Incremental Constraint Satisfaction for Equational Logic Programming *

María Alpuente ‡ Moreno Falaschi § Giorgio Levi§

Abstract

In this paper we are concerned with an instance of the CLP scheme specialized in solving equations with respect to a Horn equational theory $\mathcal{E}$. The intended structure $\mathcal{H}/\mathcal{E}$ is given by the finest partition induced by $\mathcal{E}$ on the Herbrand Universe $\mathcal{H}$ over a finite one sorted alphabet. This work deals with the description of an incremental constraint solver as the kernel of an operational semantics for the language CLP($\mathcal{H}/\mathcal{E}$). The primary issues are: how to verify the solvability of constraints in the structure $\mathcal{H}/\mathcal{E}$ by using some sound and complete semantic unification procedure such as narrowing, how to simplify constraints in a computation sequence, how to achieve incrementality in the computation process and how to profit from finitely failed derivations as a heuristic for optimizing the algorithms.

1 Introduction

In the last few years several approaches to the integration of logic and equational programming have been developed [6, 15, 19, 24, 26, 31, 35, 11, 50, 51]. One relevant approach [31, 39] defines equational logic programs as logic programs which are augmented by Horn equational theories. These programs admit least model and fixpoint semantics. 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 some special form of equational resolution (such as SLDE resolution [9, 22, 26, 31]), 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* [16, 56, 57]). $\mathcal{E}$-*unification* means the general

*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

unification w.r.t. an equational theory $\mathcal{E}$, which is usually described by a set of equations (i.e., unification in a framework where two terms which are not syntactically identical can be considered equal under the equational theory in consideration). It is well known that 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 [10, 11, 58, 59], complete sets of transformations [23, 32, 47] and paramodulation [20, 53] or some special form of it, such as superposition [19] or narrowing [10, 14, 25, 34, 35, 43, 52, 55]. In [31], a lazy resolution rule is defined in order to overcome the problem of nontermination of the $\mathcal{E}$-*unification* procedure.

Recently, the logic programming paradigm has been generalized to the framework of Contraint Logic Programming (CLP), a family of rule-based constraint programming languages defined in [37, 36]. CLP is a generic logic programming language scheme which extends pure logic programming to include constraints. In contrast to logic programming, in CLP the domain of computation is left unspecified, though it is assumed to be definable by a satisfaction complete theory $\Im$ as defined in [36]. Roughly speaking, satisfaction completeness means that every constraint is either provably satisfiable or provably unsatisfiable.

Each instance CLP($\mathcal{X}$) of the scheme is a programming language obtained by specifying a structure $\mathcal{X}$ of computation. The structure defines the underlying domain of discourse and the operations and relations on this domain, thus giving a semantic interpretation to it. For example, pure logic programming is simply an instance of CLP, where the corresponding structure has the Herbrand Universe $\mathcal{H}$ over a finite alphabet as domain and where the only predicate symbol for constraints is $=$, which is interpreted as syntactic equality over $\mathcal{H}$. The unification algorithm is now viewed as a special purpose constraint solver testing the solvability of equalities on the Herbrand domain.

The CLP scheme guarantees that the fundamental semantic properties of definite clauses are preserved in these extensions. The existence of a canonical domain of computation, least and greatest model semantics, the existence of a least and greatest fixpoint semantics, the soundness and completeness results for successful derivations and the soundness and completeness results for finite failure and negation as failure are inherited by any extension which can be formalized as an instance of the scheme [40]. Suitable models which correspond to different observable behaviours have been developed for the CLP scheme [21]. These results apply to each instance of the scheme.

The CLP scheme provides a unified view of several powerful extensions to pure logic programming, as object-oriented features [8, 1], real arithmetics [41] or infinite terms and disequalities [38]. In this paper we are concerned with an instance of the CLP scheme specialized in solving equations with respect to a Horn equational theory $\mathcal{E}$ [31, 39]. 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 to be considered in CLP must be *solution compact* as defined in [37, 36]. 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. As pointed in [36], 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 work 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 [27] and our results rely on the innermost selection narrowing defined in [10]. We discuss the following issues:

- How to verify the solvability of constraints in the structure $\mathcal{H}/\mathcal{E}$ by using some sound and complete semantic unification procedure, such as narrowing.
- How to simplify constraints in a computation sequence.
- How to achieve incrementality in the computation process.
- How to profit from finitely failed derivations as a heuristic for optimizing the algorithms to achieve an intelligent narrowing.

This paper is organized as follows. In Section 2 we briefly recall the basic concepts of the CLP framework, conditional term rewriting systems and universal unification. In Section 3 we define CLP($\mathcal{H}/\mathcal{E}$) logic programs and an incremental constraint solver as a kernel of an operational semantics for them. Section 4 is devoted to the heuristic narrowing calculus. Section 5 presents the calculus which builds incrementally the search tree of a constraint.

We assume the reader to be familiar with logic programming [46], constraint logic programming [21, 37, 36], equations and conditional rewriting systems [42] and universal unification [56].

A preliminary short version of this paper appeared in [3]. [2, 5] investigate the application of the language CLP($\mathcal{H}/\mathcal{E}$) to databases and [2] shows an initial prototype of the language.

2 Preliminaries

In this paper we refer to a language which is an instance of CLP, as defined in [37, 36]. We first recall the basic concepts of the CLP framework. Let $SORT = \bigcup SORT_i$ be a finite set of sorts. A *signature* of an n-ary function, predicate or variable symbol f is a sequence of respectively $n+1, n$ and 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 denumerable collections of function symbols, predicate symbols and variable symbols with their signatures. We assume that each sort is non-empty. $\tau(\Sigma \cup V)$ and $\tau(\Sigma)$ denote the sets of terms and ground terms built on Σ and V, respectively. 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) finite set of (Π, Σ)-atoms. Intuitively, a constraint is a conjunction of (Π, Σ)-atoms. The empty constraint will be denoted by $true$. The symbol $\tilde{}$ will denote a finite sequence of symbols.

$\tau(\Sigma)$ is usually called the Herbrand Universe ($\mathcal{H}_\Sigma$) over Σ. We will drop Σ if it is fixed and clear from the context and we will refer to $\tau(\Sigma)$ simply by $\mathcal{H}$.

DEFINITION **2.1**. (CLP programs) *[37] 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*

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

where c is a possibly empty finite (Π_C, Σ)-constraint, H (the head) and $B_1, \ldots, B_n$ (the body), $n \geq 0$, are (Π_B, Σ)-atoms. A goal *is a program clause with no head.*

In the CLP($\mathcal{X}$) framework, $\mathcal{X}$ stands for the *structure* which specifies the domain on which computations are performed. It gives the semantic interpretation for functions and relations and is the key element in the algebraic semantics.

DEFINITION **2.2**. *[37] 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 $D\Re_{s_1} \times \ldots \times D\Re_{s_n} \to D\Re_s$ 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 $=$, of a function $D\Re_{s_1} \times \ldots \times D\Re_{s_n} \to \{True, False\}$, where $(s_1, s_2, \ldots, s_n)$ is the signature of p.*

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

(Π, Σ)-programs, (Π_B, Σ)-atoms, (Π_C, Σ)-constraints and (Π_B, Σ)-constrained atoms will often be called programs, atoms, 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 **2.3**. (solvability) *[37] A constraint c is $\Re$-solvable (in symbols $\Re \models (\tilde{\exists})c$) 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.*

DEFINITION **2.4**. *An expression is either a term, a sequence of terms, an atom or a conjunction of atoms.*

DEFINITION **2.5**. *[37] A (ground) (Π, Σ)-substitution is an endomorphism θ: $\tau(\Sigma \cup V) \rightarrow \tau(\Sigma \cup V)$ on the term algebra $\tau(\Sigma \cup V)$, which is identical almost everywhere on V and hence can be represented as a finite set $\{x_1/t_1, ..., x_k/t_k\}$ where*

- *the t_i, $i = 1, ..., k$, are (ground) terms over Σ which do not contain any occurrences of x_j, $j = 1, ..., k$*
- *x_i and t_i have the same sort, $i = 1, ..., k$.*

The equational representation of a substitution $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ is the set of equations, $\widehat{\theta} = \{x_1 = t_1, \ldots, x_n = t_n\}$. The empty substitution is denoted by ϵ.

For any substitution θ and set of variables $W \subseteq V$, we define θ restricted to W, denoted by $\theta_{\upharpoonright W}$, to be $\{(x/t) \mid (x/t) \in \theta \wedge x \in W\}$. $var(e)$ is the set of distinct variables occurring in the expression e. In abuse of notation, $Dom(\sigma) = \{x \in W \mid x\sigma \neq x\}$ is called the domain of σ and $Cod(\sigma) = \{x\sigma \mid x \in Dom(\sigma)\}$ is called the codomain of σ. We denote by $VCod(\sigma) = var(Cod(\sigma))$ the set of variables introduced by σ.

The notions of application, composition and relative generality are defined in the usual way [45, 46, 57], e.g., a substitution θ is more general than γ, denoted by $\theta \leq \gamma$, if there exists a (Π, Σ)-substitution δ such that $\gamma = \theta\delta$.

DEFINITION **2.6**. *A substitution θ is a unifier of a set of equations $c = \{t_1 = t'_1, \ldots, t_n = t'_n\}$ iff $t_1\theta = t'_1\theta \ \wedge \ \ldots \ \wedge \ t_n\theta = t'_n\theta$. We say that θ is a most general unifier (mgu) for c iff for every unifier θ' of c, $\exists\delta$ such that $\theta' = \theta\delta$.*

DEFINITION **2.7**. *[46] Let c be a set of equations. We denote as $mgu(c)$ the set of most general unifiers of c.*

In this paper we follow the operational approach to semantics which is based on transition systems following the standard terminology as in Plotkin's approach [54]. Thus, to specify the transition system CLP($\mathcal{X}$) we need to show the set of configurations (states), the transition relation $\rightarrow_{CLP}$ and the sets of terminal configurations and labels. [1]

DEFINITION **2.8**. CLP($\mathcal{X}$) configurations
Let P be a constraint logic program and $G = \leftarrow c\Box A_1, \ldots, A_n$. be a goal *over the structure $\mathcal{X} = \Re(\Pi, \Sigma)$. 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.*

[1] In [28] a structural operational approach is also used to define the semantics for a class of CLP languages. However our approach is different as our algorithms are incremental.

DEFINITION **2.9**. 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*

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

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* $s_i \stackrel{\tilde{c}}{\hookrightarrow} s_{i+1}$ *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 **2.10**. 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* $s_0 \stackrel{c_0}{\hookrightarrow} s$ *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 **2.11**. 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 [37]:

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

In this case, the concepts of (P,$\Re$)-derivation and successful and finitely failed (P,$\Re$)-derivations can be defined in the usual way [37]. The constraint in the last goal of a successful derivation is the answer constraint of the derivation.

Let us next briefly recall some basic notions and results about equations, conditional rewrite systems and universal unification. Full definitions can be found in [13, 42, 44, 56].

DEFINITION **2.12**. *Terms are viewed as labelled trees in the standard way [33, 34]. A labelled tree* t *is a partial function from the set* $\mathcal{N}^{+*}$ *of sequences of positive integers to* $\Sigma \bigcup V$ *whose domain* $O(t)$ *satisfies:*

- $O(t)$ *is not empty and prefix closed*
- *if* $u \in O(t)$ *and* $t[u]$ *is a n-ary symbol, then* $u.i \in O(t)$ *iff* $i \in \{1, ..., n\}$.

$O(t)$ denotes the set of occurrences (sequences of positive integers denoting an access path in a term, with the empty sequence represented by λ) of the term t and is partially ordered by the prefix ordering: $u \leq v$ iff $\exists w u.v = w$. We use t/u to denote the subterm of t at occurrence $u \in O(t)$ defined as follows:

- $t/\lambda = t$
- $f(t_1, t_2, ..., t_n)/i.u = t_i/u$

and $t[u \leftarrow s]$ to denote the term obtainable from t by replacing the subterm t/u by s. $t[u]$ denotes the label in t at occurrence $u \in O(t)$

The concepts of occurrence set, subterm and subterm replacement can be extended in an obvious manner to equations and conjunctions of equations, as shown in [35]. For instance, the definition of occurrence set for a conjunction of equations would be as follows:
$O(t_1 = t'_1, \ldots, t_n = t'_n) = \{i.1.u \mid i \in \{1, \ldots, n\}, u \in O(t_i)\} \bigcup \{i.2.u \mid i \in \{1, \ldots, n\}, u \in O(t'_i)\}$

Definition **2.13**. *Let $\rightarrow$ be a binary relation on a set S. Then we use $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$. The relation $\rightarrow$ is called confluent if $\forall s_1, s_2, s_3 \in S$ such that $s_1 \rightarrow^* s_2 \wedge s_1 \rightarrow^* s_3$, $\exists s \in S$ such that $s_2 \rightarrow^* s$ and $s_3 \rightarrow^* s$.*

The relation $\rightarrow$ is called noetherian if there is no infinite chain $s1 \rightarrow s2 \rightarrow s3 \rightarrow \ldots$

Definition **2.14**. *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 $l = r \leftarrow e_1, e_2, ..., e_n$, $n \geq 0$, 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 [7, 11, 50]. We need the following:

Definition **2.15**. *[25, 49]Level Confluence*
Let $\rightarrow_\mathcal{R}$ be the one-step rewriting relation corresponding to a Horn equational theory $\mathcal{E}$. Then $\rightarrow_\mathcal{R}$ is equivalent to $\rightarrow_\mathcal{R} = \cup_{i \geq 0}\{\xrightarrow{i}_\mathcal{R}\}$, where:

1. *$\xrightarrow{0}_\mathcal{R}$ is the empty relation;*
2. *$t \xrightarrow{n+1}_\mathcal{R} s$ holds iff there is a rule $l = r \leftarrow e_1 = e'_1 \ldots e_m = e'_m \in \mathcal{R}$, $m \geq 0$, a non-variable occurrence u of t, a substitution σ and m terms γ_i such that $t/u = l\sigma$, $e_i\sigma \xrightarrow{n}{}^*_\mathcal{R} \gamma_i$, $e'_i\sigma \xrightarrow{n}{}^*_\mathcal{R} \gamma_i$ $i = 1, \ldots, m$ and $s = t[u \leftarrow r\sigma]$.*

The relation $\rightarrow_R$ is called level-confluent [25] iff, for each $n \geq 0$, the relation $\xrightarrow{n}_R$ is confluent.

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 **2.16**. *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[2], $\mathcal{E} \models s\sigma = t\sigma$. The substitution σ is called an $\mathcal{E}$-unifier of s and t.*

Given a set of variables $W \subseteq V$, $\mathcal{E}$-equality is extended to substitutions in the standard way, by $\sigma =_\mathcal{E} \theta[W]$ iff $x\sigma =_\mathcal{E} x\theta$ $\forall x \in W$. W will be omitted if equal to V.

DEFINITION **2.17**. *We say σ' is an $\mathcal{E}$-instance of σ and σ is more general than σ' on W, in symbols $\sigma \leq_\mathcal{E} \sigma'[W]$ iff $(\exists \rho)\ \sigma' =_\mathcal{E} \sigma\rho[W]$. Two substitutions σ, σ' are $\mathcal{E}$-equivalent on W, in symbols $\sigma \equiv_\mathcal{E} \sigma'[W]$ iff $\sigma' \leq_\mathcal{E} \sigma[W]$ and $\sigma \leq_\mathcal{E} \sigma'[W]$.*

DEFINITION **2.18**. *Let e be an equation and $W = var(e)$. An $\mathcal{E}$-unifier σ of e is called a most general $\mathcal{E}$-unifier (mgu over $\mathcal{E}$) of e on W if for any $\mathcal{E}$-unifier σ' of e $\sigma \leq_\mathcal{E} \sigma'[W]$.*

DEFINITION **2.19**. *A set $U_\mathcal{E}$ of substitutions is said to be a complete set of $\mathcal{E}$-unifiers of e on W iff*

1. *For all $\sigma \in U_\mathcal{E}$, σ is an $\mathcal{E}$-unifier of e* (coherence)
2. *For all $\mathcal{E}$-unifier σ of e, $\exists\ \sigma' \in U_\mathcal{E}$ such that $\sigma' \leq_\mathcal{E} \sigma[W]$.* (completeness)

   The set $U_\mathcal{E}$ is said to be minimal iff it satisfies the further condition:
3. *$\forall\ \sigma, \sigma' \in U_\mathcal{E}$, $\sigma \leq_\mathcal{E} \sigma'[W] \Rightarrow \sigma = \sigma'$* (minimality)

   For practical reasons it is useful to have the additional technical requirement:
4. *For a set of variables $Z \supseteq W$, $(\forall\ \sigma \in U_\mathcal{E})$ $Dom(\sigma) \subseteq W$ and $VCod(\sigma) \cap Z = \emptyset$* (purity or protection of Z)

Let us notice that this additional condition implies that *$\mathcal{E}$-unifiers* θ are idempotent $(\theta\theta = \theta)$ since $Dom(\sigma) \cap VCod(\sigma) = \emptyset$.

Minimal complete sets of *$\mathcal{E}$-unifiers* do not always exists. When they do, they are unique, up to equivalence generated by $\leq_\mathcal{E}$. In the literature on universal unification, the set $U_\mathcal{E}$ satisfying the conditions (1-4) of Definition 2.19 is called the set of maximally general *$\mathcal{E}$-unifiers* [39] or most general unifiers over $\mathcal{E}$ [56] away from Z.

$\mathcal{E}$-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

[2] 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.

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 [15, 19, 24, 32, 35, 42]. For instance, conditional narrowing has been shown to be complete for conditional theories satisfying different restrictions [15, 14, 25, 31, 42].

The extended concepts of occurrence set, subterm and subterm replacement allow to map $\mathcal{E}$-*unification* of conjunctions of equations into simple $\mathcal{E}$-*unification*. Therefore the same algorithms which solve a single equation can be applied.

DEFINITION **2.20**. *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 $l \in V$ or f occurs as the outermost function symbol in l; otherwise it is a defined function symbol.*

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 defined function symbols.

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

3.1 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 [36]. It is known [39] that the equational theory $\mathcal{E}$ and the structure $\mathcal{H}/\mathcal{E}$ correspond in the sense defined in [36]:

1. $\mathcal{H}/\mathcal{E} \models \mathcal{E}$, i.e. $\mathcal{H}/\mathcal{E}$ models $\mathcal{E}$ and
2. $\mathcal{H}/\mathcal{E} \models \tilde{\exists} c$ implies $\mathcal{E} \models \tilde{\exists} c$ for every constraint c

i.e. 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.

DEFINITION **3.1**. 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 1. *What follows is a CLP($\mathcal{H}/\mathcal{E}$) program $P \cup \mathcal{E}$:*

Number of nodes of a binary tree

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

CLP($\mathcal{H}/\mathcal{E}$) programs enjoy the following property [39]: 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 [37]. Unfortunately this is not possible. Hence the soundness and completeness results for finite failure and negation as failure are not inherited.

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

The standard way to integrate narrowing and resolution [9, 26] 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, at first glance, one can think about building a CLP($\mathcal{H}/\mathcal{E}$) system by simply embedding a narrowing algorithm or some similar kind of mechanism as the constraint solver within the *CLP* system. A constraint solver state $s[c]$ would contain nothing more than the constraint (set of equations) c. This naive constraint solver could be described by the following transition relation:

$$\frac{\textit{there is an } \mathcal{E}\text{-unifier of } \ c \cup \tilde{c}}{c \ \overset{\tilde{c}}{\hookrightarrow} \ c \cup \tilde{c}}$$

which involves too much redundant computation since a solvability test for the whole set of constraints is required at each transition and this set monotonically grows. The problem is that the solvability test might require a significant cost. 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 [27].

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 $c \cup \tilde{c}$. The constraint solver states $s[c]$ should contain this additional information. We describe this constraint solver by the following two rules, where a constraint solver state is expressed as a pair $s[c] = \langle\, \theta, c \,\rangle$ where θ represents a solution to c:

(1) Success Rule

$$\frac{\textit{there is an } \mathcal{E}\text{-unifier} \quad \theta' \quad \text{of} \quad \tilde{c}\theta}{\langle\ \theta, c\ \rangle \stackrel{\tilde{c}}{\hookrightarrow} \langle\ \theta\theta', c \cup \tilde{c}\ \rangle}$$

(2) Start from Scratch Rule

$$\frac{\textit{there is no } \mathcal{E}\text{-unifier} \quad \text{of} \quad \tilde{c}\theta \ \wedge \ \textit{there is an } \mathcal{E}\text{-unifier} \quad \theta' \quad \text{of} \quad c \cup \tilde{c}}{\langle\ \theta, c\ \rangle \stackrel{\tilde{c}}{\hookrightarrow} \langle\ \theta', c \cup \tilde{c}\ \rangle}$$

Let us notice that if rule (2) is applied then no instance of θ can solve $c \cup \tilde{c}$. This remark suggests a method to prune the search tree when we incrementally add a new constraint to be solved, as we will show later. Another main issue related to incrementality will be how to represent a solvable constraint in a simplified way.

In [27] the matter of incremental constraint satisfaction is also addressed. [27] 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, [27] 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.

Lemma **3.2**. *Let c be a solvable constraint in $\mathcal{H}/\mathcal{E}$. 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 the constraint c has the same set of $\mathcal{H}/\mathcal{E}$-solutions as the simplified constraint $\widehat{\theta} \cup c_2$*

Proof. We denote the (possibly empty) set of $\mathcal{H}/\mathcal{E}$-solutions of a constraint c by $solns(c)$. It is immediate that $solns(c' \cup c'') = solns(c') \cap solns(c'')$. Therefore, $solns(c) = solns(c_1 \cup c_2) = solns(c_1) \cap solns(c_2) = solns(\widehat{\theta}) \cap solns(c_2) = solns(\widehat{\theta} \cup c_2)$. □

Definition **3.3**. *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 **3.4**. *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 **3.5**. 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 **3.6**. *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 $c \cup \tilde{c}$.

LEMMA **3.7**. *Let $c, \tilde{c}$ be two solvable constraints in $\mathcal{H}/\mathcal{E}$. Let us suppose that θ is an $\mathcal{E}$-unifier of c. Then if θ' is an $\mathcal{E}$-unifier of the constraint $\tilde{c}\theta$ then $\theta\theta'$ is an $\mathcal{E}$-unifier of $c \cup \tilde{c}$.*

PROOF. We must show that $\theta\theta'$ is an $\mathcal{E}$-unifier of $c \cup \tilde{c}$. If a substitution γ solves an equation then every instance $\gamma\gamma'$ solves it [12, 39]. Since θ is an $\mathcal{E}$-unifier of c, also $\theta\theta'$ is an $\mathcal{E}$-unifier of c. Moreover, since θ' is an $\mathcal{E}$-unifier of $\tilde{c}\theta$ then $\theta\theta'$ is also an $\mathcal{E}$-unifier of $\tilde{c}$. So, $\theta\theta'$ is an $\mathcal{E}$-unifier of both c and $\tilde{c}$. $\square$

LEMMA **3.8**. *Let $c, \tilde{c}$ be two constraints. Let us suppose that c has a unique mgu θ over $\mathcal{E}$. Then if θ' is the unique mgu over $\mathcal{E}$ of $\tilde{c}\theta$ then $\theta\theta'$ is the unique mgu over $\mathcal{E}$ of $c \cup \tilde{c}$. If there is no $\mathcal{E}$-unifier of $\tilde{c}\theta$ then also $c \cup \tilde{c}$ has no $\mathcal{E}$-unifier.*

PROOF. If θ' is the unique mgu over $\mathcal{E}$ of $\tilde{c}\theta$ then $solns(c \cup \tilde{c}) = solns(c) \cap solns(\tilde{c}) = solns(\widehat{\theta}) \cap solns(\tilde{c}) = solns(\widehat{\theta}) \cap solns(\tilde{c}\theta) = solns(\widehat{\theta}) \cap solns(\widehat{\theta'}) = solns(\widehat{\theta} \cup \widehat{\theta'}) = solns(\widehat{\theta\theta'})$ since $Dom(\theta) \cap Dom(\theta') = \emptyset$ and $Dom(\theta) \cap VCod(\theta') = \emptyset$. Thus the first statement holds.

If $\tilde{c}\theta$ has no $\mathcal{E}$-unifier then $solns(c \cup \tilde{c}) = solns(\widehat{\theta}) \cap solns(\tilde{c}\theta) = solns(\widehat{\theta}) \cap \emptyset = \emptyset$. $\square$

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 **3.9**. *ICS* transition relation[3] $\xrightarrow{\tilde{c}}_{ICS}$

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

Single Solution Rules

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

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

Multiple Solution Rules

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

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

Start from Scratch Rules

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

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

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 to 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 **3.10**. (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.

PROOF. We prove that the statement holds for each rule. Let us notice that one and only one rule can be chosen at any step.

- Assume that (1) has been applied. This rule can be applied only if we have a constraint c whose unique mgu over $\mathcal{E}$ is θ. By lemma 3.8, if $\tilde{c}\theta$ has a unique mgu θ' over $\mathcal{E}$, then $c \cup \tilde{c}$ is solvable and $\theta\theta'$ is the only mgu over $\mathcal{E}$ for it.

- Assume that (2) has been applied. This rule can be applied only if we have a constraint c which has been simplified to $\widehat{\theta_1} \cup c_2$ and for which an $\mathcal{E}$-unifier θ is known, and if a unique mgu θ' over $\mathcal{E}$ of $\tilde{c}\theta$ has been found. Lemma 3.7 guarantees that $\theta\theta'$ is an $\mathcal{E}$-unifier of $c \cup \tilde{c}$. Moreover, since $\widehat{\theta_1}$ has a unique mgu θ_1 over $\mathcal{E}$ and θ'' is the unique mgu over $\mathcal{E}$ of $(c_2 \cup \tilde{c})\theta_1$, lemma 3.8 guarantees that $\theta_1\theta''$ is the unique mgu over $\mathcal{E}$ of $(\widehat{\theta_1} \cup c_2 \cup \tilde{c})$ and, by lemma 3.2, the simplified constraint $\widehat{\theta_1\theta''}$ has the same set of $\mathcal{H}/\mathcal{E}$-solutions as $(\widehat{\theta_1} \cup c_2 \cup \tilde{c}) = c \cup \tilde{c}$.

- Assume that (3) has been applied. This rule can be applied only if we have a constraint c which has been simplified to $\widehat{\theta_1} \cup c_2$ and for which an $\mathcal{E}$-unifier θ is known, and if a unique mgu θ' over $\mathcal{E}$ of $\tilde{c}\theta$ has been found. Lemma 3.7 guarantees that $\theta\theta'$ is an $\mathcal{E}$-unifier of $c \cup \tilde{c}$. Besides, since $\widehat{\theta_1}$ has a unique mgu θ_1 over $\mathcal{E}$, lemma 3.2 guarantees that the constraint $c \cup \tilde{c}$ has the same set of solutions as the simplified constraint $(\widehat{\theta_1} \cup c_2 \cup \tilde{c})$.

- Assume that (4) has been applied. This rule can be applied only if we have a constraint c which has been simplified to $\widehat{\theta_1} \cup c'$ and for which an $\mathcal{E}$-unifier θ is known, and if an $\mathcal{E}$-unifier θ' to $\tilde{c}\theta$ has been found. Lemma 3.7 guarantees that $\theta\theta'$ is an $\mathcal{E}$-unifier to $c \cup \tilde{c}$. Besides, since $\widehat{\theta_1}$ has a unique mgu over $\mathcal{E}$ θ_1, lemma 3.2 guarantees that the constraint $c \cup \tilde{c}$ has the same set of solutions as $(\widehat{\theta_1} \cup c' \cup \tilde{c})$.

- Assume that (5) has been applied. This rule can be applied only if we have a constraint c represented by $r_c = (\widehat{\theta_1}, c_2)$ for which an $\mathcal{E}$-unifier θ is known. If $\tilde{c}\theta$ has no $\mathcal{E}$-unifier then no instance of θ can solve $(\widehat{\theta_1}, c_2) \cup \tilde{c}$. So, the accumulated $\mathcal{E}$-unifier θ has to be thrown out and the whole constraint $(\widehat{\theta_1}, c_2) \cup \tilde{c}$ has to be tested for solvability, while θ has to be added to the set f of substitutions that will serve for a search of other solutions. Thus, if there exists a unique solution θ' to $(\widehat{\theta_1}, c_2) \cup \tilde{c}$ then by a similar argument as for (1) the theorem holds.

- Assume that (6) has been applied. Analogously to (5) θ has to be added to the set f of substitutions that will serve for a search of other solutions. Besides, this rule can be applied only if we have a constraint $(\widehat{\theta_1}, c_2) \cup \tilde{c}$ for which an $\mathcal{E}$-unifier θ' has been found, and if $\tilde{c}\theta$ has no solution. Then, the theorem holds and $r'_c = (\widehat{\theta_1}, c_2 \cup \tilde{c})$ represents the constraints in a not simplified form.

□

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 [28].

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

The following definitions instantiate definitions 2.9, 2.10 and 2.11 to the case of CLP($\mathcal{H}/\mathcal{E}$) programs.

Definition **3.11**. 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 **3.12**. 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 $c = (\widehat{\theta}_1 \cup c_2)_{\restriction var(G_0)}$[4] 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 **2**. *Let us consider the following CLP($\mathcal{H}/\mathcal{E}$) program[5] $P \cup \mathcal{E}$ [30]:*
0/1 Knapsack problem

$$\begin{array}{lllll}
P = & \{ & (p1) & knapsack(M, L, W) & \leftarrow addweight(M) = W \ \Box \ sublist(M, L). \\
 & & (p2) & sublist([\,], Z). & \\
 & & (p3) & sublist([X \mid Y], [X \mid Z]) & \leftarrow \ \Box \ sublist(Y, Z). \\
 & & (p4) & sublist(Y, [X \mid Z]) & \leftarrow \ \Box \ sublist(Y, Z).\ \} \\
\mathcal{E} = & \{ & (e1) & 0 + Y = Y. & \\
 & & (e2) & s(X) + Y = s(X + Y). & \\
 & & (e3) & weight(a) = 1. & \\
 & & (e4) & weight(b) = 1. & \\
 & & (e5) & weight(c) = 2. & \\
 & & (e6) & addweight([\,]) = 0. & \\
 & & (e7) & addweight([X \mid Y]) = weight(X) + addweight(Y).\ \} &
\end{array}$$

% 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 %

---

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

Let us consider the initial goal $G_0 = \leftarrow \ \Box\ knapsack(K,[a,b],2)$. *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*[6] $\theta = \{M/[a,a], K/[a,a], L/[a,b], W/2\}$, *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*[7] $\theta'' = \{M/[a,b], K/[a,b], L/[a,b], W/2, X/a, Y/[b], Z/[b], X'/b, Y'/[\,], Z'/[\,]\}$, *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* $\theta''' = \widehat{\{Z''/[\,]\}}$, *applying* (1),

$\mathcal{C}_3 \rightarrow_{CLP(\mathcal{H}/\mathcal{E})} \mathcal{C}_4 = \langle\ \leftarrow \langle \theta''\theta''', \theta\theta', (\widehat{\theta''\theta'''}, \emptyset)\rangle \ \diamond\ \rangle$

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

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.

4 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 [35] 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 [31]. A narrowing procedure for first order unification in an equational theory defined by a canonical and unconditional term rewriting system was given by Fay [18] and later improved by Hullot in [34]. Fay worked with normalizing narrowing and Hullot introduced the notion of basic narrowing. Nutt, Rety and Smolka [52] studied basic narrowing and its optimizations combined with normalizing narrowing. Bosco et al. [9] introduced narrowing selection. Lazy narrowing stems from Reddy [55]. Padawitz [53] presented refinements of narrowing and proved the correctness of the

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

[7] The narrowing calculus we define in Section 4 is able to find this solution and prove its uniqueness.

optimizations. Dershowitz [15], Hussmann [35] and Kaplan [43] 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 [31]. Giovannetti and Moiso [25] explored the consequences of allowing the presence of extra-variables and gave a sufficient condition for completeness. Hölldobler [31] generalized the results obtained by Hullot and Hussmann. Fribourg [19] employed an innermost case of strategy-controlled narrowing. Middeldorp [48] carefully analyzes the completeness results for conditional basic narrowing. Echahed [17] and Padawitz [53] studied criteria to ensure completeness of narrowing selection strategies.

Let us briefly recall the Conditional Narrowing Algorithm CNA presented in [35]. 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 $c_0 \ with \ \epsilon$ and tries to derive subgoals (recording applied substitutions in the with-parts) until a terminal goal of the form $\emptyset \ with \ \theta$ 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
$narred(c, u, k, \sigma) \Leftrightarrow (\ u \in O'(c) \quad \wedge \quad (l_k = r_k \leftarrow \tilde{e}_k) \in \mathcal{E} \ \wedge \ \ \sigma = mgu(c/u, l_k) \ )$

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 4.1. $\mathcal{N}$-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 $g_i = c_i \ with \ \tau_i$, 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 $\mathcal{N}_0 = \langle 0, \epsilon, [\, c_0 \ \mathit{with}\ \epsilon \,]\rangle$ and tries to derive subgoals (recording applied substitutions in the with-parts) until a terminal $\mathcal{N}$-state $\langle n, \theta, L\rangle \not\to_{\mathcal{HCNC}}$ 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 $c_i\ \mathit{with}\ \tau_i$ 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 $(\tilde{e}_{k'}, c_i[u' \leftarrow r_{k'}])\sigma'\ \mathit{with}\ \tau_i\sigma'$ 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 [60] 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 **4.2**. $\mathcal{H}$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[8]*:*

Heuristic Branching Rules

$$(1)\qquad \frac{\{\langle u, k, \sigma\rangle \mid \ narred(c, u, k, \sigma)\ \} = \emptyset}{c\ \mathit{with}\ \tau \to_{\mathcal{B}ranch} [\,]}$$

$$(2)\qquad \frac{s\_to\_l(\{\langle u, k, \sigma\rangle \mid narred(c, u, k, \sigma) \wedge \{\rho \in f \mid \rho \preceq \tau\sigma\} = \emptyset\}) = [\langle u_i, k_i, \sigma_i\rangle]_{i=1}^n}{c\ \mathit{with}\ \tau \to_{\mathcal{B}ranch} [\ (\tilde{e}_{k_i}, c[u_i \leftarrow r_{k_i}])\sigma_i\ \mathit{with}\ \tau\sigma_i\ ]_{i=1}^n}$$

where the choice function $s\_to\_l(S)$ returns a list composed by the elements of the set S.

Success Rules

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

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

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

Narrowing Rules

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

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

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

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 4.2 by the following:

Example 3. *Let us consider the constraint* $c = \{U + s(Z) = s(0)\}$ *within the equational theory* $\mathcal{E}$ *in Example 2. Let* $f = \{\{U/s(X)\}\}$. *The sequence* $\langle 0, \epsilon, [U + s(Z) = s(0) \quad with\ \epsilon]\rangle \xrightarrow{(7)}_{\mathcal{HCNC}} \langle 0, \epsilon, [s(Z) = s(0) \quad with \quad \{U/0\}]\rangle \xrightarrow{(3)}_{\mathcal{HCNC}} \langle 1, \{U/0, Z/0\}, [\,]\rangle$ *shows that* $\mathcal{HCNC}$ *is able to find the (unique) solution* $\{U/0,\ Z/0\}$ *for* c. *Notice that* $U + s(Z) = s(0) \quad with\ \epsilon \rightarrow_{Branch} [s(Z) = s(0) \quad with \quad \{U/0\}]$ *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 **4.3**. Behaviour of the $\mathcal{HCNC}$ calculus
Let $\mathcal{N}_0 = \langle 0, \epsilon, [\ c_0\ with\ \epsilon\ ]\rangle$. *We define:*

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

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 4.4. (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.*

Proof.

- Assume that the output *no solution* has been returned. Then the whole search tree of CNA has been explored (except for subtrees not containing any solution). Therefore, since CNA is complete, no solution exists, i.e., the constraint c is not solvable.
- Assume that the output *unique solution* θ has been returned. Then the whole search tree of CNA has been explored (except for subtrees not containing any solution) and a solution θ more general than all other possibly found solutions (if they exist) has been found. Then, since CNA is correct and complete, θ is the unique mgu over $\mathcal{E}$ of c.
- Assume that the output *first solution* θ has been returned. Then a prefix of the search tree of CNA has been explored (except for subtrees not containing any solution) and a solution θ has been found. Since CNA is correct, θ is a solution over $\mathcal{E}$ to c though, may be, it is not the unique mgu over $\mathcal{E}$ of 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 [34]. 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 [34], a sufficient condition for termination is given for canonical non-conditional theories.

To overcome these difficulties, several proposals have been made. In [31] 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($\Re$) [41]. [4] presents a static analysis which allows to approximate the problem of detection of unsatisfiability of a set of equations. [4] also shows how to use actively this analysis to prune the search tree of an incremental equational constraint solver.

4.1 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 [35, 19, 52]) 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 [35].

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 [35, 53]. Several strategies to reduce the number of subterms that have to be narrowed can be considered [10, 25, 34, 52]. The following section proposes an optimization strategy based on the results about selection narrowing presented in [10, 25].

5 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 [10]. 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 [10, 19]) in the set Occ. $\sharp c$ stands for the cardinality of the set c.

DEFINITION **5.1**. $\mathcal{I}$nnermost $\mathcal{C}$onditional $\mathcal{N}$arrowing $\mathcal{C}$alculus ($i\mathcal{CNC}$) *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. Terminal states have the form $\langle 1, L\rangle$*

The $i\mathcal{CNC}$ transition relation is the following[9]*:*

Don't-care Branching Rule

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

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

Unification Rule

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

Narrowing Rules

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

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

Roughly speaking, in Definition 5.1 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* (τ, 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 **5.2**. Behaviour of the $i\mathcal{CNC}$ calculus
Let us define the function:
$i\mathcal{N}(L) = \ L' \ \ if \ \ \langle 0, L\rangle \rightarrow^*_{i\mathcal{CNC}} \langle 1, L'\rangle$

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

We notice that $i\mathcal{N}$ is a function since the rules (2), (3) and (4) in Definition 5.1 are deterministic, i.e. if a rule applies then the other two do not.

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 [10] extended to conditional theories (innermost conditional narrowing for short).

DEFINITION **5.3**. *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 (τ, 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 **5.4**. *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 5.4 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 **5.5**. *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 **5.6**. *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.*

PROOF. We prove the claim by induction on the length n of the computation. We need the following

- Fact (1): Rule (1) establishes that if
  c *with* $(\tau, Occ) \rightarrow_{DcB} L' = c$ *with* $(\tau, (Occ \setminus \{u_0\})) \bullet [\ c'_i$ *with* $(\tau\sigma_i, Occ_i)\ ]_{i=1}^{n}$
  can be proven we have the following. Occ is the set of narrowable occurrences in c, one of the innermost ones (u_0) is nondeterministically chosen by the function *select-don't-care*(Occ), the list L' of subgoals corresponds to all possible narrowing steps from c at occurrence u_0 and $\tau\sigma_i$ consists of the composition of all the substitutions applied along the i-th narrowing path.

To prove Lemma 5.6 we need to show that the substitution in the with part of a subgoal corresponds to the composition of all the substitutions in a path of a innermost narrowing search tree.

By induction on the length n of a computation.

$n = 1$) There are two possibilities. Either rule (2) or rule (3) apply.

If rule (2) is applied then $\tau = \epsilon$ and there is a syntactic unifier for c which is also a solution to the constraint c.

If rule (3) is applied and the transition $\langle 0, cwith(\tau, Occ) \bullet L\rangle \rightarrow_{iC\mathcal{N}C} \langle 0, L \bullet L'\rangle$ is proven then, by the premise of the rule and fact (1), L' is the list of subgoals corresponding to the don't care branching of c at some innermost occurrence u_0 by all the possible applicable rewriting rules. The leftmost node $c\ with\ (\tau, Occ)$ is expanded, while its expansion L' is appended at the end of the new list $L \bullet L'$. Thus the nodes in L will be considered before those in L'. By Fact (1), if $c\ with\ (\epsilon, Occ) \rightarrow_{DcB} L' = c\ with\ (\epsilon, (Occ \setminus \{u_0\})) \bullet [\ c'_i\ with\ (\epsilon\sigma_i, Occ_i)\ ]_{i=1}^{n}$ one innermost occurrence u_0 of c is expanded and L' consists of a list of subgoals such that the with-parts of the i-th subgoal contains the substitution $\epsilon\sigma_i$ computed after one narrowing step (in the i-th narrowing path).

$n > 1$) The inductive case is similar to the basic case.

If rule (2) is applied then, by the inductive hypothesis we know that τ is the composition of the substitutions of a innermost narrowing path for the initial constraint and that c' is the constraint to be narrowed. Then, as c' syntactically unifies, $\sigma = mgu(c')$, and $\tau\sigma$ is a solution for the initial constraint.

If rule (3) is applied then by the inductive hypothesis we know that τ is the composition of the substitutions of a innermost narrowing path for the initial constraint and by Fact (1), if $c\ with\ (\tau, Occ) \rightarrow_{DcB} L' = c\ with\ (\tau, (Occ \setminus \{u_0\})) \bullet [\ c'_i\ with\ (\tau\sigma_i, Occ_i)\ ]_{i=1}^{n}$ one innermost occurrence u_0 of c is expanded and L' consists of a list of subgoals such that the with-part of the i-th subgoal contains the substitution $\tau\sigma_i$ (applied along the i-th narrowing path).

If rule (4) is applied and the transition $\langle 0,\ c\ with\ (\tau, \emptyset) \bullet L\rangle \rightarrow_{iC\mathcal{N}C} \langle 0,\ L\rangle$ is proven then the leftmost subgoal $c\ with\ (\tau, \emptyset)$ contains a constraint c which is not unifiable and has no narrowable occurrence. Thus this subgoal has just to be dropped from the list since it does not contain solutions.

□

Definition **5.7**. *(iCS transition relation)* $\overset{\tilde{c}}{\hookrightarrow}_{iCS}$

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

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 5.7, 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 **5.8**. *Let L_c represent an innermost search tree for the constraint c denoting the 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.*

PROOF.

- If c is empty, then $L_c = [\,]$. Since $merge(L_c, \tilde{c}) = merge([\,], \tilde{c}) = [\tilde{c}\ with\ (\epsilon, O'(\tilde{c}))]$ and by definition 5.3, $[\tilde{c}\ with\ (\epsilon, O'(\tilde{c}))]$ represents an innermost search tree for $\emptyset \cup \tilde{c}$.
- If c is not empty we have the following. The facts that every innermost narrowable occurrence of c is, likewise, an innermost narrowable occurrence of $c \cup \tilde{c}$ and that L_c represents an innermost search tree for c corresponding to the execution state when the first solution to c has been found, guarantee that $merge(L_c, \tilde{c})$ represents an innermost search tree for $c \cup \tilde{c}$ corresponding to the same state.

$\square$

Let us now illustrate Lemma 5.8 by an example.

EXAMPLE **4**. *Let us consider the equational theory $\mathcal{E}$ in Example 2. Figure 1 shows an innermost search tree for the constraint $c = \{U + Z = s(0)\}$ with respect to $\mathcal{E}$.*

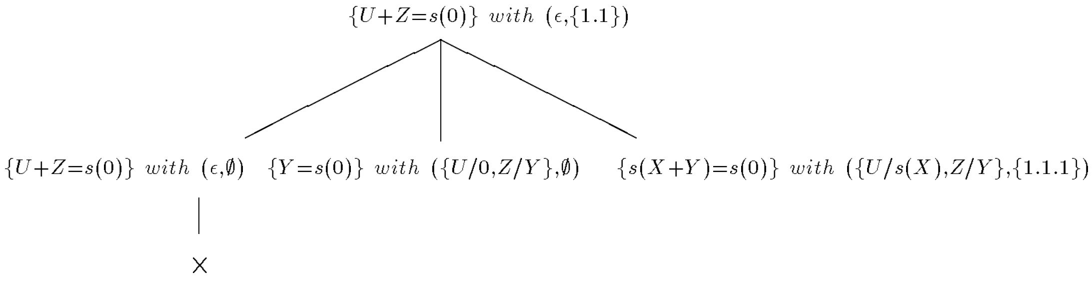

figure 1

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.

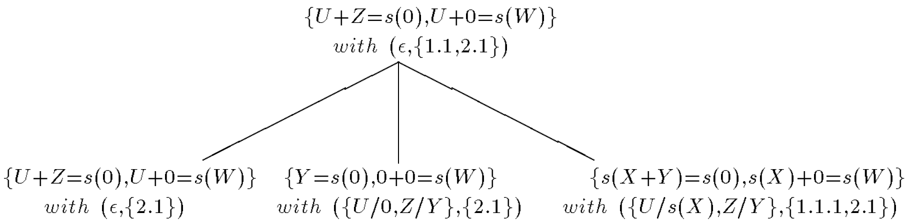

figure 2

The strategy defined by the calculus iCS can be proved complete by using the following result of completeness of the conditional selection narrowing in [49].

PROPOSITION **5.9**. *[Conjectured in [25] and proven in [49]]*
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 **5.10**. (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 $c \cup \tilde{c}$.

PROOF. Let $\langle c\,, L_c\rangle \overset{\tilde{c}}{\hookrightarrow}_{iCS} \langle c \cup \tilde{c}\,, L'\rangle$ be an iCS transition. Then by Definition 5.7 $L' = i\mathcal{N}(merge(L_c, \tilde{c}))$. By Lemma 5.8 $merge(L_c, \tilde{c})$ builds the representation of an innermost search tree for $c \cup \tilde{c}$, i.e. $\bar{L} = merge(L_c, \tilde{c}) = L_{c\cup\tilde{c}}$. Then by Lemma 5.6 $i\mathcal{N}(\bar{L}) = L'$ iff L' represents the innermost search tree for the contraint $c \cup \tilde{c}$ when a solution θ to it is found.

By proposition 5.9 the completeness follows, i.e. a solution θ to a contraint $c \cup \tilde{c}$ is found iff there is one.

□

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 3.2.

6 Conclusions

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 [26] where narrowing generates the solutions which are then tested by the logic program. One problem with the definition of CLP($\mathcal{H}/\mathcal{E}$) is that in general the narrower which solves the constraint may not terminate and thus the standard CLP($\mathcal{H}/\mathcal{E}$) reduction step would not work well in these cases. In [4] we have investigated some static techniques which allow to approximate the problem of the detection of the unsatisfiability of a set of equations with respect to a given equational theory, which allows to improve the termination of the basic CLP($\mathcal{H}/\mathcal{E}$) reduction step.

The CLP framework we have considered is the one originally introduced by Jaffar and Lassez [21, 36]. Our construction however could equivalently be defined in other CLP frameworks, such as the one proposed in [29].

The fixpoint and the model-theoretic semantics can be obtained from the results in [21, 36] in a straightforward way [2]. A simplified prototype of our language has been defined in [2].

7 Acknowledgement

We gratefully acknowledge Pierpaolo Degano and Maurizio Gabbrielli for many helpful discussions.

References

[1] H. Aït-Kaci and R. Nasr. A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215, 1986.

[2] M. Alpuente. *El lenguaje $CLP(\mathcal{H}/\mathcal{E})$: una aproximación basada en restricciones a la integración de la programación lógica y ecuacional*. PhD thesis, DSIC - Universidad Politécnica de Valencia, 1991.

[3] 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.

[4] 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.

[5] M. Alpuente and M.J. Ramírez. An Equational Constraint Logic Approach to Conceptual Modelling. In A.M. Tjoa and I. Ramos, editors, *Proc. 3rd Int'l Conf. on Database and Expert Systems Applications DEXA'92*, pages 26–31. Springer-Verlag, Wien New York, 1992.

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

[7] J.A. Bergstra and J.W. Klop. Conditional Rewrite Rules: confluence and termination. *Journal of Computer and System Sciences*, 32:323–362, 1986.

[8] H. Beringer and F. Porcher. A Relevant Scheme for Prolog Extensions: CLP(Conceptual Theory). In G. Levi and M. Martelli, editors, *Proc. Sixth Int'l Conf. on Logic Programming*, pages 131–148. The MIT Press, Cambridge, Mass., 1989.

[9] 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.

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

[11] 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.

[12] H.J. Burckert. Solving Disequations in Equational Theories. In E. Lusk and R. Overbeek, editors, *9th Int'l Conf. on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 517–526. Springer-Verlag, Berlin, 1988.

[13] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 243–320. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

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

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

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

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

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

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

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

[33] G. Huet and D.C. Oppen. Equations and Rewrite Rules: a Survey. In *Formal Languages: perspectives and open problems*, pages 349–405. Academic Press, 1980.

[34] 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.

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

[36] 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.

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

[38] J. Jaffar, J.-L. Lassez, and M. J. Maher. PROLOG-II an n instance of the logic programming language scheme. In M. Wirsing, editor, *Formal Descriptions of Programming Concepts*. North-Holland, 1986.

[39] 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.

[40] J. Jaffar, J.-L. Lassez, and M.J. Maher. Some issues and trends in the semantics of logic programming. In E. Y. Shapiro, editor, *Proc. Third Int'l Conf. on Logic Programming*, volume 225 of *Lecture Notes in Computer Science*, pages 223–241. Springer-Verlag, Berlin, 1986.

[41] 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.

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

[43] 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.

[44] J.W. Klop. Term Rewriting Systems. Technical Report CS-R9073, Centre for Mathematics and Computer Science, Amsterdam, 1991. to appear in "Handbook of Logic in Computer Science" Vol.1.

[45] 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.

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

[47] 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.

[48] 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.

[49] A. Middeldorp and E. Hamoen. Counterexamples to completeness results for basic narrowing. Technical report, Centre for Mathematics and Computer Science, Amsterdam, December 1991.

[50] 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.

[51] 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.

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

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

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

[55] 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.

[56] 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.

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

[58] 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.

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

[60] 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.