# Narrowing as an Incremental Constraint Satisfaction Algorithm

María Alpuente [†] Moreno Falaschi [‡]

## 1 Introduction

In the last few years several approaches to the integration of logic and equational programming have been developed [3,4,8,9,11,14,20]. One relevant approach [11] 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 [7,9,11]), 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* [23]). 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 [2], complete sets of transformations [12] and paramodulation [5] or some special form of it, such as superposition [4] or narrowing [2,13,14,21]. In [11], 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) [15,16]. CLP is a generic logic programming language scheme which extends pure logic programming to include constraints. Each instance CLP($\mathcal{X}$) of the scheme is a programming language that arises 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. The implementations of one instance of the CLP scheme can differ for the choice of the specific constraint solvers. The existence of a canonical domain of computation, least and greatest model semantics, the existence of a least and greatest fixpoint semantics and the soundness and completeness results for successful derivations are inherited by any extension which can be formalized as an instance of the scheme. Suitable models which correspond to different observable behaviours have been developed for the CLP scheme [6]. These results apply to each instance of the scheme.

[†]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.

In this paper we are concerned with an instance of the CLP scheme specialized in solving equations with respect to a Horn equational theory $\mathcal{E}$ [11]. The intended structure is just given by the finest partition induced by $\mathcal{E}$ on the Herbrand Universe $\mathcal{H}$ over a finite one sorted alphabet $\Sigma$. $=$ 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 [15,16]. 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 [16], 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. 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.

The paper is organized as follows. In Section 2 we briefly recall the basic concepts of the CLP framework, conditional rewrite 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. Finally Section 5 concludes.

All the proofs of the lemmata and theorems presented in this paper can be found in [1].

# 2 Preliminaries

In this paper we refer to a language which is an instance of CLP, as defined in [15]. We assume the reader to be familiar with logic programming [19], constraint logic programming [6,15,16], equations and conditional rewrite systems [18] and universal unification [23]. We first recall the basic concepts of the CLP framework. By $\Sigma, \Pi$ 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 $\Sigma$ and $V$, respectively. $\tau(\Sigma)$ is usually called the Herbrand Universe ($\mathcal{H}$) over $\Sigma$. A $(\Pi, \Sigma)$-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 $(\Pi, \Sigma)$-constraint is a possibly empty or infinite set of $(\Pi, \Sigma)$-atoms. Intuitively, a constraint is a conjunction of $(\Pi, \Sigma)$-atoms. The empty constraint will be denoted by *true*. The symbol $\tilde{}$ will denote a finite sequence of symbols. A structure $\Re(\Pi, \Sigma)$ over $\Pi$ and $\Sigma$ can be defined as in [15,16].