebraic semantics only is not a restriction. Indeed our approach can be adapted to the logical semantics as well, following the previously mentioned correspondence.

3. Basic definitions on equations and term rewriting systems

Let us briefly recall some basic notions and results about equations, conditional rewrite systems and universal unification. Full definitions can be found in [37, 51].

Definition 3.1 *A Σ-equation $s{=}t$ is a pair of terms s and t of the same sort ($s, t \in \tau(\Sigma \cup V)$). A Horn equational Σ-theory $\mathcal{E}$ consists of a finite set of equational Horn*

clauses of the form $l = r \leftarrow e_1, e_2, ..., e_n, \quad n \geq 0$, *where the* Σ*-equation* $l = r$ *in the head is implicitely oriented from left to right and the literals* e_i *in the body are ordinary non-oriented* Σ*-equations.* Σ*-equations and* Σ*-theories will often be called equations and theories, respectively.*

A Horn equational theory $\mathcal{E}$ can be viewed as a term rewriting system $\mathcal{R}$ where the rules are the heads and the conditions are the respective bodies. If all clauses in $\mathcal{E}$ have an empty body then $\mathcal{E}$ and $\mathcal{R}$ are said to be unconditional, otherwise they are said to be conditional. An equational theory $\mathcal{E}$ has no extra-variables if every variable occurring in the body or in the right-hand side of the head of a clause in $\mathcal{E}$ also occurs in the left-hand side of the head. The equational theory $\mathcal{E}$ is said to be canonical if the binary one-step rewriting relation $\rightarrow_{\mathcal{R}}$ defined by $\mathcal{R}$ is noetherian and confluent. For syntactical characterizations of confluent conditional theories refer to [45]. Each Horn equational theory $\mathcal{E}$ generates a smallest congruence relation $=_{\mathcal{E}}$ called $\mathcal{E}$*-equality* on the set of terms $\tau(\Sigma \cup V)$. We will denote by $\mathcal{H}/\mathcal{E}$ the finest partition $\tau(\Sigma)/ =_{\mathcal{E}}$ induced by $\mathcal{E}$ over the set of ground terms $\tau(\Sigma)$.

Definition 3.2 *Given two terms* s *and* t*, we say that they are* $\mathcal{E}$*-unifiable (or* $\mathcal{E}$*-equal) iff there exists a substitution* σ *such that* $s\sigma$ *and* $t\sigma$ *are in the congruence* $=_{\mathcal{E}}$*, i.e. such that* $s\sigma =_{\mathcal{E}} t\sigma$ *or, equivalently*[1]*,* $\mathcal{E} \models s\sigma = t\sigma$*. The substitution* σ *is called an* $\mathcal{E}$*-unifier of* s *and* t*.*

Roughly speaking S is a complete set of unifiers for e if any other unifier can be obtained by instance from some element of S (see [51] for a formal definition). Complete sets of minimal $\mathcal{E}$*-unifiers* do not always exists [51]. $\mathcal{E}$*-unification* can be viewed as the process of solving an equation within the theory $\mathcal{E}$. Since $\mathcal{E}$*-unification* is only semidecidable, an $\mathcal{E}$*-unification* algorithm can be viewed as a semidecision procedure for testing the solvability of equational constraints over the quotient $\mathcal{H}/\mathcal{E}$. Each instance of an $\mathcal{E}$*-unifier* represents a solution over this structure (an $\mathcal{H}/\mathcal{E}$-solution).

A $\mathcal{E}$*-unification* procedure is complete if it generates a complete set of $\mathcal{E}$*-unifiers* for all input equations. A number of $\mathcal{E}$*-unification* procedures have been developed in order to deal with conditional equational theories [9, 16, 22, 30, 32, 37]. For instance, conditional narrowing has been shown to be complete for conditional theories satisfying different restrictions [9, 29, 37].

Definition 3.3 *Let* $\mathcal{E}$ *be a Horn equational theory. A function symbol* $f \in \Sigma$ *is called irreducible iff there is no clause* $(l = r \leftarrow e_1, e_2, ..., e_n) \in \mathcal{E}$ *such that* f *occurs as the outermost function symbol in* l *or* l *is a variable; otherwise it is a defined function symbol.*

[1] We assume that interpretations obey the axioms of equality, i.e. the reflexivity, symmetry, transitivity and the substitutivity for functions and predicates. Hence, satisfiability and logical consequence are defined w.r.t. these axioms.

The use of irreducible function symbols is justified from the several optimization techniques defined in [32]. This affects the size of the search tree of the $\mathcal{E}$-unifiers for a set of equations. In the best case an infinite search tree can be reduced to a finite one. In theories where the above distinction is made, the signature Σ is partitioned as $\Sigma = C \uplus F$, where C is the set of irreducible function symbols and F is the set of definite function ones. The members of C are also called *constructors*. Terms are viewed as labelled trees in the usual way [31]. We will use, among others, the following standard notions: Occurrences are represented by sequences, possibly empty, of naturals. The empty sequence is represented by λ. t/u is the subterm at the occurrence u of t. $t[u \leftarrow r]$ is the term t with the subterm at the occurrence u replaced with r.

4. The language CLP($\mathcal{H}/\mathcal{E}$) and its Operational Semantics

4.1. CLP($\mathcal{H}/\mathcal{E}$) logic programs

In this section we define the language CLP($\mathcal{H}/\mathcal{E}$), which is an instance of CLP that caters for solving constraints in the structure $\mathcal{H}/\mathcal{E}$, defined as the finest partition induced by a canonical Horn equational theory without extra variables $\mathcal{E}$ on the Herbrand Universe $\mathcal{H}$ over a finite one sorted alphabet Σ. $=$ is the only predicate symbol for constraints and is interpreted as semantic equality over the domain. This structure is *solution compact* since has no limit element [33]. It is known [35] that the equational theory $\mathcal{E}$ and the structure $\mathcal{H}/\mathcal{E}$ correspond in the sense defined in [33], i.e. $\mathcal{H}/\mathcal{E} \models \mathcal{E}$, i.e. $\mathcal{H}/\mathcal{E}$ models $\mathcal{E}$ and $\mathcal{H}/\mathcal{E} \models \tilde{\exists} c$ implies $\mathcal{E} \models \tilde{\exists} c$ for all constraint c. Thus, a constraint c is $\mathcal{E}$-unifiable iff it is $\mathcal{H}/\mathcal{E}$-solvable. We assume that (Π_C, Σ)-constraints are sets of Σ-equations to be solved in $\mathcal{E}$ by a suitable conditional narrowing algorithm which will be described in the following. The equational representation of a substitution $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ is the set of equations, $\widehat{\theta} = \{x_1 = t_1, \ldots, x_n = t_n\}$.

Definition 4.1 CLP($\mathcal{H}/\mathcal{E}$) programs
Let $\Pi_C = \{=\}$, $\Pi = \Pi_C \cup \Pi_B$ *and* $\Pi_C \cap \Pi_B = \emptyset$. *We define a CLP($\mathcal{H}/\mathcal{E}$)* (Π, Σ)*-program as a* (Π, Σ)*-program augmented by a canonical Horn equational* Σ*-theory without extra variables* $\mathcal{E}$.

Example 4.1 *What follows is a CLP($\mathcal{H}/\mathcal{E}$) program* $P \cup \mathcal{E}$:

Number of nodes of a binary tree

$$
\begin{array}{lll}
P = & \{ & \textit{(p1)}\ nn(\Lambda, 0). \\
 & & \textit{(p2)}\ nn(t(L_t, E, R_t), N) \leftarrow N = s(0) + L_n + R_n \,\square\, nn(L_t, L_n), nn(R_t, R_n).\} \\
\mathcal{E} = & \{ & \textit{(e1)}\ 0 + y = y. \\
 & & \textit{(e2)}\ s(x) + y = s(x + y).\ \}
\end{array}
$$

CLP($\mathcal{H}/\mathcal{E}$) programs enjoy the following property [35]: there is a canonical domain of computation, namely $\mathcal{H}/\mathcal{E}$, over which a canonical class of interpretations can be built. This is analogous to the canonicity of Herbrand interpretations for definite clause programs. For reasoning about negation, the theory $\mathcal{E}$ should be extended to be satisfaction complete, i.e. $\Im \models \neg c$ whenever not $\Im \models \tilde{\exists} c$. Roughly speaking, satisfaction completeness means that every constraint is either provably satisfiable or provably unsatisfiable [34]. Unfortunately this is not possible. Hence the soundness and completeness results for finite failure and negation as failure are not inherited.

4.2. CLP($\mathcal{H}/\mathcal{E}$) Incremental Constraint Solver

The standard way to integrate narrowing and resolution [5, 24] is using narrowing to generate the solutions which are then tested by the logic program. This method does not fit in the CLP scheme and requires some complex interactions between the backtrackings of the narrowing and the resolution algorithms. Instead, narrowing can be used as a procedure to test the consistency of the new constraint, looking for a solution, since the constraint is proved consistent once a solution has been found. Thus we consider how to incrementally reuse the work done in the previous step to keep smaller the cost of solving the new set. An approach that does not try to exploit the information which can be gathered from the previous steps to guide the search would be impractical [25].

In the CLP($\mathcal{H}/\mathcal{E}$) framework it seems very natural to reuse the solution θ to the constraint c found in the previous step to check the solvability of the new constraint $c \cup \check{c}$. In [25] the matter of incremental constraint satisfaction is also addressed. [25] shows that an approach based on backtracking is inadequate since it can only use passively the new constraints to test if they are satisfied by the newly generated tentative solutions. In the context of CLP it seems to be more appropriate using actively the new constraints to guide the search towards a solution. Accordingly, [25] presents a scheme based on reexecution and pruning of the search tree of resolution. We have a similar problem, yet related to the search tree of the constraint solving algorithm. In this section we are concerned with an incremental algorithm for testing the solvability and an appropriate simplified representation for constraints. In the following we describe a constraint solver for CLP($\mathcal{H}/\mathcal{E}$) programs by means of a transition system labeled on (Π_C, Σ)-constraints and specially tailored to be incremental.

Definition 4.2 *ICS*-representation of a solvable constraint
Let c be a solvable constraint. Let us suppose that c can be written in the form $c_1 \cup c_2$ (either of which can be empty) where c_1 has a unique mgu θ over $\mathcal{E}$. Then $r_c = (\widehat{\theta}, c_2)$ is the ICS-representation of the simplified constraint $\widehat{\theta} \cup c_2$. The ICS-representation of an empty constraint is $(\emptyset, \emptyset)$, which will be abbreviated as $()$.

Let us notice that if $r_c = (\widehat{\theta}\ , \emptyset)$ then the constraint c has a unique mgu θ over $\mathcal{E}$.

Definition 4.3 *ICS*-states
We define an Incremental Constraint Solver (ICS) state s as a triple $s = \langle\ \theta, f,\ r_c\ \rangle$, where r_c is the ICS-representation of a (Π_C, Σ)-constraint c, θ represents a $\mathcal{E}$-unifier of c and f is a set of (Π, Σ)-substitutions.

Roughly speaking, for the Incremental Constraint Solver which will be described next, r_c represents the accumulated simplified constraint and f represents the set of substitutions which have already been unsuccessfully tried by the constraint solver and that are useful for a heuristic search of other solutions.

Definition 4.4 empty *ICS*-state, terminal *ICS*-states
$\langle\ \epsilon, \emptyset,\ ()\ \rangle$ *is the empty ICS-state. Any ICS-state can be a terminal ICS-state.*

Definition 4.5 *ICS*-labels
An ICS label $\tilde{c}$ is a (Π_C, Σ)-constraint. It represents the new constraint $\tilde{c}$ to be added to the accumulated simplified constraint c in order to test the solvability of the new set $c \cup \tilde{c}$.

In the following we describe the ICS transition system. Roughly speaking, let $s = \langle\ \theta, f,\ r_c\ \rangle$ be an ICS-state. In order to achieve incrementality, the costs of testing the solvability of the new set of constraints $c \cup \tilde{c}$ should be as close as possible to the cost of solving the new constraint $\tilde{c}$ plus the cost of combining the new solution with the previous one. With this aim, since the substitution θ represents an $\mathcal{E}$-unifier of the accumulated solvable constraint c, we search for an $\mathcal{E}$-unifier to the new set $c \cup \tilde{c}$ looking for the solutions to $\tilde{c}\theta$. If $\tilde{c}\theta$ has no solution, we must start from scratch, looking for the solutions to the whole constraint $c \cup \tilde{c}$ and add θ to the set of unsuccessfully tried substitutions. We use a $\mathcal{H}$euristic $\mathcal{C}$onditional $\mathcal{N}$arrowing procedure $\mathcal{N}$ which will be described later for the search of a solution θ' of this constraint and combine adequately θ and θ' to obtain a new accumulated solution and a new accumulated simplified constraint.

Definition 4.6 *ICS* transition relation[2] $\xrightarrow{\tilde{c}}_{ICS}$

Single Solution Rules

$$(1)\quad \frac{\mathcal{N}(\tilde{c}\theta, \emptyset)\ \textit{has a unique solution}\ \theta'}{\langle\ \theta, f,\ (\widehat{\theta}, \emptyset)\ \rangle \xrightarrow{\tilde{c}}_{ICS} \langle\ \theta\theta', f,\ (\widehat{\theta\theta'}, \emptyset)\ \rangle}$$

[2]For the sake of simplicity, in this set of rules c_2 denotes a non empty constraint

$$(2)\quad \frac{\mathcal{N}(\tilde{c}\theta, \emptyset) \textit{ has a unique solution } \theta' \wedge\ \mathcal{N}((c_2 \cup \tilde{c})\theta_1, \emptyset) \textit{ has a unique sol } \theta''}{\langle\ \theta, f,\ (\widehat{\theta}_1, c_2)\ \rangle \xrightarrow{\tilde{c}}_{ICS} \langle\ \theta_1\theta'', f,\ (\widehat{\theta_1\theta''}, \emptyset)\ \rangle}$$

Multiple Solution Rules

$$(3)\quad \frac{\mathcal{N}(\tilde{c}\theta, \emptyset) \textit{ has a unique solution } \theta' \wedge\ \mathcal{N}((c_2 \cup \tilde{c})\theta_1, \emptyset) \textit{ has a first sol } \theta''}{\langle\ \theta, f,\ (\widehat{\theta}_1, c_2)\ \rangle \xrightarrow{\tilde{c}}_{ICS} \langle\theta\theta', f,\ (\widehat{\theta}_1, c_2 \cup \tilde{c})\ \rangle}$$

$$(4)\quad \frac{\mathcal{N}(\tilde{c}\theta, \emptyset) \textit{ has a first solution } \theta'}{\langle\ \theta, f,\ (\widehat{\theta}_1, c')\ \rangle \xrightarrow{\tilde{c}}_{ICS} \langle\ \theta\theta', f,\ (\widehat{\theta}_1, c' \cup \tilde{c})\ \rangle}$$

Start from Scratch Rules

$$(5)\quad \frac{\mathcal{N}(\tilde{c}\theta, \emptyset) \textit{ has no solution } \wedge\ \mathcal{N}(\widehat{\theta_1 \cup c_2 \cup \tilde{c}}, f \cup \theta) \textit{ has a unique solution } \theta'}{\langle\ \theta, f,\ (\widehat{\theta}_1, c_2)\ \rangle \xrightarrow{\tilde{c}}_{ICS} \langle\ \theta', f \cup \theta\ ,\ (\widehat{\theta}', \emptyset)\ \rangle}$$

$$(6)\quad \frac{\mathcal{N}(\tilde{c}\theta, \emptyset) \textit{ has no solution } \wedge\ \mathcal{N}(\widehat{\theta_1 \cup c_2 \cup \tilde{c}}, f \cup \theta) \textit{ has a first solution } \theta'}{\langle\ \theta, f,\ (\widehat{\theta}_1, c_2)\ \rangle \xrightarrow{\tilde{c}}_{ICS} \langle\ \theta', f \cup \theta,\ (\widehat{\theta}_1, c_2 \cup \tilde{c})\ \rangle}$$

Let us notice that each transition of this system depends on the termination of the $\mathcal{N}$arrowing procedure $\mathcal{N}$. We assume that if $\mathcal{N}$arrowing procedure $\mathcal{N}(c, f)$ terminates returning the output *no solution* then the constraint c has no $\mathcal{E}$-unifier, if the output *unique solution* θ then θ is the unique mgu over $\mathcal{E}$ of c and if the output is *first solution* θ then θ is an $\mathcal{E}$-unifier of c but, possibly, it is not the unique mgu over $\mathcal{E}$ of c. In the following we will define a transition system for $CLP(\mathcal{H}/\mathcal{E})$ which depends on the ICS solver. The transition relation for $CLP(\mathcal{H}/\mathcal{E})$ will depend on one single ICS transition step. Therefore, we do not rely on a termination proof for the ICS system.

The following theorem establishes the correctness of the Incremental Constraint Solver.

Theorem 4.1 (Correctness of ICS)
If an ICS transition $\langle\ \theta, f,\ r_c\ \rangle \xrightarrow{\tilde{c}}_{ICS} \langle\theta', f',\ r'_c\ \rangle$ can be proved then the constraint $c \cup \tilde{c}$ is solvable in $\mathcal{H}/\mathcal{E}$. r'_c represents the simplified form of this constraint and θ' represents an $\mathcal{E}$-unifier to it. f' represents the set of substitutions unsuccessfully tried.

Obviously the incremental constraint solver as defined above is, in general, not complete because narrowing may loop forever in the evaluation of the condition of a rule

not only when the constraint is not $\mathcal{E}$-*unifiable* but also when it has a unique mgu over $\mathcal{E}$ that has already been found and it tries to obtain another solution. However, for some classes of problems, an incomplete but efficient constraint solver might turn out to be valuable from a programming standpoint, as pointed out in [26].

4.3. CLP($\mathcal{H}/\mathcal{E}$) Operational Model

The following definitions instantiate definitions 2.13, 2.14 and 2.15 to the case of CLP($\mathcal{H}/\mathcal{E}$) programs.

Definition 4.7 CLP($\mathcal{H}/\mathcal{E}$) transition relation $\rightarrow_{CLP(\mathcal{H}/\mathcal{E})}$
Let $P \cup \mathcal{E}$ be a CLP($\mathcal{H}/\mathcal{E}$) (Π, Σ)-program. The CLP($\mathcal{H}/\mathcal{E}$) Transition Relation is given by the following rule describing a computation step from a configuration $\langle \leftarrow s_i \diamond A_1, \ldots, A_n \rangle$ where $s_i = \langle\ \theta_i\ ,\ f_i\ ,\ r_{c_i}\ \rangle$:

$$\frac{s_i \xrightarrow{\tilde{c}}_{ICS} s_{i+1}}{\langle \leftarrow s_i \diamond A_1, \ldots, A_n \rangle \rightarrow_{CLP(\mathcal{H}/\mathcal{E})} \langle \leftarrow s_{i+1} \diamond \tilde{B}_1, \ldots, \tilde{B}_n \rangle}$$

if there exist n variants of clauses in P, $H_j \leftarrow c'_j \Box \tilde{B}_j, j = 1, \ldots, n$, with no variables in common with $\leftarrow c_i \Box A_1, \ldots, A_n$. and with each other and $\tilde{c} = \{c'_1, c'_2, \ldots, c'_n, A_1 = H_1, \ldots, A_n = H_n\}$.

Definition 4.8 CLP($\mathcal{H}/\mathcal{E}$) initial configuration, CLP($\mathcal{H}/\mathcal{E}$) terminal configurations
Let $G_0 = \leftarrow c_0 \Box \tilde{B}$. be a goal. If c_0 is not an empty constraint and the empty ICS-state $s_0 \xrightarrow{c_0}_{ICS} s$ then $\mathcal{C}_0 = \langle \leftarrow s \diamond \tilde{B} \rangle$ is the CLP($\mathcal{H}/\mathcal{E}$) initial configuration. If c_0 is empty then $\mathcal{C}_0 = \langle \leftarrow s_0 \diamond \tilde{B} \rangle$. A terminal configuration has the form $\langle \leftarrow s \diamond \rangle$, where $s = \langle\ \theta, f,\ (\widehat{\theta_1}, c_2)\ \rangle$ is an ICS-state and $c = (\widehat{\theta_1} \cup c_2)_{\restriction var(G_0)}$[3] is the answer constraint.

As in the generic case, when the constraint solver is designed for simply testing the solvability of constraints, the above defined transition relation $\rightarrow_{CLP(\mathcal{H}/\mathcal{E})}$ becomes the standard definition of derivation of goals. We give an example that illustrates the definitions above.

Example 4.2 *Let us consider the following CLP($\mathcal{H}/\mathcal{E}$) program*[4] *$P \cup \mathcal{E}$ [28]:*

0/1 Knapsack problem

[3] By $(\widehat{\theta_1} \cup c_2) \restriction var(G_0)$ we denote the least subset $E' \subseteq E = \widehat{\theta_1 \restriction var(G_0)} \cup c_2\theta_1$ containing all the equations in E in which a variable from the initial goal G_0 occurs and such that no variable occurring in an equation in E' occurs in any of the equations in $E \setminus E'$. Trivial equations of the form $t = t$ are discarded.

[4] For convenience we often abbreviate $s^n(0)$ to n.

$P = \{$ *(p1)* $knapsack(M,L,W) \leftarrow addweight(M) = W \ \square \ sublist(M,L).$
(p2) $sublist([\,],Z).$
(p3) $sublist([X \mid Y],[X \mid Z]) \leftarrow \ \square \ sublist(Y,Z).$
(p4) $sublist(Y,[X \mid Z]) \leftarrow \ \square \ sublist(Y,Z). \ \}$

$\mathcal{E} = \{$ *(e1)* $0 + Y = Y.$
(e2) $s(X) + Y = s(X + Y).$
(e3) $weight(a) = 1.$
(e4) $weight(b) = 1.$
(e5) $weight(c) = 2.$
(e6) $addweight([\,]) = 0.$
(e7) $addweight([X \mid Y]) = weight(X) + addweight(Y). \ \}$

% knapsack(M,L,W) states that items in the sublist M of a given list L can be packed into a knapsack such that the knapsack weights exactly W %

Let us consider the initial goal $G_0 = \leftarrow \ \square \ knapsack(K,[a,b],2)$. *Since the constraint in* G_0 *is empty, the initial CLP($\mathcal{H}/\mathcal{E}$) configuration is*
$\mathcal{C}_0 = \langle \ \leftarrow \langle \epsilon, \emptyset, () \rangle \ \diamond \ knapsack(K,[a,b],2) \ \rangle$
considering $(p1)$, *the new constraint* $\tilde{c}$ *is*
$\tilde{c} = \{addweight(M) = W, K = M, L = [a,b], W = 2\}$
since $(\tilde{c}\epsilon)$ *has a first solution*[5] $\theta = \{M/[a,a], K/[a,a], L/[a,b], W/2\}$, *applying* (4),
$\mathcal{C}_0 \rightarrow_{CLP(\mathcal{H}/\mathcal{E})} \mathcal{C}_1 = \langle \ \leftarrow \langle \theta, \emptyset, (\emptyset, \tilde{c}) \rangle \ \diamond \ sublist(M,L) \ \rangle$
considering $(p3)$, *the new constraint* $\tilde{c}'$ *is*
$\tilde{c}' = \{M = [X \mid Y], L = [X \mid Z]\}$
since $(\tilde{c}'\theta)$ *has a unique solution* $\theta' = \{X/a, Y/[a], Z/[b]\}$ *and* $\tilde{c} \cup \tilde{c}'$ *has a first solution* $\theta\theta' = \{M/[a,a], K/[a,a], L/[a,b], W/2, X/a, Y/[a], Z/[b]\}$, *applying* (3),
$\mathcal{C}_1 \rightarrow_{CLP(\mathcal{H}/\mathcal{E})} \mathcal{C}_2 = \langle \ \leftarrow \langle \theta\theta', \emptyset, (\emptyset, \tilde{c} \cup \tilde{c}') \rangle \ \diamond \ sublist(Y,Z) \ \rangle$
considering again $(p3)$, *the new constraint* $\tilde{c}''$ *is*
$\tilde{c}'' = \{Y = [X' \mid Y'], Z = [X' \mid Z']\}$
since $(\tilde{c}''\theta\theta')$ *has no solution and* $\tilde{c} \cup \tilde{c}' \cup \tilde{c}''$ *has a unique solution*[6]
$\theta'' = \{M/[a,b], K/[a,b], L/[a,b], W/2, X/a, Y/[b], Z/[b], X'/b, Y'/[\,], Z'/[\,]\}$, *applying* (5),
$\mathcal{C}_2 \rightarrow_{CLP(\mathcal{H}/\mathcal{E})} \mathcal{C}_3 = \langle \ \leftarrow \langle \theta'', \theta\theta', (\widehat{\theta''}, \emptyset) \rangle \ \diamond \ sublist(Y',Z') \ \rangle$
considering $(p2)$, *the new constraint* $\tilde{c}'''$ *is*
$\tilde{c}''' = \{Y' = [\,], Z' = Z''\}$
since $(\tilde{c}'''\theta'')$ *has a unique solution* $\theta''' = \{Z''/[\,]\}$, *applying* (1),
$\mathcal{C}_3 \rightarrow_{CLP(\mathcal{H}/\mathcal{E})} \mathcal{C}_4 = \langle \ \leftarrow \langle \theta''\theta''', \theta\theta', (\widehat{\theta''\theta'''}, \emptyset) \rangle \ \diamond \ \rangle$

[5] Note that other solutions (e.g. $\{M/[c], K/[c], L/[a,b], W/2\}$) can be found.

[6] The narrowing calculus we define in Section 5. is able to find this solution and prove its uniqueness.

finally, since a final ICS-configuration has been reached, the answer constraint associated to this transition sequence is $(\widehat{\theta''\theta'''} \cup \mathcal{O})_{\restriction var(G_0)} = (K = [a, b])$

As shown in the example, the accumulated solvable constraint is simplified when rules (1), (2) and (5) are applied and the solution found in the previous step is always reused to try to prove that the new accumulated constraint is solvable. Thus the test of the solvability results much less expensive than a solvability test for the whole (non simplified) constraint.

5. Heuristic Conditional Narrowing

In this section, we present a $\mathcal{H}$euristic $\mathcal{C}$onditional $\mathcal{N}$arrowing $\mathcal{C}$alculus ($\mathcal{HCNC}$), an adaptation of the Conditional Narrowing Algorithm CNA presented in [32] which was shown complete for noetherian and confluent rewrite systems without extra variables. We extend it by actively using heuristic information to quickly converge towards a solution.

Narrowing was originally introduced in the field of automated theorem proving as a restricted version of the paramodulation rule [29]. A narrowing procedure for first order unification in an equational theory defined by a canonical and unconditional term rewriting system was given by Fay [15] and later improved by Hullot in [31]. Fay worked with normalizing narrowing and Hullot introduced the notion of basic narrowing. Nutt, Rety and Smolka [47] studied basic narrowing and its optimizations combined with normalizing narrowing. Bosco et al. [5] introduced narrowing selection. Lazy narrowing stems from Reddy [50]. Padawitz [48] presented refinements of narrowing and proved the correctness of the optimizations. Dershowitz [9], Hussmann [32] and Kaplan [38] investigated narrowing for conditional term rewriting systems. Hussmann extended Fay/Hullot work to theories defined by a confluent (possibly nonterminating) system of conditional rewrite rules and a completeness result w.r.t. normalized substitutions was given which holds if the system is non-trivial and the usual no extra-variables condition in the right-hand sides of the heads and in the bodies of the equational clauses is imposed [29]. Giovannetti and Moiso [23] explored the consequences of allowing the presence of extra-variables and gave a sufficient condition for completeness. Hölldobler [29] generalized the results obtained by Hullot and Hussmann. Fribourg [16] employed an innermost case of strategy-controlled narrowing. Middeldorp [44] carefully analyzes the completeness results for conditional basic narrowing. Echahed [11] and Padawitz [48] studied criteria to ensure completeness of narrowing selection strategies.

Let us briefly recall the Conditional Narrowing Algorithm CNA presented in [32]. An expression of the form $c \; with \; \tau$ where c is a set of equations and τ is a substitution is called a (sub-)goal. The substitution τ is assumed to be restricted to $var(c_0)$, i.e. the variables of the constraint in the initial state, and the variables in the equational

clauses are standardized apart, i.e. the set of variables in c and in the clauses are disjoint. To solve c_0, the algorithm starts with the subgoal c_0 *with* ϵ and tries to derive subgoals (recording applied substitutions in the with-parts) until a terminal goal of the form $\emptyset$ *with* θ is reached. Each substitution θ in a terminal goal is an $\mathcal{E}$-*unifier* of c_0. By abuse of notation, it will often be called solution. Let $O'(c)$ be the set of nonvariable occurrences of a constraint c. We define
$narred(c, u, k, \sigma) \Leftrightarrow (\ u \in O'(c) \ \wedge \ (l_k = r_k \leftarrow \tilde{e}_k) \in \mathcal{E} \ \wedge \ \sigma = mgu(c/u, l_k)\ )$

The calculus CNA is defined by the following two rules:

CNA unification rule:

$$\frac{c \text{ syntactically unifies with } mgu\ \sigma}{c\ with\ \tau \quad \rightarrow_{CNA} \quad \emptyset\ with\ \tau\sigma}$$

CNA narrowing rule:

$$\frac{narred(c, u, k, \sigma) \text{ holds}}{c\ with\ \tau \quad \rightarrow_{CNA} \quad (\tilde{e}_k, c[u \leftarrow r_k])\sigma\ with\ \tau\sigma}$$

This calculus defines an algorithm, since all the CNA-derivations (sequences of states) from a given subgoal can be easily enumerated. These derivations can be represented by a (possibly infinite) finitely branching tree. The nodes of this tree have to be visited following a complete search strategy (e.g. breadth first) while looking for unifiable nodes.

In the following, we present the $\mathcal{H}$euristic $\mathcal{C}$onditional $\mathcal{N}$arrowing $\mathcal{C}$alculus ($\mathcal{HCNC}$).

Definition 5.1 $\mathcal{N}$-state
An $\mathcal{N}$-state is a triple $\langle n, \theta, L\rangle$, where n is a positive integer $\in \{0, 1, 2\}$, θ is a (Π, Σ)-substitution and L is a (possibly empty) list $[g_i]_{i=1}^{n}$ of subgoals $g_i = c_i$ with τ_i, where c_i is a set of (Π_c, Σ)-constraints (a conjunction of equations) and τ_i is a (Π, Σ)-substitution. List constructors are denoted by [ ] and $\bullet$. By abuse of notation, we assume $\bullet$ to be homomorphically extended to concatenation of lists.

Roughly speaking, the first component of an $\mathcal{N}$-state represents the current number of solutions. The second component represents the solution to be returned and the third component represents the list of subgoals yet to be narrowed. The nodes of the narrowing search tree are stored in the list. This list is treated as a queue to emulate a breadth first search strategy of the search tree.

To solve the conjunction of equations c_0 using the heuristic information f, the algorithm starts with the initial $\mathcal{N}$-state $\mathcal{N}_0 = \langle 0, \epsilon, [\ c_0\ with\ \epsilon\ ]\rangle$ and tries to derive subgoals (recording applied substitutions in the with-parts) until a terminal $\mathcal{N}$-state

$\langle n, \theta, L\rangle \not\rightarrow_{\mathcal{HCNC}}$ is reached. Terminal $\mathcal{N}-$states are characterized by $L = [\,]$. The set f of substitutions unsuccessfully tried by the constraint solver are used for a heuristic search of the solutions. If ρ is a substitution which belongs to f, then, by the compactness theorem it can not solve the accumulated constraint c because there is a subset of c which can not be solved. Moreover, no substitution ρ' such that, restricted to the variables of c_0, is an instance of ρ can solve c_0 either.

Given a set of variables W we say that two substitutions θ, σ are equal on W, denoted by $\sigma = \theta[W]$, iff $x\sigma = x\theta \;\; \forall x \in W$. Let us define the relation $\preceq$ over substitutions: a substitution θ is more general than σ on W, denoted by $\theta \preceq \sigma[W]$, iff $\sigma = \theta\lambda[W]$ for some substitution λ. In the following we assume $W = var(c_0)$, where c_0 is the constraint in the initial state, and thus it will be omitted.

Let $\langle n, \theta, L\rangle$ be a $\mathcal{N}$-state and $c_i \;\; with \;\; \tau_i$ be a subgoal in L. Let $S = \{\langle u, k, \sigma\rangle \mid narred(c_i, u, k, \sigma)\}$. If $\langle u', k', \sigma'\rangle \;\; \in S$ and $(\exists \rho \in f)$ such that $\rho \preceq \tau_i\sigma'$ then the corresponding subtree of the search tree which has as root the subgoal $(\tilde{e}_{k'}, c_i[u' \leftarrow r_{k'}])\sigma' \;\; with \;\; \tau_i\sigma'$ does not contain any solution. Since it is useless to explore this subtree, we will 'prune the search tree' by removing the representation of its root $(\langle u', k', \sigma'\rangle)$ from S.

The set of substitutions f can be thought of as a characterization of non unifiability. The prune based on f involves an improvement comparable to the profit drawn from a perfect intelligent backtracking. Intelligent backtracking [55] analyzes unification failure to avoid thrashing which arises when backtracking would follow paths which predictably contain no solutions. We obtain this improvement simply by comparing the previously failed substitutions with the current one by means of the relationship $\preceq$. The cost of keeping the set f is linear with the length of the $CLP(\mathcal{H}/\mathcal{E})$ derivation.

Definition 5.2 $\mathcal{H}$euristic $\mathcal{C}$onditional $\mathcal{N}$arrowing $\mathcal{C}$alculus ($\mathcal{HCNC}$)

The $\mathcal{HCNC}$ calculus is defined by means of a stratified transition system[7]*:*

Heuristic Branching Rules

$$(1) \quad \frac{\{\langle u, k, \sigma\rangle \mid \;\; narred(c, u, k, \sigma) \;\; \} = \emptyset}{c \;\; with \;\; \tau \rightarrow_{Branch} [\,]}$$

$$(2) \quad \frac{s_to_l(\{\langle u, k, \sigma\rangle \mid narred(c, u, k, \sigma) \wedge \{\rho \in f \mid \rho \preceq \tau\sigma\} = \emptyset\}) = [\langle u_i, k_i, \sigma_i\rangle]_{i=1}^{n}}{c \;\; with \;\; \tau \rightarrow_{Branch} [\;\; (\tilde{e}_{k_i}, c[u_i \leftarrow r_{k_i}])\sigma_i \;\; with \;\; \tau\sigma_i \;\;]_{i=1}^{n}}$$

where the choice function $s_to_l(S)$ returns a list composed by the elements of the set S.

Success Rules

[7] The states of the transition system for Branch are subgoals or list of subgoals

$$(3)\quad \frac{c \ with \ \tau \rightarrow_{\mathcal{B}ranch} L' \ \wedge \ c \text{ synt. unifies with } mgu \ \sigma}{\langle 0, \epsilon, \ c \ with \ \tau \bullet L\rangle \rightarrow_{\mathcal{HCNC}} \langle 1, \tau\sigma, \ L \bullet L'\rangle}$$

$$(4)\quad \frac{c \ with \ \tau \rightarrow_{\mathcal{B}ranch} L' \ \wedge \ c \text{ synt. unifies with } mgu \ \sigma \ \wedge \ (\tau\sigma) \not\preceq \theta \ \wedge \ \theta \not\preceq (\tau\sigma)}{\langle 1, \theta, \ c \ with \ \tau \bullet L\rangle \rightarrow_{\mathcal{HCNC}} \langle 2, \theta, \ [\,]\rangle}$$

Narrowing Rules

$$(5)\quad \frac{c \ with \ \tau \rightarrow_{\mathcal{B}ranch} L' \ \wedge \quad c \text{ syntactically unifies with } mgu \ \sigma \ \wedge \ \theta \preceq (\tau\sigma)}{\langle 1, \theta, \ c \ with \ \tau \bullet L\rangle \rightarrow_{\mathcal{HCNC}} \langle 1, \theta, \ L \bullet L'\rangle}$$

$$(6)\quad \frac{c \ with \ \tau \rightarrow_{\mathcal{B}ranch} L' \ \wedge \ c \text{ syntactically unifies with } mgu \ \sigma \ \wedge \ (\tau\sigma) \preceq \theta}{\langle 1, \theta, \ c \ with \ \tau \bullet L\rangle \rightarrow_{\mathcal{HCNC}} \langle 1, \tau\sigma, \ L \bullet L'\rangle}$$

$$(7)\quad \frac{c \ with \ \tau \rightarrow_{\mathcal{B}ranch} L' \ \wedge \quad mgu(c) = \emptyset \ \wedge \quad n \in \{0, 1\}}{\langle n, \theta, \ c \ with \ \tau \bullet L\rangle \rightarrow_{\mathcal{HCNC}} \langle n, \theta, \ L \bullet L'\rangle}$$

Roughly speaking, the Narrowing and Success rules basically consider two cases: the constraint c has an *mgu* σ or not. In the latter case we just proceed. In the first case we compare $\tau\sigma$ (where τ is the substitution accumulated by narrowing so far) with the first solution θ. In case they are incomparable we stop as a second solution for the initial constraint has been found. In case $\tau\sigma$ is more general we replace the first solution by it (we have found a more general *first* solution). Finally, we just proceed in case the solution θ is more general than $\tau\sigma$. Let us now illustrate Definition 5.2 by the following:

Example 5.1 *Let us consider the constraint* $c = \{U + s(Z) = s(0)\}$ *within the equational theory* $\mathcal{E}$ *in Example 4.2. Let* $f = \{\{U/s(X)\}\}$. *The sequence* $\langle 0, \epsilon, [U + s(Z) = s(0) \ \ with \ \epsilon]\rangle \xrightarrow{(7)}_{\mathcal{HCNC}} \langle 0, \epsilon, [s(Z) = s(0) \ \ with \ \ \{U/0\}]\rangle \xrightarrow{(3)}_{\mathcal{HCNC}} \langle 1, \{U/0, Z/0\}, [\,]\rangle$ *shows that* $\mathcal{HCNC}$ *is able to find the (unique) solution* $\{U/0, Z/0\}$ *for* c. *Notice that* $U + s(Z) = s(0) \ \ with \ \epsilon \rightarrow_{\mathcal{B}ranch} [s(Z) = s(0) \ \ with \ \ \{U/0\}]$ *as* $U + s(Z)$ *unifies with the heads of the clauses* $(e1)$ *and* $(e2)$, *but for the mgu* $\rho' = \{U/s(X'), Y'/s(Z)\}$ *with the head of* $(e2)$ *there exists* $\rho = \{U/s(X)\} \in f$ *such that* $\rho \preceq \rho'$.

Definition 5.3 Behaviour of the $\mathcal{HCNC}$ calculus
Let $\mathcal{N}_0 = \langle 0, \epsilon, [\ c_0 \ with \ \epsilon \ ]\rangle$. *We define:*

$$\mathcal{N}(c_0, f) = \begin{cases} no \ solution & if \ \mathcal{N}_0 \rightarrow^*_{\mathcal{HCNC}} \langle 0, \epsilon, [\,]\rangle \\ unique \ solution \ \theta & if \ \mathcal{N}_0 \rightarrow^*_{\mathcal{HCNC}} \langle 1, \theta, [\,]\rangle \\ first \ solution \ \theta & if \ \mathcal{N}_0 \rightarrow^*_{\mathcal{HCNC}} \langle 2, \theta, [\,]\rangle \end{cases}$$

Let us notice that if $f = \emptyset$ then the above algorithm explores (the prefix of) the search tree of CNA which would have been explored following a breadth first search strategy until the conditions characterizing a terminal $\mathcal{N}$-state would have been reached. If $\mathcal{HCNC}$ does not terminate or terminates with $L = [\,]$ and $n \in \{0, 1\}$ then $\mathcal{N}(c, f)$ is equivalent to CNA as it explores exactly the same search tree.

When $f \neq \emptyset$ the heuristic pruning based on f only removes subtrees which do not contain any solution so that it is guaranteed that no solution is lost.

The following theorem establishes the correctness of the $\mathcal{H}$euristic $\mathcal{C}$onditional $\mathcal{N}$arrowing $\mathcal{N}(c, f)$.

Theorem 5.1 (Correctness of the $\mathcal{H}$euristic $\mathcal{C}$onditional $\mathcal{N}$arrowing)
Let $\mathcal{E}$ be a canonical conditional Horn equational theory without extra variables. Let c be a constraint. Let f be a set of substitutions such that no instance of any of them can solve c. If $\mathcal{N}(c, f)$ returns the output no solution *then the constraint c is not solvable. If the output is* unique solution *θ then θ is the unique mgu over $\mathcal{E}$ of c. If the output is* first solution *θ then θ is a solution over $\mathcal{E}$ to c.*

Let us notice that when $\mathcal{N}(c, f)$ returns the output *first solution* θ, θ might be the unique mgu over $\mathcal{E}$ of c. In fact, the solutions are checked not to be syntactic instances of one another while only two solutions which are not $\mathcal{E}$-instances of one another should be distinguished. This means that the incremental constraint solver may simplify the constraint less than it would be possible. The Heuristic Conditional Narrowing as defined above is a procedure to test the solvability of constraints over the structure $\mathcal{H}/\mathcal{E}$. If the constraint is solvable then the procedure finds a solution (an $\mathcal{E}$-$unifier$) to it and looks for another incomparable $\mathcal{E}$-$unifier$. If the constraint is not $\mathcal{E}$-$unifiable$ or if there is not a second solution, the procedure may run forever.

Even in the case of canonical theories, the use of narrowing as a semantic unification procedure presents several drawbacks since it may not terminate when there are no (more) solutions and the generation of an independent set of $\mathcal{E}$-$unifiers$ is not guaranteed. An infinite set of solutions can be derived even if the equation admits a finite complete minimal one [31]. This problem was also inherited by the various techniques developed for equational resolution, as SLDE-resolution, where a single resolution step may loop forever without producing any answer. In [31], a sufficient condition for termination is given for canonical non-conditional theories.

To overcome these difficulties, several proposals have been made. In [29] a lazy resolution rule is introduced. With this rule, the problem of the unification of a set of equations within the theory $\mathcal{E}$ must not be solved before the rule can be applied but the equations are added as a constraint to the derived goal clause without testing its solvability. For instance, the constraint solving could be delayed until all other goals are solved and eventually the accumulated constraint can be solved by a complete

$\mathcal{E}$-*unification* algorithm. This is similar to the postponement of the satisfiability test of non linear constraints in CLP($\Re$) [36]. [3] presents a static analysis which allows to approximate the problem of detection of unsatisfiability of a set of equations. [3] also shows how to use actively this analysis to prune the search tree of an incremental equational constraint solver.

5.1. Optimizations of $\mathcal{HCNC}$

The high degree of non-determinism inherent to narrowing often causes the generation of many redundant derivations. It is clear that the above procedure should be optimized in order to be suitably implemented and embedded into CLP($\mathcal{H}/\mathcal{E}$). In this section we recall two useful optimizations (defined in [32, 16, 47]) of $\mathcal{HCNC}$ that significantly reduce the size of the search tree.

1. This optimization is based on the concept of irreducible function symbol to detect the unsolvability of an equation :

   If a subgoal in a $\mathcal{N}$-state occurring in $\mathcal{HCNC}$ contains an equation $f(t_1, t_2, \ldots, t_n) = g(t'_1, t'_2, \ldots, t'_n)$ and f, g are irreducible, then:

   - if $f \neq g$ then the subgoal is not solvable and can be removed.
   - if $f = g$ then the equation can be replaced by the set of equations $t_1 = t'_1,\ t_2 = t'_2,\ \ldots,\ t_n = t'_n$.

   This optimization can reduce an infinite search tree to a finite one [32].

2. If a subgoal in a $\mathcal{N}$-state occurring in $\mathcal{HCNC}$ contains an equation $x = t$ or $t = x$ where $x \in V, x \notin var(t)$ and all function symbols in t are irreducible, then the equation can be dropped. The substitution $\{x/t\}$ has to be applied to the whole subgoal.

For other optimizations, we refer to [32, 48]. Several strategies to reduce the number of subterms that have to be narrowed can be considered [6, 23, 31, 47]. The following section proposes an optimization strategy based on the results about selection narrowing presented in [6, 23].

6. CLP($\mathcal{H}/\mathcal{E}$) Optimized Incremental Constraint Solver

In this section we define another incremental constraint solver iCS for CLP($\mathcal{H}/\mathcal{E}$) which allows to drastically cut the search space under certain circumstances. iCS reuses the work already done when a new constraint is incrementally added. The

kernel of iCS is an innermost conditional narrowing calculus $i\mathcal{CNC}$ which explores (a prefix of) a search tree for the constraint $c \cup \tilde{c}$ which is built by extending the innermost selection narrowing strategy defined in [6]. This tree is incrementally built as long as new constraints are added.

The innermost conditional narrowing calculus $i\mathcal{CNC}$ defines a strategy which looks for a single $\mathcal{E}$-unifier θ of the input constraint c and returns as output the list L of subgoals yet to be narrowed.

We assume in the following a function *select-don't-care*(Occ) which nondeterministically selects one innermost occurrence (i.e. an occurrence which is not the prefix of any other occurrence [6, 16]) in the set Occ. $\sharp c$ stands for the cardinality of the set c.

Definition 6.1 *Innermost Conditional Narrowing Calculus ($i\mathcal{CNC}$) An $i\mathcal{CNC}$-state is a pair $\langle n, L\rangle$, where $n \in \{0,1\}$ and L is a (possibly empty) list $[g_i]_{i=1}^n$ of subgoals $g_i = c_i$ with (τ_i, Occ_i), where c_i is a set of (Π_c, Σ)-constraints (a conjunction of equations), τ_i is a (Π, Σ)- substitution and Occ_i is the set of narrowable occurrences of c_i.*

The $i\mathcal{CNC}$ transition relation is the following[8]*:*

Don't-care Branching Rule

$$(1)\quad \frac{\begin{array}{l} u_0 = \textit{select-don't- care}(Occ) \wedge \\ \textit{s_to_l}(\{\langle u_0, k, \sigma\rangle \mid \ \textit{narred}(c, u_0, k, \sigma)\}) \ = [\langle u_0, k_i, \sigma_i\rangle]_{i=1}^n \end{array}}{\begin{array}{l} c \ \textit{with} \ (\tau, Occ) \ \rightarrow_{DcB} \\ c \ \textit{with} \ (\tau, (Occ \setminus \{u_0\})) \bullet [\ (c[u_0 \leftarrow r_{k_i}], \tilde{e}_{k_i})\sigma_i \ \textit{with} \\ \ (\tau\sigma_i, ((Occ \setminus \{u_0\}) \cup \{u_0.v \mid \ v \in O'(r_{k_i})\} \cup O(c, \tilde{e}_{k_i}))) \ ]_{i=1}^n \end{array}}$$

where $O(c, \tilde{c}) = \{(\sharp c + j).k \mid \ j.k \in O'(\tilde{c})\}$.

Unification Rule

$$(2)\quad \frac{c \text{ syntactically unifies}}{\langle 0, \ c \ \textit{with} \ (\tau, \emptyset) \bullet L\rangle \rightarrow_{i\mathcal{CNC}} \langle 1, \ c \ \textit{with} \ (\tau, \emptyset) \bullet L\rangle}$$

Narrowing Rules

$$(3)\quad \frac{c \ \textit{with} \ (\tau, Occ) \rightarrow_{DcB} L'}{\langle 0, \ c \ \textit{with} \ (\tau, Occ) \bullet L\rangle \rightarrow_{i\mathcal{CNC}} \langle 0, \ L \bullet L'\rangle}$$

[8] We assume for simplicity that in the following rules Occ is a non empty set.

$$(4) \quad \frac{mgu(c) = \emptyset}{\langle 0,\ c\ with\ (\tau, \emptyset) \bullet L\rangle \rightarrow_{i\mathcal{CNC}} \langle 0,\ L\rangle}$$

Roughly speaking, in Definition 6.1 the effect of the function *select-don't-care*(Occ) is to limit the search at each application of rule (1) to one subterm only from all the possible ones. This strategy can be further optimized by the following test. Remove from the list in the second argument of the state any *failed subgoal*, i.e. any subgoal $c\,with\,(\tau, Occ)$ such that one equation $e \in c$ is not unifiable and there is no occurrence in Occ for e. A node of a search tree is called *failed* if it contains a failed subgoal.

Definition 6.2 Behaviour of the $i\mathcal{CNC}$ calculus
Let us define the function:

$i\mathcal{N}(L) = \ L' \ \ if \ \ \langle 0, L\rangle \rightarrow^*_{i\mathcal{CNC}} \langle 1, L'\rangle$

We introduce an innermost search tree as a tree whose nodes are labeled by subgoals, the root is labeled by the constraint to be solved and at each node the innermost narrowable occurrence (nondeterministically chosen) is expanded by the innermost selection narrowing algorithm in [6] extended to conditional theories (innermost conditional narrowing for short).

Definition 6.3 *An innermost search tree for a constraint c to be solved is a tree whose nodes are labeled by subgoals $c_i\ with\ (\tau_i, Occ_i)$, where the subgoal in the root is $c\ with\ (\epsilon, O'(c))$.*

Let $g = c\ with\ (\tau, Occ)$ be a node where $Occ \neq \emptyset$. Let $u_0 =$ select-don't-care(Occ) *and assume that $s_to_l(\{\langle u_0, k, \sigma\rangle \mid \ narred(c, u_0, k, \sigma)\}) = [\langle u_0, k_i, \sigma_i\rangle]_{i=1}^m$. Then g has a child $c\ with\ (\tau, Occ\backslash\{u_0\})$ and m children $(c[u_0 \leftarrow r_{k_i}], \tilde{e}_{k_i})\sigma_i\ with$ $(\tau\sigma_i, ((Occ \setminus \{u_0\}) \cup \{u_0.v \mid \ v \in O'(r_{k_i})\} \cup O(c, \tilde{e}_{k_i})))$, $i = 1, \ldots, m$.*

Definition 6.4 *Let us consider an innermost search tree t for the constraint c. The list L_c of the non-failed leaves of t is called the representation of t.*

Notice that in Definition 6.4 L_c represents an execution state of the innermost conditional narrowing algorithm, i.e. the set of derived subgoals which are yet to be narrowed.

Definition 6.5 *An iCS-state is a pair $\langle c\,, L\rangle$, where c is a constraint and L is a list of subgoals. The empty iCS-state is $\langle \emptyset\,, [\,]\rangle$.*

Lemma 6.1 *Let c be a constraint.*

- *$i\mathcal{N}(L_c) = L' = c'$ with $(\tau, \emptyset) \bullet L''$ iff there exists a terminal $i\mathcal{CNC}$-state $\langle 1, L' \rangle$ such that $\sigma = mgu(c')$, $\theta = \tau\sigma$ is a solution for c and the list L' represents an execution state of the innermost conditional narrowing algorithm when θ is found.*

Definition 6.6 *(iCS* transition relation *)* $\stackrel{\tilde{c}}{\hookrightarrow}_{iCS}$

$$\frac{L' = i\mathcal{N}(merge(L, \tilde{c}))}{\langle c\,, L\rangle \quad \stackrel{\tilde{c}}{\hookrightarrow}_{iCS} \quad \langle c \cup \tilde{c}\,, L'\rangle}$$

where

$merge([\,], \tilde{c}) = [\tilde{c} \ with \ (\epsilon, O'(\tilde{c}))]$

$merge([c_i \ with \ (\tau_i, Occ_i)]_{i=1}^{n\geq 1}, \tilde{c}) = [(c_i, \tilde{c}\tau_i) \ with \ (\tau_i, Occ_i \cup O(c_i, \tilde{c}))]_{i=1}^{n}$

Roughly speaking, in definition 6.6, L is thought as the representation in form of a list of the search tree for the constraint c. The following lemma shows that the function $merge(L, \tilde{c})$ merges L and the constraint $\tilde{c}$ building the representation of a search tree for $c \cup \tilde{c}$.

Lemma 6.2 *Let L_c represent an innermost search tree for the constraint c denoting an execution state of the innermost conditional narrowing algorithm for c when the first solution to c is found. Then $merge(L_c, \tilde{c})$ represents an innermost search tree for $c \cup \tilde{c}$ corresponding to this state.*

Let us now illustrate Lemma 6.2 by an example.

Example 6.1 *Let us consider the equational theory $\mathcal{E}$ in Example 4.2. Figure 1 shows an innermost search tree for the constraint $c = \{U + Z = s(0)\}$ with respect to $\mathcal{E}$.*

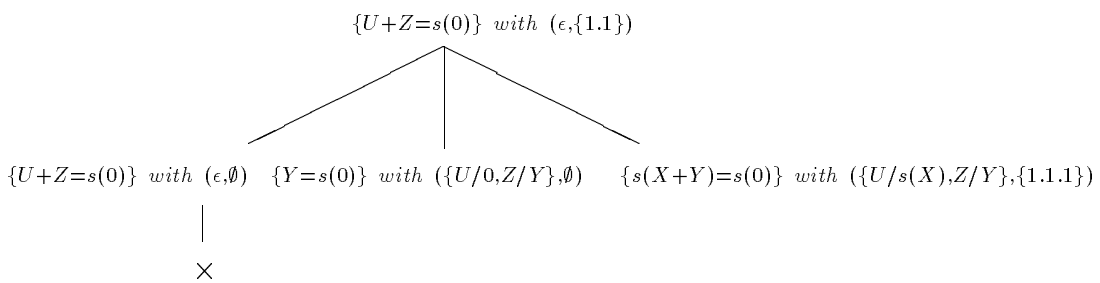

figure 1

The list $L_c = [\{Y = s(0)\}$ *with* $(\{U/0, Z/Y\}, \emptyset)$, $\{s(X+Y) = s(0)\}$ *with* $(\{U/s(X), Z/Y\}, \{1.1.1\})]$ *of the leaves of the tree in Figure 1 represents the tree.*

Let $\tilde{c} = \{U + 0 = s(W)\}$ *be a new constraint. The list* $L_{c\cup\tilde{c}} = merge(L_c, \tilde{c}) = [\{Y = s(0), 0+0 = s(W)\}$ *with* $(\{U/0, Z/Y\}, \{2.1\})$, $\{s(X+Y) = s(0), s(X)+0 = s(W)\}$ *with* $(\{U/s(X), Z/Y\}, \{1.1.1, 2.1\})]$ *represents the innermost search tree for the constraint* $c \cup \tilde{c}$ *which is shown in Figure 2. Note that the leftmost node in this tree can be dropped since the equation* $U + Z = s(0)$ *in the node does not unify and no occurrence in the set* $\{2.1\}$ *of the narrowable occurrences corresponds to this equation.*

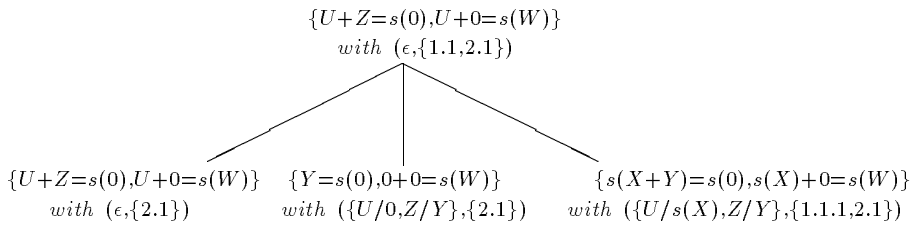

figure 2

The strategy defined by the calculus iCS can be proved complete by using the following result of completeness of the conditional selection narrowing in [44].

Proposition 6.3 *[23], [44]*
Let $\mathcal{E}$ *be a conditional equational theory such that the associated term rewriting system is noetherian and level confluent. Then the conditional selection narrowing algorithm is complete for* $\mathcal{E}$.

Theorem 6.1 (Correctness and Completeness of iCS)
An iCS *transition* $\langle c\,, L_c\rangle \overset{\tilde{c}}{\hookrightarrow}_{iCS} \langle c \cup \tilde{c}\,, L'\rangle$ *is performed iff the constraint* $c \cup \tilde{c}$ *is solvable in* $\mathcal{H}/\mathcal{E}$, *and* L' *represents an innermost search tree of* $c \cup \tilde{c}$.

Let us notice that the above described techniques can also be easily adapted to the optimization of the Incremental Constraint Solver ICS described in Section 4.2..

7. Conclusions

In this paper we have considered the language CLP($\mathcal{H}/\mathcal{E}$), its semantics and its constraint solver. On the semantics side, we have first defined an operational semantics $SS(P, \Re)$ and an equivalent fixpoint semantics $Fix(P, \Re)$ which fully characterize the operational behaviour of CLP programs. We have given a notion of n-model based

on the truth on (standard) $\mathcal{R}$-interpretations [34] and we have defined a complete partial order on n-models. This semantics is similar to the S-semantics of pure logic programs and turns out to be the adequate semantic framework for program analysis and transformation. This semantics is inherited by every instance of CLP and, in particular, by CLP($\mathcal{H}/\mathcal{E}$).

Then we have presented, in a formal setup, several strategies to obtain an incremental narrower as constraint solver for canonical Horn equational theories. We have presented a calculus which allows to reuse the substitutions discarded as a heuristic to look for other solutions. Finally we have given a calculus which builds incrementally the search tree which represents the set of solutions to a constraint.

Our methods fits in the CLP scheme, as opposed to the standard way to integrate narrowing and resolution [24] where narrowing generates the solutions which are then tested by the logic program.

The CLP framework we have considered is the one originally introduced by Jaffar and Lassez [18, 33]. Our construction however could equivalently be defined in other CLP frameworks, such as the one proposed in [27].

References

[1] M. Alpuente and M. Falaschi. Narrowing as an Incremental Constraint Satisfaction Algorithm. In J. Maluszyński and M. Wirsing, editors, *Proc. of PLILP'91*, volume 528 of *Lecture Notes in Computer Science*, pages 111–122. Springer-Verlag, Berlin, 1991.

[2] M. Alpuente, M. Falaschi, and G. Levi. Incremental Constraint Satisfaction for Equational Logic Programming. Technical Report TR-20/91, Dipartimento di Informatica, Università di Pisa, 1991. To appear on *Theoretical Computer Science*.

[3] M. Alpuente, M. Falaschi, and F. Manzo. Analyses of unsatisfiability for incremental equational logic programming. In M. Bruynooghe and M. Wirsing, editors, *Proc. of PLILP'92*, volume 631 of *Lecture Notes in Computer Science*, pages 443–457. Springer-Verlag, Berlin, 1992.

[4] M. Bellia and G. Levi. The relation between logic and functional languages. *Journal of Logic Programming*, 3:217–236, 1986.

[5] P. Bosco, E. Giovannetti, and C. Moiso. Refined strategies for semantic unification. In H. Ehrig, R. Kowalski, G. Levi, and U. Montanari, editors, *Proc. of TAPSOFT'87*, volume 250 of *Lecture Notes in Computer Science*, pages 276–290. Springer-Verlag, Berlin, 1987.

[6] P. Bosco, E. Giovannetti, and C. Moiso. Narrowing vs. SLD-resolution. *Theoretical Computer Science*, 59:3–23, 1988.

[7] P.G. Bosco, E. Giovannetti, G. Levi, and C. Palamidessi. A complete semantic characterization of K-leaf, a logic language with partial functions. In *Proceedings of the IEEE Symposium on Logic Programming*, pages 318–327, 1987.

[8] N. Dershowitz and M. Okada. A rationale for conditional equational programming. *Theoretical Computer Science*, 75:111–138, 1990.

[9] N. Dershowitz and A. Plaisted. Logic Programming cum Applicative Programming. In *Proc. First IEEE Int'l Symp. on Logic Programming*, pages 54–66. IEEE, 1984.

[10] M. Dincbas and P. van Hentenryck. Extended Unification Algorithms for the Integration of Functional Programming into Logic Programming. *Journal of Logic Programming*, 4:197–227, 1987.

[11] R. Echahed. On completeness of narrowing strategies. In *Proc. of CAAP'88*, volume 299 of *Lecture Notes in Computer Science*, pages 89–101. Springer-Verlag, Berlin, 1988.

[12] N. Heintze et al. *The CLP Programmer's Manual.* Dept. of Computer Science, Monash University, Australia, 1986.

[13] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative Modeling of the Operational Behavior of Logic Languages. *Theoretical Computer Science*, 69(3):289–318, 1989.

[14] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs. Technical Report TR 32/89, Dipartimento di Informatica, Università di Pisa, 1989. To appear in *Information and Computation.*

[15] M. Fay. First Order Unification in an Equational Theory. In *4th Int'l Conf. on Automated Deduction*, pages 161–167, 1979.

[16] L. Fribourg. Slog: a logic programming language interpreter based on clausal superposition and rewriting. In *Proc. Second IEEE Int'l Symp. on Logic Programming*, pages 172–185. IEEE, 1985.

[17] U. Furbach, S. Hölldobler, and J. Schreiber. Horn equality theories and paramodulation. *Journal of Automated Reasoning*, 5:309–337, 1989.

[18] M. Gabbrielli and G. Levi. Modeling Answer Constraints in Constraint Logic Programs. In K. Furukawa, editor, *Proc. Eighth Int'l Conf. on Logic Programming*, pages 238– 252. The MIT Press, Cambridge, Mass., 1991.

[19] M. Gabbrielli and G. Levi. Modeling answer constraints in Constraint Logic Programs. Technical Report TR 4/91, Dipartimento di Informatica, Università di Pisa, 1991.

[20] J.H. Gallier and S. Raatz. Extending SLD-resolution to equational Horn clauses using E-unification. *Journal of Logic Programming*, 6:3–43, 1989.

[21] J.H. Gallier and W. Snyder. A general complete E-unification procedure. In P. Lescanne, editor, *Rewriting Techniques and Applications*, volume 256 of *Lecture Notes in Computer Science*, pages 216–227. Springer-Verlag, Berlin, 1987.

[22] E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel Leaf: A Logic plus Functional Language. *Journal of Computer and System Sciences*, 42:139–185, 1991.

[23] E. Giovannetti and C. Moiso. A completeness result for E-unification algorithms based on Conditional Narrowing. In M. Boscarol, L. Carlucci, and G. Levi, editors, *Foundations of Logic and Functional Programming*, volume 306 of *Lecture Notes in Computer Science*, pages 157–167. Springer-Verlag, Berlin, 1986.

[24] J. Goguen and J. Meseguer. Eqlog: Equality, Types and Generic Modules for Logic Programming. In D. de Groot and G. Lindstrom, editors, *Logic Programming, Functions, Relations and Equations*, pages 295–363. Prentice-Hall, 1986.

[25] P. Van Hentenryck. Incremental Constraint Satisfaction in logic programming. In D. H. D. Warren and P. Szeredi, editors, *Proc. Seventh Int'l Conf. on Logic Programming*, pages 189–202. The MIT Press, Cambridge, Mass., 1990.

[26] P. Van Hentenryck and Y. Deville. Operational Semantics of Constraint Logic Programming over Finite Domains. In J. Maluszyński and M. Wirsing, editors, *Proc. of PLILP'91*, volume 528 of *Lecture Notes in Computer Science*, pages 395–406. Springer-Verlag, Berlin, 1991.

[27] M. Höhfeld and G. Smolka. Definite relations over constraint languages. Technical report, IBM Deutschland GmbH, Stuttgart, 1988.

[28] S. Hölldobler. Equational Logic Programming. In *Proc. Second IEEE Symp. on Logic In Computer Science*, pages 335–346. IEEE Computer Society Press, 1987.

[29] S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1989.

[30] S. Hölldobler. Conditional equational theories and complete sets of transformations. *Theoretical Computer Science*, 75:85–110, 1990.

[31] J.M. Hullot. Canonical Forms and Unification. In *5th Int'l Conf. on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Springer-Verlag, Berlin, 1980.

[32] H. Hussman. Unification in conditional-equational theories. Technical report, Fakultät für Mathematik und Informatik, Universität Passau, 1986.

[33] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proc. Fourteenth Annual ACM Symp. on Principles of Programming Languages*, pages 111–119. ACM, 1987.

[34] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. Technical report, Department of Computer Science, Monash University, June 1986.

[35] J. Jaffar, J.-L. Lassez, and M.J. Maher. A logic programming language scheme. In D. de Groot and G. Lindstrom, editors, *Logic Programming, Functions, Relations and Equations*, pages 441–468. Prentice Hall, Englewood Cliffs, NJ, 1986.

[36] J. Jaffar and S. Michaylov. Methodology and Implementation of a CLP System. In J.-L. Lassez, editor, *Proc. Fourth Int'l Conf. on Logic Programming*. The MIT Press, Cambridge, Mass., 1987.

[37] S. Kaplan. Conditional Rewrite Rules. *Theoretical Computer Science*, 33:175–193, 1984.

[38] S. Kaplan. Fair conditional term rewriting systems: unification, termination and confluence. In H.-J. Kreowski, editor, *Recent Trends in Data Type Specification*, volume 116 of *Informatik-Fachberichte*, pages 136–155. Springer-Verlag, Berlin, 1986.

[39] J.-L. Lassez, M. J. Maher, and K. Marriott. Unification Revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, Los Altos, Ca., 1988.

[40] G. Levi. Models, Unfolding Rules and Fixpoint Semantics. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth Int'l Conf. on Logic Programming*, pages 1649–1665. The MIT Press, Cambridge, Mass., 1988.

[41] G. Levi and P. Mancarella. The Unfolding Semantics of Logic Programs. Technical Report TR-13/88, Dipartimento di Informatica, Università di Pisa, 1988.

[42] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1987. Second edition.

[43] A. Martelli, C. Moiso, and G.F. Rossi. An algorithm for unification in equational theories. In *Proc. IEEE Symp. on Logic Programming*, pages 180–186. IEEE, 1986.

[44] A. Middeldorp and E. Hamoen. Counterexamples to completeness results for basic narrowing. In H. Kirchner and G. Levi, editors, *Proc. Third Int'l Conf. on Algebraic and Logic Programming*, volume 632 of *Lecture Notes in Computer Science*, pages 244–258. Springer-Verlag, Berlin, 1992.

[45] J.J. Moreno and M. Rodriguez-Artalejo. BABEL: A Functional and Logic Programming Language based on a constructor discipline and narrowing. In I. Grabowski, P. Lescanne, and W. Wechler, editors, *Algebraic and Logic Programming*, volume 343 of *Lecture Notes in Computer Science*, pages 223–232. Springer-Verlag, Berlin, 1988.

[46] J.J. Moreno and M. Rodriguez-Artalejo. Logic Programming with Functions and Predicates: The language Babel. *Journal of Logic Programming*, 12(3):191–224, 1992.

[47] W. Nutt, P. Réty, and G. Smolka. Basic narrowing revisited. *Journal of Symbolic Computation*, 7:295–317, 1989.

[48] P. Padawitz. *Computing in Horn Clause Theories*, volume 16 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1988.

[49] G. Plotkin. A structured approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[50] U.S. Reddy. Narrowing as the Operational Semantics of Functional Languages. In *Proc. Second IEEE Int'l Symp. on Logic Programming*, pages 138–151. IEEE, 1985.

[51] J.H. Siekmann. Universal unification. In *7th Int'l Conf. on Automated Deduction*, volume 170 of *Lecture Notes in Computer Science*, pages 1–42. Springer-Verlag, Berlin, 1984.

[52] J.H. Siekmann. Unification Theory. *Journal of Symbolic Computation*, 7:207–274, 1989.

[53] H. Tamaki. Semantics of a Logic Programming Language with a Reducibility Predicate. In *Proc. First IEEE Int'l Symp. on Logic Programming*, pages 259–264. IEEE, 1984.

[54] M.H. van Emden and K. Yukawa. Logic Programming with Equations. *Journal of Logic Programming*, 4:265–288, 1987.

[55] D. A. Wolfram. Intractable unifiability problems and backtracking. In E. Y. Shapiro, editor, *Proc. Third Int'l Conf. on Logic Programming*, volume 225 of *Lecture Notes in Computer Science*, pages 107–121. Springer-Verlag, Berlin, 1986.