Narrowing Approximations as an Optimization for Equational Logic Programs *

María Alpuente[1], Moreno Falaschi[2], María José Ramis[1] and Germán Vidal[1]

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

[2] Dipartimento di Elettronica e Informatica,
Università di Padova,
Via Gradenigo 6/A, 35131 Padova, Italy.

Abstract. Solving equations in equational theories is a relevant programming paradigm which integrates logic and equational programming into one unified framework. Efficient methods based on narrowing strategies to solve systems of equations have been devised. In this paper, we formulate a narrowing-based equation solving calculus which makes use of a top-down abstract interpretation strategy to control the branching of the search tree. We define a refined, but still complete, equation solving procedure which allows us to reduce the branching factor. Our main idea consists of building an *abstract narrower* for equational theories and executing the set of equations to be solved in the approximated narrower. We define a generic technique of loop detection to ensure termination of our method. We prove that the set of answers computed by the abstract narrower has the property that each concrete solution of the set of equations is an instance of one of the substitutions in the answer set. Thus we define a strategy which computes such a set and uses the substitutions in it for cutting down the search space of the program without losing completeness. We also report on experimental results which demonstrate that our optimization can result in significant speed-ups in program execution.

Keywords: Abstract interpretation, equational logic programming, term rewriting systems, universal unification.

1 Introduction

The recent interest in logic programming with equations [12, 16, 18] has promoted much work on equational unification [14, 16, 27] and narrowing [16, 23]. Equational unification ($\mathcal{E}$-unification) characterizes the problem of solving equations *modulo* an equational theory $\mathcal{E}$. The narrowing mechanism is a powerful

* This work has been partially supported by CICYT under grant TIC 92-0793-C02-02. M.J. Ramis is supported by Spanish *Conselleria de Cultura, Educació i Ciència de la Generalitat Valenciana*.

tool for constructing complete $\mathcal{E}$-unification algorithms for useful classes of equational theories. In this context, completeness means that for every solution to a given set of equations, a more general solution can be found by narrowing. Since unrestricted narrowing has quite a large search space, several strategies to control the selection of redexes have been devised to improve the efficiency of narrowing by getting rid of some useless derivations [7, 16, 17, 23]. Narrowing at only *basic* positions has been proven to be a complete method for solving equations in the theory defined by a level-canonical conditional term rewriting system [15, 16, 17, 23, 24]. In [7, 15] a further refinement is considered which derives from simulating SLD-resolution on flattened equations and in which the search reduces to an innermost selection strategy.

The ability to detect the unsatisfiability of a set of equations is also very important to prune useless paths from the search tree and to save a lot of unnecessary computations. This problem is, in general, undecidable. [5] defines a scheme based on abstract interpretation for the (static) analysis of the unsatisfiability of equation sets, and shows how various analyses such as [2, 4, 12, 13] can be seen as instances of the scheme.

In this work, we are concerned with equation solving or $\mathcal{E}$-unification with respect to a given set $\mathcal{E}$ of equations. We define an optimization of (basic) conditional narrowing for the purpose of generating complete sets of $\mathcal{E}$-unifiers in equational theories that can be described by a (level-canonical) conditional term rewriting system. The optimization is based on the idea that narrowing derivations can be approximated by means of an abstract calculus whose derivations cover all concrete computations. [4, 5] define an abstract narrower which approximates the decision problem of the unsatisfiability of an equation set $\mathbf{E}$ with respect to an equational theory $\mathcal{E}$. The abstract narrower always terminates and, if it terminates computing a "finitely failed tree", it states that the set of equations has no $\mathcal{E}$-unifier. In this paper we show that this abstract narrower yields useful information also when it is unable to state that an equation set is unsatisfiable. In this case a finite description of the set of $\mathcal{E}$-unifiers of $\mathbf{E}$ can be collected. We show that this description can be used to filter those derivations that give rise to substitutions which are 'incompatible' with the description. Thus we present a calculus which on one side is able to detect unsatisfiability of equation sets, and if it fails to do so then it collects a finite description which allows us to prune the (concrete) narrowing search tree. We prove that the completeness result of basic conditional narrowing is preserved in the optimization. To the best of our knowledge this method is new and it allows an immediate combination with the existing interpreters of equational logic programs whose computational engine is narrowing [3, 7, 16]. In practice, the branches of the narrowing search tree whose root contains a substitution which is 'incompatible' with the set of collected abstract substitutions can be safely pruned. We show that this 'consistency check' can be performed compositionally. We define a composition of abstract substitutions which would allow a parallel implementation of the test. In the paper we show some experimental results for the case of a sequential implementation, we have not tried with a parallel interpreter yet.

The paper is organized as follows. After introducing some preliminary notions in Section 2, we present the basic solution procedure in Section 3. Section 4 recalls an abstract algorithm to analyze the unsatisfiability of equation sets [4, 5], and Section 5 defines a strategy that exploits the abstractions computed by this algorithm to reduce the alternatives that the solution procedure has to explore. Section 6 presents some experimental results for our method and concludes. More details and missing proofs can be found in [6].

2 Preliminaries

Let us first summarize some known results about equations, conditional rewrite systems and equational unification. For full definitions refer to [11, 19]. Throughout this paper, $\mathbf{V}$ will denote a countably infinite set of variables and Σ denotes a set of function symbols, each with a fixed associated arity. $\tau(\Sigma \cup \mathbf{V})$ and $\tau(\Sigma)$ denote the sets of terms and ground terms built on Σ and $\mathbf{V}$, respectively. A Σ-equation $\mathbf{s} = \mathbf{t}$ is a pair of terms $\mathbf{s}, \mathbf{t} \in \tau(\Sigma \cup \mathbf{V})$. Terms are viewed as labelled trees in the usual way. Occurrences are represented by sequences, possibly empty, of naturals. $\bar{\mathbf{O}}(\mathbf{t})$ denotes the set of nonvariable occurrences of a term $\mathbf{t}$. $\mathbf{t}_{|\mathbf{u}}$ is the subterm at the occurrence $\mathbf{u}$ of $\mathbf{t}$. $\mathbf{t}[\mathbf{r}]_{\mathbf{u}}$ is the term $\mathbf{t}$ with the subterm at the occurrence $\mathbf{u}$ replaced with $\mathbf{r}$. $\mathbf{t}[\mathbf{u}]$ denotes the label in $\mathbf{t}$ at occurrence $\mathbf{u} \in \bar{\mathbf{O}}(\mathbf{t})$. These notions extend to equations in a natural way. Identity of syntactic objects is denoted by $\equiv$. $\mathbf{Var}(\mathbf{s})$ is the set of distinct variables occurring in the syntactic object $\mathbf{s}$. A *fresh* variable is a variable that appears nowhere else. The symbol $\tilde{}$ denotes a finite sequence of symbols.

We describe the lattice of equation sets following [10]. We let **Eqn** denote the set of possibly existentially quantified finite sets of equations over terms. We let **fail** denote the unsatisfiable equation set, which (logically) implies all other equation sets. Likewise, the empty equation set, denoted **true**, is implied by all elements of **Eqn**. We write $\mathbf{E} \leq \mathbf{E}'$ if $\mathbf{E}'$ logically implies $\mathbf{E}$. Thus **Eqn** is a lattice ordered by $\leq$ with bottom element **true** and top element **fail**. An equation set is *solved* if it is either **fail** or it has the form $\exists \mathbf{y}_1 \ldots \exists \mathbf{y}_\mathbf{m} . \{\mathbf{x}_1 = \mathbf{t}_1, \ldots, \mathbf{x}_\mathbf{n} = \mathbf{t}_\mathbf{n}\}$ where each $\mathbf{x}_\mathbf{i}$ is a distinct variable not occurring in any of the terms $\mathbf{t}_\mathbf{i}$ and each $\mathbf{y}_\mathbf{i}$ occurs in some $\mathbf{t}_\mathbf{j}$. Any set of equations $\mathbf{E}$ can be transformed into an equivalent one $\mathbf{solve}(\mathbf{E})$ which is solved. We restrict our interest to the set of idempotent substitutions over $\tau(\Sigma \cup \mathbf{V})$, which is denoted by **Sub**. There is a natural isomorphism between substitutions and unquantified equation sets.

We use the same notation for a substitution $\{\mathbf{x}_1/\mathbf{t}_1, \ldots, \mathbf{x}_\mathbf{n}/\mathbf{t}_\mathbf{n}\}$, the corresponding set of equations $\{\mathbf{x}_1 = \mathbf{t}_1, \ldots, \mathbf{x}_\mathbf{n} = \mathbf{t}_\mathbf{n}\}$ and the corresponding conjunction of equations $\mathbf{x}_1 = \mathbf{t}_1 \wedge \ldots \wedge \mathbf{x}_\mathbf{n} = \mathbf{t}_\mathbf{n}$. For example we may write $\theta \wedge \mathbf{x}\theta$, where the first occurrence of θ is a conjunction of equations and the second occurrence is the application of a substitution.

We consider the usual preorder on substitutions $\leq$: $\theta \leq \sigma$ iff $\exists \gamma . \sigma \equiv \theta\gamma$. Note that $\theta \leq \sigma$ iff $\sigma \Rightarrow \theta$ [25]. A substitution $\{\mathbf{x}_1/\mathbf{t}_1, \ldots, \mathbf{x}_\mathbf{n}/\mathbf{t}_\mathbf{n}\}$ is a *unifier* of an equation set $\mathbf{E}$ iff $\{\mathbf{x}_1 = \mathbf{t}_1, \ldots, \mathbf{x}_\mathbf{n} = \mathbf{t}_\mathbf{n}\} \Rightarrow \mathbf{E}$. We denote the set of unifiers of $\mathbf{E}$ by $\mathbf{unif}(\mathbf{E})$ and $\mathbf{mgu}(\mathbf{E})$ denotes the *most general unifier* of the unquantified

equation set $\mathbf{E}$. While every unquantified equation set has a *most general unifier* [20], this is not true in general for equation sets with existentially quantified variables. We write $\mathbf{s} \stackrel{?}{=} \mathbf{t}$ when we want to point out the fact that two terms $\mathbf{s}$ and $\mathbf{t}$ do unify.

We note that quantifiers will not be used when dealing with the concrete operational semantics. It is only when arguing the relation between the concrete and abstract operational semantics that we need to consider quantified equation sets, since our abstract algorithm replaces some of the terms in the term rewriting system by occurrences of a special symbol which from the logical viewpoint stands for a quantified variable.

A Horn equational Σ-theory $\mathcal{E}$ consists of a finite set of equational Horn clauses of the form $\mathbf{e} \Leftarrow \mathbf{e_1}, \ldots, \mathbf{e_n}, \ \mathbf{n} \geq 0$, where $\mathbf{e}, \mathbf{e_i}, \ \mathbf{i} = 1, \ldots, \mathbf{n}$, are Σ-equations. Σ-equations and Σ-theories will often be called equations and theories, respectively. An equational goal is an equational Horn clause with no head.

A Term Rewriting System (TRS for short) is a pair $(\Sigma, \mathcal{R})$ where $\mathcal{R}$ is a finite set of reduction (or rewrite) rule schemes of the form $(\lambda \rightarrow \rho \Leftarrow \widetilde{\mathbf{e}})$, λ, $\rho \in \tau(\Sigma \cup \mathbf{V})$, $\lambda \notin \mathbf{V}$ and $\mathbf{Var}(\rho) \subseteq \mathbf{Var}(\lambda)$. The condition $\widetilde{\mathbf{e}}$ is a possibly empty conjunction $\mathbf{e_1}, \ldots, \mathbf{e_n}, \ \mathbf{n} \geq 0$, of equations. We will often write just $\mathcal{R}$ instead of $(\Sigma, \mathcal{R})$.

A Horn equational theory $\mathcal{E}$ which satisfies the above assumptions can be viewed as a term rewriting system $\mathcal{R}$ where the rules are the heads (implicitely oriented from left to right) and the conditions are the respective bodies. We assume that these assumptions hold for all theories we consider in this paper. 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 [19].

A function symbol $\mathbf{f} \in \Sigma$ is irreducible iff there is no rule $(\lambda \rightarrow \rho \Leftarrow \widetilde{\mathbf{e}}) \in \mathcal{R}$ such that $\mathbf{f}$ occurs as the outermost function symbol in λ, otherwise it is a defined function symbol. In theories where the above distinction is made, the signature Σ is partitioned as $\Sigma = \mathbf{C} \uplus \mathbf{F}$, where $\mathbf{C}$ is the set of irreducible function symbols and $\mathbf{F}$ is the set of defined function symbols.

For TRS $\mathcal{R}$, $\mathbf{r} \ll \mathcal{R}$ denotes that $\mathbf{r}$ is a new variant of a rule in $\mathcal{R}$ such that $\mathbf{r}$ contains no variable previously met during computation (standardised apart). Given a conditional TRS $\mathcal{R}$, an equational goal clause $\mathbf{G}$ conditionally narrows into a goal clause $\mathbf{G'}$ if there exists an equation $\mathbf{e} \in \mathbf{G}$, $\mathbf{u} \in \bar{\mathbf{O}}(\mathbf{e})$, a standardised apart variant $(\lambda \rightarrow \rho \Leftarrow \widetilde{\mathbf{e}}) \ll \mathcal{R}$ and a substitution σ such that $\sigma = \mathbf{mgu}(\{\mathbf{e}_{|\mathbf{u}} = \lambda\})$ and $\mathbf{G'} = ((\mathbf{G} \sim \{\mathbf{e}\}) \cup \{\mathbf{e}[\rho]_{\mathbf{u}}\} \cup \widetilde{\mathbf{e}})\sigma$. $\mathbf{s}$ is called a (narrowing) *redex* (*red*ucible *ex*pression) iff there exists a new variant $(\lambda \rightarrow \rho \Leftarrow \widetilde{\mathbf{e}})$ of a reduction rule in $\mathcal{R}$ and a substitution σ such that $\mathbf{s}\sigma \equiv \lambda\sigma$.

Given a set of equations $\mathbf{E}$, we say that $\mathbf{E}$ is **$\mathcal{E}$-unifiable** iff there exists a substitution σ such that $\mathcal{E} \models \mathbf{E}\sigma$ [22]. The substitution σ is called an **$\mathcal{E}$-unifier** of $\mathbf{E}$. The set of all **$\mathcal{E}$-unifiers** of $\mathbf{E}$ is recursively enumerable [14, 16, 27]. Conditional narrowing has been shown to be a complete $\mathcal{E}$-unification algorithm for equational theories satisfying different restrictions [16, 23].

3 Basic Conditional Narrowing

Basic (conditional) narrowing is a restricted form of (conditional) narrowing where only terms at *basic* occurrences are considered to be narrowed [17, 23]. Informally, a basic occurrence is a nonvariable occurrence of the original goal or one that was introduced into the goal by the nonvariable part of the right-hand side or the condition of a rule applied in a preceeding narrowing step. The idea behind the concept of *basic* is to avoid narrowing steps on subterms that are introduced by instantiation. Basic Conditional Narrowing is a complete $\mathcal{E}$-**unification** algorithm for level-canonical Horn equational theories [23].

Let $\mathcal{R}$ be a level-canonical TRS. We formulate a Basic Conditional Narrowing calculus according to the partition of equational goals into a *skeleton* and an *environment* part, as in [16]. The *skeleton* part is a set of equations $\mathbf{g}$ and the *environment* part is a substitution θ. Substitutions are composed in the environment part, but are not applied to the terms in the skeleton part. Due to this representation, the basic occurrences in $\mathbf{g}\theta$ are all in $\mathbf{g}$, whereas the non-basic occurrences are all in the codomain of θ. We let **State** denote the set of goals (**states**). The calculus is defined as a transition system, which is a directed graph that has states as nodes. The initial state is a 'source' node and edges correspond to reductions between the states. Thus reduction sequences correspond to paths in the graph starting from the source. To solve the equation set $\mathbf{g}$, the algorithm starts with the initial state $\langle \mathbf{g}, \epsilon \rangle$ and tries to derive new goals until a terminal goal $\langle \mathbf{true}, \sigma \rangle$ is reached. Each substitution σ in a terminal goal is an $\mathcal{E}$-**unifier** of $\mathbf{g}$. By abuse of notation, it is often called solution. A *successful* derivation is a reduction sequence which ends in a terminal goal. A *basic Conditional Narrowing* calculus ($\mathbf{b}\mathcal{CN}$) is defined by the two following rules:

unification rule:

$$\frac{\sigma = \mathbf{mgu}(\mathbf{g}\theta)}{\langle \mathbf{g}, \theta \rangle \quad \leadsto_{\mathbf{b}\mathcal{CN}} \quad \langle \mathbf{true}, \theta\sigma \rangle}$$

narrowing rule:

$$\frac{\mathbf{e} \in \mathbf{g} \quad \wedge \quad \mathbf{u} \in \bar{\mathbf{O}}(\mathbf{e}) \quad \wedge \quad (\lambda \to \rho \Leftarrow \widetilde{\mathbf{e}}) \ll \mathcal{R} \quad \wedge \quad \sigma = \mathbf{mgu}(\{(\mathbf{e}_{|\mathbf{u}})\theta = \lambda\})}{\langle \mathbf{g}, \theta \rangle \quad \leadsto_{\mathbf{b}\mathcal{CN}} \quad \langle (\mathbf{g} \sim \{\mathbf{e}\}) \cup \{\mathbf{e}[\rho]_{\mathbf{u}}\} \cup \widetilde{\mathbf{e}}, \theta\sigma \rangle}$$

4 Abstract Basic Conditional Narrowing

Abstract interpretation is a theory of semantic approximations which is used to provide statically sound answers to some questions about the run-time behaviour of programs. In this section, we formulate in terms of abstract interpretation an algorithm that approximates the problem of the $\mathcal{E}$-unification of a given set of equations. The main idea behind our method is to abstract the transition system semantics $\mathbf{b}\mathcal{CN}$ we introduced in Section 3. For this purpose, we follow the top-down approach to abstract interpretation which is based on constructing and examining abstract transition systems as defined in [10].

Most of the definitions at the beginning of this section are already in [4, 5] and are reported for completeness. A difference with respect to [4] is in the definition of abstract most general unifier (Definition 5) which has been revised and simplified. Another difference is that we present Definition 10 as parametric with respect to a functional dependency graph (or loop-check). The corresponding definition in [5] is an instance for the specific dependency graph considered there.

A *description* is the association of an *abstract domain* $(\mathbf{D}, \leq)$ (a poset) with a *concrete domain* $(\mathbf{E}, \leq)$ (a poset). When $\mathbf{E} = \mathbf{Eqn}$, $\mathbf{E} = \mathbf{Sub}$ or $\mathbf{E} = \mathbf{State}$, the description is called an *equation description*, a *substitution description* or a *state description*, respectively. The correspondence between the abstract and concrete domain is established through a 'concretization' function $\gamma : \mathbf{D} \to \wp\mathbf{E}$. We say that $\mathbf{d}$ *approximates* $\mathbf{e}$, written $\mathbf{d} \propto \mathbf{e}$, iff $\mathbf{e} \in \gamma(\mathbf{d})$. The approximation relation can be lifted to relations and cross products as usual [5].

In the following, we approximate the behaviour of a TRS and initial state by an abstract transition system which can be viewed as a finite transition graph with nodes labeled by state descriptions, where transitions are proved by (abstract) narrowing reduction [4, 5]. State descriptions consist of a set of equations with substitution descriptions. The descriptions for equations, substitutions and term rewriting systems are defined as follows:

Definition 1. By $\mathcal{T} = (\tau(\Sigma \cup \mathbf{V}), \leq)$ we denote the standard domain of (equivalence classes of) terms ordered by the standard partial order $\leq$ induced by the preorder on terms given by the relation of being "more general". Let $\perp$ be an irreducible symbol, where $\perp \notin \Sigma$. Roughly speaking, the special symbol $\perp$ introduced in the abstract domains represents any concrete term. Let $\mathcal{T}^{\perp} = (\tau(\Sigma \cup \mathbf{V} \cup \{\perp\}), \preceq)$ be the domain of terms over the signature augmented by $\perp$, where the partial order $\preceq$ is defined as follows:

(a) $\forall \mathbf{t} \in \mathcal{T}^{\perp}. \perp \preceq \mathbf{t}$ and $\mathbf{t} \preceq \mathbf{t}$ and

(b) $\forall \mathbf{s_1}, \ldots, \mathbf{s_n}, \mathbf{s'_1}, \ldots, \mathbf{s'_n} \in \mathcal{T}^{\perp}, \forall \mathbf{f/n} \in \Sigma.\ \mathbf{s'_1} \preceq \mathbf{s_1} \wedge \ldots \wedge \mathbf{s'_n} \preceq \mathbf{s_n} \Rightarrow \mathbf{f}(\mathbf{s'_1}, \ldots, \mathbf{s'_n}) \preceq \mathbf{f}(\mathbf{s_1}, \ldots, \mathbf{s_n})$.

This order can be extended to equations: $\mathbf{s'} = \mathbf{t'} \preceq \mathbf{s} = \mathbf{t}$ iff $\mathbf{s'} \preceq \mathbf{s}$ and $\mathbf{t'} \preceq \mathbf{t}$, and to sets of equations $\mathbf{S}, \mathbf{S'}$:

1) $\mathbf{S'} \preceq \mathbf{S}$ iff $\forall \mathbf{e'} \in \mathbf{S'}. \exists \mathbf{e} \in \mathbf{S}$ such that $\mathbf{e'} \preceq \mathbf{e}$. Note that $\mathbf{S'} \preceq \mathbf{true} \Rightarrow \mathbf{S'} \equiv \mathbf{true}$.

2) $\mathbf{S'} \sqsubseteq \mathbf{S}$ iff $(\mathbf{S'} \preceq \mathbf{S})$ and $(\mathbf{S} \preceq \mathbf{S'}$ implies $\mathbf{S'} \subseteq \mathbf{S})$.

The behaviour of the symbol $\perp$ from a programming viewpoint resembles that of an "anonymous" variable in Prolog. From a logical viewpoint, $\perp$ stands for an existentially quantified variable [5, 21, 22]. Define $[\![\mathbf{S}]\!] = \mathbf{S'}$, where the n-tuple of occurrences of $\perp$ in $\mathbf{S}$ is replaced by an n-tuple of existentially quantified fresh variables in $\mathbf{S'}$. We note that $\mathbf{S'} \preceq \mathbf{S}$ implies $[\![\mathbf{S}]\!] \Rightarrow [\![\mathbf{S'}]\!]$ [5].

Definition 2. An abstract substitution is a set of the form $\{\mathbf{x_1/t_1}, \ldots, \mathbf{x_n/t_n}\}$ where, for each $\mathbf{i} = 1, \ldots, \mathbf{n}$, $\mathbf{x_i}$ is a distinct variable in $\mathbf{V}$ not occurring in any of the terms $\mathbf{t_1}, \ldots, \mathbf{t_n}$ and $\mathbf{t_i} \in \tau(\Sigma \cup \mathbf{V} \cup \{\perp\})$.

We can now characterize the ordering on abstract substitutions as logical implication of the corresponding logical expressions: let $\theta, \kappa \in \mathbf{Sub}^{\perp}$, $\kappa \preceq \theta$ iff $[\![\theta]\!] \Rightarrow [\![\kappa]\!]$.

We can now introduce the abstract domains which we will use in our analysis.

Definition 3. *upward*
Let $(\mathbf{X}, \leq)$ be a poset and let $\mathbf{Y} \subseteq \mathbf{X}$. Define $\mathbf{upw}(\mathbf{Y}) = \{\mathbf{x} \in \mathbf{X} | \exists \mathbf{y} \in \mathbf{Y}.\mathbf{y} \leq \mathbf{x}\}$.

Definition 4. Let $\mathcal{T} = (\tau(\Sigma \cup \mathbf{V}), \leq)$ and $\mathcal{T}^{\perp} = (\tau(\Sigma \cup \mathbf{V} \cup \{\perp\}), \preceq)$. The *term description* is $\langle \mathcal{T}^{\perp}, \gamma, \mathcal{T} \rangle$ where $\gamma : \mathcal{T}^{\perp} \to \wp\mathcal{T}$ is defined by: $\gamma(\mathbf{t}') = \{\mathbf{t} \in \mathcal{T} \mid \mathbf{t} \in \mathbf{upw}(\{\mathbf{t}'\})\}$.

Let $\mathbf{Eqn}$ be the set of finite sets of equations over $\tau(\Sigma \cup \mathbf{V})$ and $\mathbf{Eqn}^{\perp}$ be the set of finite sets of equations over $\tau(\Sigma \cup \mathbf{V} \cup \{\perp\})$. The *equation description* is $\langle (\mathbf{Eqn}^{\perp}, \sqsubseteq), \gamma, (\mathbf{Eqn}, \leq) \rangle$, where $\gamma : \mathbf{Eqn}^{\perp} \to \wp\mathbf{Eqn}$ is defined by: $\gamma(\mathbf{g}') = \{\mathbf{g} \in \mathbf{Eqn} \mid \mathbf{g} \in \mathbf{upw}(\{\mathbf{g}'\})$ and $\mathbf{g}$ is unquantified $\}$.

Let $\mathbf{Sub}$ be the set of substitutions over $\tau(\Sigma \cup \mathbf{V})$ and $\mathbf{Sub}^{\perp}$ be the set of substitutions over $\tau(\Sigma \cup \mathbf{V} \cup \{\perp\})$. The *substitution description* is $\langle (\mathbf{Sub}^{\perp}, \preceq), \gamma, (\mathbf{Sub}, \leq) \rangle$, where $\gamma : \mathbf{Sub}^{\perp} \to \wp\mathbf{Sub}$ is defined by: $\gamma(\kappa) = \{\theta \in \mathbf{Sub} \mid \theta \in \mathbf{upw}(\{\kappa\})\}$.

Define the abstract state domain $\mathbf{State}^{\perp}$ induced by $\mathbf{Eqn}^{\perp}$ and $\mathbf{Sub}^{\perp}$ to be $\mathbf{State}^{\perp} = \{\langle \mathbf{g}, \kappa \rangle \mid \mathbf{g} \in \mathbf{Eqn}^{\perp}, \kappa \in \mathbf{Sub}^{\perp}\}$.

Abstract states are intended to describe states which are equivalent modulo variable renaming. To ensure finiteness of the analysis, we collapse states which are variable renamings of each other into a single 'equivalent' state, as in [10]. A formal description can be found in [4, 5] and is omitted here for brevity.

In the following, we formalize the idea that abstract narrowing reduction approximates narrowing reduction with abstract states, abstract unification and abstract term rewriting systems replacing concrete states, unification and term rewriting systems. The abstract most general unifier for our method is very simple and roughly speaking it boils down to computing a solved form of an equation set with (possibly) existentially quantified variables.

We define the abstract most general unifier for an equation set $\mathbf{E}' \in \mathbf{Eqn}^{\perp}$ as follows. First replace all occurrences of $\perp$ in $\mathbf{E}'$ by existentially quantified fresh variables. Then take a solved form of the resulting quantified equation set and finally replace the existentially quantified variables again by $\perp$. Formally:

Definition 5. *Abstract most general unifier*
Let $\mathbf{solve}([\![\mathbf{E}']\!]) = \exists \mathbf{y_1} \ldots \mathbf{y_n}.\ \mathbf{E}$ and $\kappa = \{\mathbf{y}_1/\perp, \ldots, \mathbf{y_n}/\perp\}$. Then,
$\mathbf{mgu}_{\mathcal{A}}(\mathbf{E}') = \mathbf{E}\kappa$.

Example 1. Let $\mathbf{E} = \{\mathbf{f}(\mathbf{g}(\perp), \mathbf{h}(\mathbf{z}, \mathbf{z})) = \mathbf{f}(\mathbf{x}, \mathbf{h}(\mathbf{x}, \mathbf{g}(\mathbf{a}))),\ \mathbf{y} = \mathbf{g}(\perp)\}$. Then,
$\mathbf{mgu}_{\mathcal{A}}(\mathbf{E}) = \{\mathbf{x}/\mathbf{g}(\mathbf{a}), \mathbf{y}/\mathbf{g}(\perp), \mathbf{z}/\mathbf{g}(\mathbf{a})\}$.

The following proposition justifies our use of 'most general'.

Proposition 6. [5] $\forall \theta \in \mathbf{unif}([\![\mathbf{E}]\!])$. $\mathbf{mgu}_{\mathcal{A}}(\mathbf{E}) \preceq \theta$.

The safety of the abstract unification algorithm has been proven in [5] as stated by the following proposition.

Proposition 7. [5] *Let* $\mathbf{g}$, $\mathbf{g}'$ *be (quantifier free) finite sets of equations over* $\tau(\Sigma \cup \mathbf{V})$ *and* $\tau(\Sigma \cup \mathbf{V} \cup \{\bot\})$, *respectively. If* $\mathbf{g}' \propto \mathbf{g}$ *and there exists* $\mathbf{mgu}(\mathbf{g}) = \sigma$, *then there exists* $\mathbf{mgu}_{\mathcal{A}}(\mathbf{g}') = \sigma_{\mathcal{A}}$ *where* $\sigma_{\mathcal{A}} \propto \sigma$.

In general, the use of narrowing results in a semidecision procedure for testing the solvability of equation sets. A few approaches to improve the termination of narrowing have been reported [1, 8, 9, 17, 26]. For example, [8, 26] consider a graph of terms which allows the detection of some loops in the search tree which do not lead to any solution. The improved algorithms described in [8, 26] are still complete but termination is only guaranteed for theories which satisfy some rather strong conditions. The graphs are built using information about the equations being narrowed as well as the reduction rules being used for narrowing. In the following we propose a generic technique of loop detection which is used to obtain a form of 'compiled' program which always satisfies a simple termination condition.

Our notion of abstract term rewriting system is parametric with respect to a loop-check.

Definition 8. *loop-check*
A loop-check is a graph $\mathcal{G}_{\mathcal{R}}$ associated with a term rewriting system $\mathcal{R}$, i.e. a relation consisting of a set of pairs of terms, such that: (1) the transitive closure $\mathcal{G}^+_{\mathcal{R}}$ is decidable and (2) Let $\widehat{\mathbf{t}} = \mathbf{t}'$ be a function which assigns to any term $\mathbf{t}$ some node $\mathbf{t}'$ in $\mathcal{G}_{\mathcal{R}}$. If there is an infinite sequence:

$\langle \mathbf{g}_0, \theta_0 \rangle \leadsto_{\mathbf{bCN}} \langle \mathbf{g}_1, \theta_1 \rangle \leadsto_{\mathbf{bCN}} \ldots$

then

$$\exists \mathbf{i} \geq 0.\ \langle \widehat{\mathbf{t_i}}, \widehat{\mathbf{t_i}} \rangle \in \mathcal{G}^+_{\mathcal{R}},\ \text{where } \mathbf{t_i} = \mathbf{e}_{|\mathbf{u}}\theta_{\mathbf{i}},\ \mathbf{e} \in \mathbf{g_i} \text{ and } \mathbf{u} \in \bar{\mathbf{O}}(\mathbf{e}).$$

(we refer to $\langle \widehat{\mathbf{t_i}}, \widehat{\mathbf{t_i}} \rangle$ as a 'cycle' of $\mathcal{G}^+_{\mathcal{R}}$.)

Any node of the graph represents one or more variable renamings of itself, to avoid variable names clash. We refer to $\mathcal{G}_{\mathcal{R}}$ as "the graph of functional dependencies". The graph can be useful to prove the termination of basic narrowing derivations for simple cases as stated by the following proposition.

Proposition 9. *Let* $\mathcal{R}$ *be a term rewriting system and* $\mathcal{G}_{\mathcal{R}}$ *be the corresponding graph of functional dependencies. If there is no cycle in* $\mathcal{G}_{\mathcal{R}}$, *then every basic narrowing derivation terminates.*

A TRS is abstracted by simplifying the right-hand side and the body of each rule. This definition is given inductively on the structure of terms and equations. Terms whose main functor $\mathbf{f}$ is a defined symbol are drastically simplified by replacing them by $\bot$. Notice that we do not perform this approximation when there is no cycle in $\mathcal{G}_{\mathcal{R}}$ for the term which we are considering. In this case, we can be more accurate and retain the subterm originating in $\mathbf{f}$.

Definition 10. *Abstract term rewriting system*
Let $\mathcal{R}$ be a TRS. Let $\mathcal{G}_{\mathcal{R}}$ be the corresponding graph of functional dependencies. We define the abstraction of $\mathcal{R}$ as follows:

$\mathcal{R}_{\mathcal{A}} = \{\lambda \to \mathbf{sh}(\rho) \Leftarrow \mathbf{sh}(\widetilde{\mathbf{e}}) \mid \lambda \to \rho \Leftarrow \widetilde{\mathbf{e}} \in \mathcal{R}\}$, where the shell $\mathbf{sh}(\mathbf{x})$ of an expression $\mathbf{x}$ is defined inductively

$$\mathbf{sh}(\mathbf{x}) = \begin{cases} \mathbf{x} & \text{if } \mathbf{x} \in \mathbf{V} \\ \mathbf{f}(\mathbf{sh}(\mathbf{t}_1), \ldots, \mathbf{sh}(\mathbf{t_k})) & \text{if } \mathbf{x} = \mathbf{f}(\mathbf{t}_1, \ldots, \mathbf{t_k}) \text{ and } \langle \widehat{\mathbf{x}}, \widehat{\mathbf{x}} \rangle \notin \mathcal{G}_{\mathcal{R}}^{+} \\ \mathbf{sh}(\mathbf{l}) = \mathbf{sh}(\mathbf{r}) & \text{if } \mathbf{x} = (\mathbf{l} = \mathbf{r}) \\ \mathbf{sh}(\mathbf{e}_1), \ldots, \mathbf{sh}(\mathbf{e_n}) & \text{if } \mathbf{x} = \mathbf{e}_1, \ldots, \mathbf{e_n} \\ \bot & \text{otherwise} \end{cases}$$

Note that a corresponding concretization function for abstract term rewriting systems can be easily defined. We can now define abstract basic narrowing.

Definition 11. Let $\mathcal{R}_{\mathcal{A}}$ be an abstract **TRS**. *Abstract (basic) narrowing* is a relation $\mathcal{B}^{\mathcal{R}_{\mathcal{A}}} \subseteq \mathbf{State}^{\bot} \times \mathbf{State}^{\bot}$, defined as follows.

Let $\mathbf{t}, \mathbf{t}' \in \mathbf{State}^{\bot}$. $\langle \mathbf{t}, \mathbf{t}' \rangle \in \mathcal{B}^{\mathcal{R}_{\mathcal{A}}}$ (we also say that there is an *abstract (basic) narrowing* reduction from $\mathbf{t}$ to $\mathbf{t}'$) iff:

1) $\exists \lambda \to \rho \Leftarrow \widetilde{\mathbf{e}} \ll \mathcal{R}_{\mathcal{A}}$ such that $\mathbf{t} = \langle \mathbf{g}, \kappa \rangle$ and $\mathbf{t}' = \langle \mathbf{g}', \kappa' \rangle$ where $\mathbf{g}' = (\mathbf{g} \sim \{\mathbf{e}\}) \cup \{\mathbf{e}[\rho]_{\mathbf{u}}\} \cup \widetilde{\mathbf{e}}$, $\mathbf{e} \in \mathbf{g}$, $\mathbf{u} \in \bar{\mathbf{O}}(\mathbf{e})$, $\sigma = \mathbf{mgu}_{\mathcal{A}}(\{(\mathbf{e}_{|\mathbf{u}})\kappa = \lambda\})$ and $\kappa' = \kappa\sigma$; or,
2) $\mathbf{t} = \langle \mathbf{g}, \kappa \rangle$, $\sigma = \mathbf{mgu}_{\mathcal{A}}(\mathbf{g}\kappa)$ and $\mathbf{t}' = \langle \mathbf{true}, \kappa\sigma \rangle$; or,
3) $\mathbf{t} = \mathbf{t}'$.

The following lemma is proven in [5] and basically states that abstract (basic) narrowing reduction approximates (basic) narrowing reduction.

Lemma 12. [5] *Let $\mathbf{s}' \propto \mathbf{s}$. If there is a (basic) narrowing reduction from $\mathbf{s}$ to $\mathbf{t}$, then there is an abstract (basic) narrowing reduction from $\mathbf{s}'$ to some $\mathbf{t}'$ such that $\mathbf{t}' \propto \mathbf{t}$.*

The following proposition shows that our analysis terminates.

Proposition 13. [5] *Let $\mathcal{R}_{\mathcal{A}}$ be an abstract TRS. The (abstract) transition system for $\mathcal{B}^{\mathcal{R}_{\mathcal{A}}}$, state $\mathbf{s}$ and $\mathbf{State}^{\bot}$ has a finite number of (distinct) nodes.*

The following auxiliary notions will be used below.

Definition 14. An abstract (basic) narrowing derivation for $\mathcal{R}_{\mathcal{A}}$, $\mathbf{s}$ and $\mathbf{State}^{\bot}$ is *relevant* if the last node in the derivation is labeled by $\langle \mathbf{true}, \kappa \rangle$.

Definition 15. Let $\mathcal{R}_{\mathcal{A}}$ be an abstract TRS. Define $\Delta(\mathcal{R}_{\mathcal{A}}, \mathbf{s})$ to be the set of abstract substitutions κ in the terminal states $\langle \mathbf{true}, \kappa \rangle$ of the abstract (basic) narrowing derivations which are relevant for $\mathcal{R}_{\mathcal{A}}$, $\mathbf{s}$ and $\mathbf{State}^{\bot}$.

$\Delta(\mathcal{R}_\mathcal{A}, \mathbf{s})$ is the set of the 'computed answer abstract substitutions' for $\mathbf{s}$. At the end of this section we show that $\Delta(\mathcal{R}_\mathcal{A}, \mathbf{s})$ allows us to detect branches of the concrete narrowing search tree which can be safely pruned. We give an example which can help to understand these definitions and motivates the remainder of the section. Let us introduce the auxiliary function $\lfloor \mathbf{x} \rfloor$, which inductively replaces by a fresh variable any term whose outermost symbol is not an irreducible function symbol, i.e.

$$\lfloor \mathbf{x} \rfloor = \begin{cases} \mathbf{c}(\lfloor \mathbf{t_1} \rfloor, \ldots, \lfloor \mathbf{t_k} \rfloor) & \text{if } \mathbf{x} = \mathbf{c}(\mathbf{t_1}, \ldots, \mathbf{t_k}) \text{ and } \mathbf{c} \in \mathbf{C} \\ \mathbf{y} & \text{otherwise, where } \mathbf{y} \text{ is a new fresh variable} \end{cases}$$

The following definition introduces a particular case of graph that satisfies the condition of loop-check of Definition 8. Another different instance can be found in [4, 5].

Definition 16. Let $\mathcal{R}$ be a TRS. The following transformation defines a directed graph $\mathcal{G}_\mathcal{R}$ of functional dependencies induced by $\mathcal{R}$. Define $\widehat{\mathbf{t}} = \mathbf{f}(\lfloor \mathbf{t_1} \rfloor, \ldots, \lfloor \mathbf{t_n} \rfloor)$ if $\mathbf{t} = \mathbf{f}(\mathbf{t_1}, \ldots, \mathbf{t_n})$. To build $\mathcal{G}_\mathcal{R}$, the algorithm starts with $\langle \mathcal{R}, \emptyset \rangle$ and applies the inference rules as long as they add new arrows. The symbol $\cup$ stands for set union.

$$(1)\ \frac{\mathbf{r} = (\lambda \to \rho \Leftarrow \widetilde{\mathbf{e}}) \ll \mathcal{R}}{\langle \mathcal{R}, \mathcal{G}_\mathcal{R} \rangle \quad \longmapsto \quad \langle \mathcal{R} \sim \{\mathbf{r}\}, \mathcal{G}_\mathcal{R} \cup \{\lambda \xrightarrow{\mathcal{R}} \widehat{\mathbf{t}} \mid (\mathbf{t} = \rho_{|\mathbf{u}} \wedge \mathbf{u} \in \bar{\mathbf{O}}(\rho)) \text{ or } (\mathbf{t} = \mathbf{e}_{|\mathbf{u}} \wedge \mathbf{e} \in \widetilde{\mathbf{e}} \wedge \mathbf{u} \in (\bar{\mathbf{O}}(\mathbf{e}) \sim \{\Lambda\}))\} \rangle}$$

$$(2)\ \frac{(\lambda \xrightarrow{\mathcal{R}} \mathbf{r}) \in \mathcal{G}_\mathcal{R} \ \wedge \ (\lambda' \xrightarrow{\mathcal{R}} \mathbf{r}') \in \mathcal{G}_\mathcal{R} \ \wedge \ \mathbf{r} \stackrel{?}{=} \lambda'}{\langle \mathcal{R}, \mathcal{G}_\mathcal{R} \rangle \quad \longmapsto \quad \langle \mathcal{R}, \mathcal{G}_\mathcal{R} \cup \{\mathbf{r} \xrightarrow{\mathbf{u}} \lambda'\} \rangle}$$

Termination of this calculus is ensured since the number of terms occurring in the rules in $\mathcal{R}$ is finite. Roughly speaking, in Definition 16, for each rule $(\lambda \to \rho \Leftarrow \widetilde{\mathbf{e}})$ in $\mathcal{R}$ and for each term $\mathbf{f}(\mathbf{t_1}, \ldots, \mathbf{t_n})$ occurring in ρ or in $\widetilde{\mathbf{e}}$, rule (1) adds an arrow $\lambda \xrightarrow{\mathcal{R}} \mathbf{f}(\lfloor \mathbf{t_1} \rfloor, \ldots, \lfloor \mathbf{t_n} \rfloor)$ to $\mathcal{G}_\mathcal{R}$. Rule (2) adds an arrow $\mathbf{r} \xrightarrow{\mathbf{u}} \lambda'$ between the right-hand side $\mathbf{r}$ of an arrow $\lambda \xrightarrow{\mathcal{R}} \mathbf{r}$ in $\mathcal{G}_\mathcal{R}$ and the left-hand side λ' of each arrow $\lambda' \xrightarrow{\mathcal{R}} \mathbf{r}'$ with which $\mathbf{r}$ unifies. We note that $\mathcal{G}_\mathcal{R}$ associates a path with every basic narrowing derivation issued from a given goal and that it does not require the inspection of the equation set to be solved, as opposed to [8]. We also note that $\mathcal{G}_\mathcal{R}$ consists of the graph of top symbols in [4, 5] when $\widehat{\mathbf{t}}$ is defined as the function $\widehat{\mathbf{t}} = \mathbf{f}$ if $\mathbf{t} = \mathbf{f}(\mathbf{t_1}, \ldots, \mathbf{t_n})$. In the following, $\to^*$ denotes a path in the graph that may contain arrows $\xrightarrow{\mathcal{R}}$ and arrows $\xrightarrow{\mathbf{u}}$.

Proposition 17. *Let $\mathcal{R}$ be a TRS and $\mathcal{G}_\mathcal{R}$ be the graph of terms built from $\mathcal{R}$ according to Definition 16. Then $\mathcal{G}_\mathcal{R}$ satisfies the condition of loop-check as established in Definition 8.*

Example 2. Let us consider the following level-canonical TRS $\mathcal{R}$ and its abstraction $\mathcal{R}_\mathcal{A}$. Note that $\mathcal{R}_\mathcal{A}$ yields no infinite abstract basic narrowing sequences.

$$
\begin{array}{ll}
\mathcal{R} = \{ & \\
\mathbf{r}1)\ \mathbf{h}(0) & \to\ 0. \\
\mathbf{r}2)\ \mathbf{f}(0) & \to\ 0. \\
\mathbf{r}3)\ \mathbf{f}(\mathbf{c}(\mathbf{X})) & \to\ \mathbf{c}(\mathbf{f}(\mathbf{X})) \Leftarrow \mathbf{g}(\mathbf{X}) = \mathbf{X}. \\
\mathbf{r}4)\ \mathbf{g}(\mathbf{c}(\mathbf{X})) & \to\ \mathbf{c}(\mathbf{X}). \ \}
\end{array}
\qquad
\begin{array}{ll}
\mathcal{R}_\mathcal{A} = \{ & \\
\mathbf{r}1_\mathcal{A})\ \mathbf{h}(0) & \to\ 0. \\
\mathbf{r}2_\mathcal{A})\ \mathbf{f}(0) & \to\ 0. \\
\mathbf{r}3_\mathcal{A})\ \mathbf{f}(\mathbf{c}(\mathbf{X})) & \to\ \mathbf{c}(\bot) \Leftarrow \mathbf{g}(\mathbf{X}) = \mathbf{X}. \\
\mathbf{r}4_\mathcal{A})\ \mathbf{g}(\mathbf{c}(\mathbf{X})) & \to\ \mathbf{c}(\mathbf{X}). \ \}
\end{array}
$$

We depict in Figure 1 the dependency graph built from $\mathcal{R}$. There is a cycle $\mathbf{f}(\mathbf{X}) \to^* \mathbf{f}(\mathbf{X}) \to^* \ldots$ in the graph.

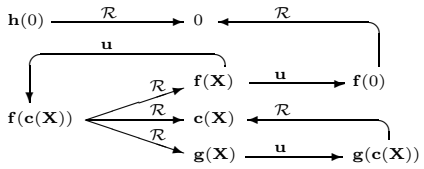

Fig. 1. *Graph of functional dependencies*

From Lemma 12, there is an abstract narrowing reduction that approximates each basic narrowing reduction from a given goal. The correspondence is most clearly seen in a diagram. We give here an example.

Example 3. Consider again the TRS $\mathcal{R}$ from Example 2. Figures 2 and 3 depict the search space of basic narrowing and abstract basic narrowing with initial state $\mathbf{s} = \langle \mathbf{h}(\mathbf{f}(\mathbf{Z})) = 0, \epsilon \rangle$. The leftmost derivation in Figure 3 is *relevant*. $\Delta(\mathcal{R}_\mathcal{A}, \mathbf{s}) = \{\mathbf{Z}/0\}$.

We now establish a preliminary result that clarifies our interest in abstract narrowing reduction.

Theorem 18. *Let $\mathcal{R}$ be a TRS, $\mathbf{s} = \langle \mathbf{g}, \epsilon \rangle$ and $\mathbf{s}' \propto \mathbf{s}$. Let $\Delta' = \Delta(\mathcal{R}_\mathcal{A}, \mathbf{s}')$. Then, for every solution θ computed in any successful basic narrowing derivation for $\mathcal{R}$ with $\mathbf{s}$, there exists $\kappa \in \Delta'$ such that $\kappa \propto \theta$.*

This theorem suggests that Δ' can be used to detect useless concrete computations, i.e. computations which do not lead to any success. We now introduce a relation $\diamondsuit$ which is used to compare two abstract substitutions. In particular, we use $\diamondsuit$ to compare the abstract substitutions in Δ' with concrete substitutions.

Definition 19. Let $\phi, \kappa \in \mathbf{Sub}^\bot$. We define the relation $\phi \diamondsuit \kappa$ by:

$$\phi \diamondsuit \kappa \ \textit{ iff } \ \exists \theta \in \mathbf{Sub}^\bot \ \textit{ such that } \ \phi \preceq \theta \textit{ and } \kappa \preceq \theta.$$

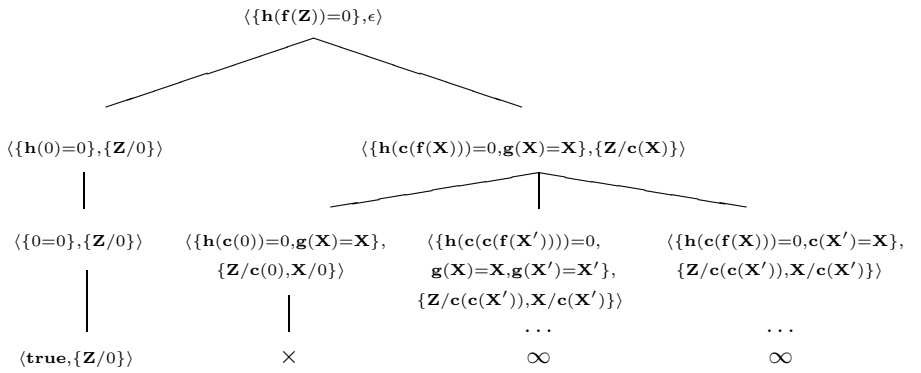

Fig. 2. *Basic narrowing*

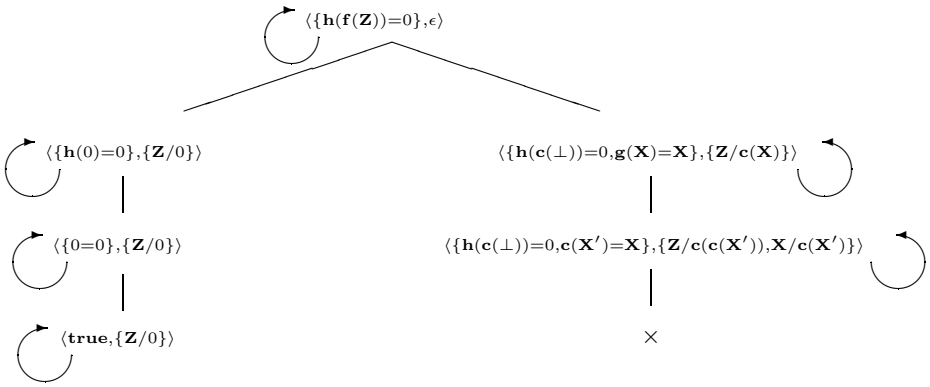

Fig. 3. *Abstract basic narrowing*

Roughly speaking, two abstract substitutions are in the relation $\diamondsuit$ if and only if they have a common instance. This definition does not provide much intuition about how the test $\diamondsuit$ on abstract substitutions can be performed. At the end of Section 5 we formulate an algorithm which decides this relation.

From Theorem 18 we derive a stronger characterization of *successful* narrowing derivations. Actually, successful derivations are distinguishable by the following property.

Theorem 20. *Let $\mathcal{R}$ be a TRS, $\mathbf{s} = \langle \mathbf{g}, \epsilon \rangle$ and $\mathbf{s}' \propto \mathbf{s}$. Let $\Delta' = \Delta(\mathcal{R}_\mathcal{A}, \mathbf{s}')$. Then, for every substitution θ in any successful basic narrowing derivation for $\mathcal{R}$ with $\mathbf{s}$, there exists $\kappa \in \Delta'$ such that $\kappa \diamondsuit \theta$.*

The following result follows immediately from Theorem 20 and represents the basis for our optimization:

Corollary 21. *Let $\mathcal{R}$ be a TRS and* $\mathbf{s}$ *an initial state. Let $\langle \mathbf{g}, \theta \rangle$ be any state of the (basic) conditional narrowing tree for $\mathcal{R}$ with* $\mathbf{s}$*. If there is no $\kappa \in \Delta(\mathcal{R}_{\mathcal{A}}, \mathbf{s})$ such that $\kappa \diamondsuit \theta$, then there is no successful basic narrowing derivation for $\langle \mathbf{g}, \theta \rangle$.*

Since the subtree originating in any node which satisfies the requirements of Corollary 21 does not contain any solution, it is useless to explore it. This remark suggests a method to reduce the size of the search tree while still retaining completeness as we formalize in the following section.

5 Using Abstract Narrowing to optimize Narrowing

Let us first introduce a *collecting* narrowing calculus which explores with a depth-first strategy the nodes of the abstract transition system for a given set of equations and picks up every abstract substitution in the terminal state of any *relevant* abstract basic narrowing derivation. Since the number of nodes to be visited is finite, any search strategy which visits the whole graph is complete. We use a depth-first strategy since it results in a 'programmed' search which is easy to be tackled formally and easy to implement.

Definition 22. **c***ollecting* $\mathcal{C}$*onditional* $\mathcal{N}$*arrowing calculus* (**c**$\mathcal{CN}$)
Let $(\Sigma, \mathcal{R}_{\mathcal{A}})$ be an abstract term rewriting system. A **c**$\mathcal{CN}$-state is a pair $\langle \mathbf{L}, \Delta \rangle$. The first component $\mathbf{L}$ is a list of pairs $(\mathbf{s}, \mathcal{S})$, where $\mathbf{s} = \langle \mathbf{g}, \kappa \rangle \in \mathbf{State}^{\perp}$ and $\mathcal{S}$ is a set. The elements of the set are triples $\langle \mathbf{e}, \mathbf{u}, \mathbf{r} \rangle$, where $\mathbf{e}$ is a Σ-equation, $\mathbf{u}$ is an occurrence of $\mathbf{e}$ and $\mathbf{r}$ is (a variant of) a reduction rule in $\mathcal{R}_{\mathcal{A}}$. $\langle\rangle$ is a distinguished element of the sets. The second component Δ of a **c**$\mathcal{CN}$-state is a set of (abstract) substitutions.

The **c**$\mathcal{CN}$ transition relation is defined as follows:

Unification Rule:

$$(1)\ \frac{\langle\rangle \notin \mathcal{S} \ \wedge \ \sigma = \mathbf{mgu}_{\mathcal{A}}(\mathbf{g}\kappa)}{\langle (\langle \mathbf{g}, \kappa \rangle, \mathcal{S}) \bullet \mathbf{L}, \Delta \rangle \rightarrow_{\mathbf{c}\mathcal{CN}} \langle (\langle \mathbf{g}, \kappa \rangle, \mathcal{S} \cup \{\langle\rangle\}) \bullet \mathbf{L}, \Delta \cup \kappa\sigma \rangle}$$

Narrowing Rule:

$$(2)\ \frac{\begin{array}{c}\mathbf{e} \in \mathbf{g} \ \wedge \ \mathbf{u} \in \bar{\mathbf{O}}(\mathbf{e}) \ \wedge \ \mathbf{r} = (\lambda \rightarrow \rho \Leftarrow \widetilde{\mathbf{e}}) \ll \mathcal{R}_{\mathcal{A}} \ \wedge \\ \langle \mathbf{e}, \mathbf{u}, \mathbf{r} \rangle \notin \mathcal{S} \ \wedge \ \sigma = \mathbf{mgu}_{\mathcal{A}}(\{(\mathbf{e}_{|\mathbf{u}})\kappa = \lambda\})\end{array}}{\begin{array}{l}\langle (\langle \mathbf{g}, \kappa \rangle, \mathcal{S}) \bullet \mathbf{L}, \Delta \rangle \rightarrow_{\mathbf{c}\mathcal{CN}} \\ \langle ((\mathbf{g} \sim \{\mathbf{e}\}) \cup \{\mathbf{e}[\rho]_{\mathbf{u}}\} \cup \widetilde{\mathbf{e}}, \kappa\sigma \rangle, \emptyset) \bullet (\langle \mathbf{g}, \kappa \rangle, \mathcal{S} \cup \{\langle \mathbf{e}, \mathbf{u}, \mathbf{r} \rangle\}) \bullet \mathbf{L}, \Delta \rangle\end{array}}$$

Removal Rule:

$$(3)\ \frac{\text{(1) and (2) do not apply}}{\langle (\langle \mathbf{g}, \kappa \rangle, \mathcal{S}) \bullet \mathbf{L}, \Delta \rangle \rightarrow_{\mathbf{c}\mathcal{CN}} \langle \mathbf{L}, \Delta \rangle}$$

Roughly speaking, the abstract state in the left part of a pair $(\mathbf{s}, \mathcal{S})$ represents a node of the abstract transition system. The right part of the pair records the redexes of the equation in the abstract state which are reduced together with the corresponding applied rule. $\langle\rangle$ represents the fact that an abstract mgu of the equations in the state has been found. List constructors are denoted by $[\,]$ and $\bullet$. The list representing an execution state of the above **c**ollecting $\mathcal{C}$onditional $\mathcal{N}$arrowing calculus is treated as a stack to emulate a depth-first strategy. The derivations can be represented as a finite tree. Leaves which have already been visited are simply removed. The calculus can be slightly improved if *failed* nodes are recognized and removed from the list. A *failed* node is any node $\langle \mathbf{g}, \kappa \rangle$ such that one equation $\mathbf{e} \in \mathbf{g}$ is not (abstractly) unifiable and no abstract narrowing step can be applied to any of the occurrences in $\mathbf{e}$.

Definition 23. *Behaviour of the* $\mathbf{c}\mathcal{CN}$ *calculus*
Let $\mathbf{L} = [\,(\,\mathbf{s}, \emptyset\,)\,]$, $\mathbf{s} \in \mathbf{State}$. Define the function $\mathbf{c}\mathcal{N}(\mathbf{s})$ as follows:

$$\mathbf{c}\mathcal{N}(\mathbf{s}) \;=\; \Delta \;\text{ if }\; \langle\, \mathbf{L}, \emptyset\,\rangle \rightarrow^{*}_{\mathbf{c}\mathcal{CN}} \langle\, [\,], \Delta\,\rangle.$$

We notice that in the case when $\mathbf{c}\mathcal{N}(\mathbf{s}) \;=\; \emptyset$, the set of equations in $\mathbf{s}$ is unsatisfiable.

The following theorem can easily be proved for the *collecting* narrowing algorithm given in Definitions 22 and 23.

Theorem 24. completeness of **c**ollecting $\mathcal{C}$onditional $\mathcal{N}$arrowing
Let $\mathcal{R}_{\mathcal{A}}$ be an abstract TRS. For every substitution κ in the terminal state of any relevant *abstract basic narrowing derivation for $\mathcal{R}_{\mathcal{A}}$ with* $\mathbf{s}$, $\kappa \in \mathbf{c}\mathcal{N}(\mathbf{s})$.

We can now refine the *basic $\mathcal{C}$onditional $\mathcal{N}$arrowing* calculus in Section 3 by considering the abstract substitutions which are gathered by **c**ollecting $\mathcal{C}$onditional $\mathcal{N}$arrowing.

We define a *refined $\mathcal{C}$onditional $\mathcal{N}$arrowing* calculus ($\mathbf{r}\mathcal{CN}$) by means of a two-rule transition system. To solve the equation set $\mathbf{g}_0$, the algorithm starts with the initial state $\mathbf{s} = \langle \mathbf{g}_0, \epsilon \rangle$ and tries to derive new goals until a terminal state $\langle \mathbf{true}, \sigma \rangle$ is reached. In the rules below, we assume $\Delta = \mathbf{c}\mathcal{N}(\mathbf{s})$.

Unification rule:

$$\frac{\sigma = \mathbf{mgu}(\mathbf{g}\theta)}{\langle \mathbf{g}, \theta \rangle \quad \leadsto_{\mathbf{r}\mathcal{CN}} \quad \langle \mathbf{true}, \theta\sigma \rangle}$$

Narrowing rule:

$$\frac{\begin{array}{l} \mathbf{e} \in \mathbf{g} \;\wedge\; \mathbf{u} \in \bar{\mathbf{O}}(\mathbf{e}) \;\wedge\; (\lambda \rightarrow \rho \Leftarrow \widetilde{\mathbf{e}}) \ll \mathcal{R} \;\wedge \\ \sigma = \mathbf{mgu}(\{(\mathbf{e}_{|\mathbf{u}})\theta = \lambda\}) \wedge \;\; \exists \rho \in \Delta.\;\; \rho \diamondsuit \theta\sigma \end{array}}{\langle \mathbf{g}, \theta \rangle \quad \leadsto_{\mathbf{r}\mathcal{CN}} \quad \langle (\mathbf{g} \sim \{\mathbf{e}\}) \cup \{\mathbf{e}[\rho]_{\mathbf{u}}\} \cup \widetilde{\mathbf{e}}, \theta\sigma \rangle}$$

We give an example which illustrates how the optimization works.

Example 4. Consider the TRS $\mathcal{R}$ from Example 3. Applying $\mathbf{r}\mathcal{CN}$ for $\mathcal{R}$ with initial state $\langle \mathbf{h}(\mathbf{f}(\mathbf{Z})) = 0, \epsilon \rangle$ results in the tree which is depicted in Figure 4. Let us notice that the subtree originating from the right-hand descendant of the root of the tree in Figure 2 has been eliminated. We note also that the removed subtree could not have been dropped using either of the standard simplifying strategies of narrowing which rest on the use of loop-checking [8], unification rules [13], operator joinability [12] or eager normalization [2, 12] techniques.

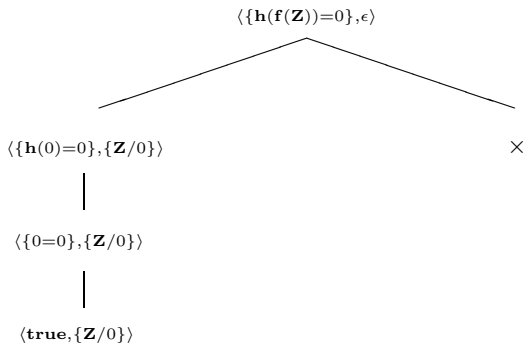

Fig. 4. *Refined basic narrowing*

The following theorem proves that our approach is sound and complete.

Theorem 25. correctness and completeness of **r**efined $\mathcal{C}$onditional $\mathcal{N}$arrowing
Let $\mathcal{E}$ be a level-canonical Horn equational theory with associated TRS $\mathcal{R}$. The set $\{\sigma_{\restriction \mathbf{Var}(\mathbf{g})} \mid \langle \mathbf{g}, \epsilon \rangle \leadsto^*_{\mathbf{r}\mathcal{CN}} \langle \mathbf{true}, \sigma \rangle\}$ is a complete set of $\mathcal{E}$-unifiers of $\mathbf{g}$.

Finally we show how the test $\diamondsuit$ of 'compatibility' on abstract substitutions can be effectively checked by providing an algorithm to decide it. First we need to extend the notion of *parallel composition* [25] from substitutions to abstract substitutions.

Parallel composition was proposed in [25] in order to provide a compositional characterization of the semantics of Horn Clause Logic. The formalization was also extended to a parallel execution model of logic programs. When two atoms (in the same goal) are run in parallel, the associated computed answer substitutions have to be combined afterwards in order to get the final result. Parallel composition of substitutions exactly corresponds to the least upper bound.

Roughly speaking, parallel composition is the operation of unification generalized to substitutions. It can be performed in the following way [25]. Given two idempotent substitutions ϑ_1 and ϑ_2, consider the set of all pairs corresponding to the bindings of both ϑ_1 and ϑ_2. Then, compute the most general unifier of such a set. Note that the *consistency check* can be thought of as a simple verification that such a set is unifiable.

We extend the notion of parallel composition from substitutions to abstract substitutions by replacing unification by abstract unification. In the following definition we introduce the notion of *abstract parallel composition*, denoted by $\Uparrow$.

Definition 26. *abstract parallel composition*
Let $\phi, \kappa \in \mathbf{Sub}^{\perp}$. We define the abstract parallel composition $\phi \Uparrow \kappa$ by:

$$\phi \Uparrow \kappa = \mathbf{mgu}_{\mathcal{A}}(\phi \cup \kappa).$$

Now the compatibility $\phi \diamondsuit \kappa$ of two abstract substitutions exactly corresponds to check that $\phi \Uparrow \kappa$ is not **fail**.

Proposition 27. *Let* $\phi, \kappa \in \mathbf{Sub}^{\perp}$. *Then* $\phi \diamondsuit \kappa$ iff $(\phi \Uparrow \kappa) \not\equiv \mathbf{fail}$.

6 Conclusions

We have formalized an optimization of basic conditional narrowing which makes use of an abstract interpretation algorithm to identify the narrowing derivations possibly leading to solutions. Thus the other derivations are not computed. The notions of *relevant* abstract basic narrowing derivation and *abstract parallel composition* have been introduced for this purpose. We have proved that the completeness result of the basic procedure is preserved in the refinement. An example illustrates the fact that this optimization can increase the efficiency of narrowing in addition to improving its termination. We note that the refinement applies also to full conditional narrowing, although the search tree exhibits, in this case, a much larger branching factor.

A prototype system based on the ideas described in this work has been implemented. In spite of the fact that it does not have the simplicity of simple basic narrowing, we have chosen to implement an *innermost selection* basic conditional narrowing procedure [3, 15, 16] since it further reduces the size of the search space.

It is difficult to state a general result for the efficiency improvement of the optimization. Table 1 gives an impression of some achievable speed-ups. We report the timings relative to some simple benchmarks for the program:

```
h(s(X)) → h(X) ⇐ g(s(X))=s(X), g(X)=X.
f(c(X)) → c(f(X)) ⇐ g(X)=X.
g(c(X)) → c(X).
g(s(X)) → s(X).
h(0) → 0.
f(0) → s(s(0)).
g(0) → 0.
```

In these tests we are able to avoid up to 70% of unnecessary narrowing attempts, with no prohibitive time overhead due to **c**ollecting $\mathcal{C}$onditional $\mathcal{N}$arrowing (column cT). Our experiments indicate that the strategy can be a useful tool in the optimization of equational logic programs.

Goal	first solution found		termination achieved		cT
	basic	*refined*	*basic*	*refined*	
g(h(s(0))) = X.	0.14	0.18	0.94	1.12	1.30
h(f(Z))=0.	8.22	1.48	∞	0.00	0.74
h(f(g(Z)))=0.	8.38	1.64	∞	0.00	1.02
f(g(X)) = Z, h(Z)=0.	111.56	13.44	∞	0.00	8.84

Table 1. *Basic* vs. *refined* narrowing times (secs, using BIM-Prolog, SUN 3/80)

Since our method can be seen as parametric with respect to the loop check, any improvement on the loop-checking technique would correspondingly yield to an improvement of the accuracy of our description. An interesting topic for further research is the question of how it is possible to improve our method by strengthening the construction of the dependency graph in order to get more accurate approximations. This seems dificult to do while retaining the ability to check the rewrite rules without inspecting the goal. Another topic of further research concerns the parallel implementation of the consistency check.

References

1. G. Aguzzi, U. Modigliani, and M.C. Verri. A Universal Unification Condition for Solving Goals in Equational Languages. In *Proc. of CTRS'90*, volume 516 of *Lecture Notes in Computer Science*, pages 416–423. Springer-Verlag, Berlin, 1990.
2. H. Aït-Kaci, P. Lincoln, and R. Nasr. Le Fun: Logic, equations, and Functions. In *Proc. of the Fourth IEEE Symposium on Logic Programming*, pages 17–23. IEEE Computer Society Press, 1987.
3. M. Alpuente, M. Falaschi, and G. Levi. Incremental Constraint Satisfaction for Equational Logic Programming. Technical Report TR-20/91, Dipartimento di Informatica, Università di Pisa, 1991. to appear in *Theoretical Computer Science*.
4. M. Alpuente, M. Falaschi, and F. Manzo. Analyses of Inconsistency for Incremental Equational Logic Programming. In M. Bruynooghe and M. Wirsing, editors, *Proc. of PLILP'92, Leuven (Belgium)*, volume 631 of *Lecture Notes in Computer Science*, pages 443–457. Springer-Verlag, Berlin, 1992.
5. M. Alpuente, M. Falaschi, and F. Manzo. Analyses of Unsatisfiability for Equational Logic Programming. Technical Report DSIC-II/29/92, UPV, 1992. submitted for publication.
6. M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. Narrowing Aproximations as an optimization for Equational Logic Programs. Technical Report DSIC-II/1/93, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 1993.
7. P. Bosco, E. Giovannetti, and C. Moiso. Narrowing vs. SLD-resolution. *Theoretical Computer Science*, 59:3–23, 1988.
8. J. Chabin and P. Réty. Narrowing directed by a graph of terms. In G. Goos and J. Hartmanis, editors, *Proc. of RTA'91*, volume 488 of *Lecture Notes in Computer Science*, pages 112–123. Springer-Verlag, Berlin, 1991.

9. J. Christian. Some termination criteria for narrowing and E-narrowing. In *11th Int'l Conf. on Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*, pages 582–588. Springer-Verlag, Berlin, 1992.
10. M. Codish, M. Falaschi, and K. Marriott. Suspension Analysis for Concurrent Logic Programs. In K. Furukawa, editor, *Proc. Eighth Int'l Conf. on Logic Programming*, pages 331– 345. The MIT Press, Cambridge, Mass., 1991.
11. 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.
12. N. Dershowitz and G. Sivakumar. Solving Goals in Equational Languages. In S. Kaplan and J. Joaunnaud, editors, *Proc. First Int'l Workshop on Conditional Term Rewriting*, volume 308 of *Lecture Notes in Computer Science*, pages 45–55. Springer-Verlag, Berlin, 1987.
13. 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.
14. J.H. Gallier and S. Raatz. Extending SLD-resolution to equational Horn clauses using E-unification. *Journal of Logic Programming*, 6:3–43, 1989.
15. 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.
16. S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1989.
17. 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.
18. 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.
19. J.W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume I. Oxford University Press, 1991.
20. 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.
21. M. J. Maher. Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees. In *Proc. Third IEEE Symp. on Logic In Computer Science*, pages 348–357. Computer Science Press, New York, 1988.
22. M. J. Maher. On parameterized substitutions. Technical Report RC 16042, IBM - T.J. Watson Research Center, Yorktown Heights, NY, 1990.
23. 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.
24. W. Nutt, P. Réty, and G. Smolka. Basic narrowing revisited. *Journal of Symbolic Computation*, 7:295–317, 1989.
25. C. Palamidessi. Algebraic properties of idempotent substitutions. In M. S. Paterson, editor, *Proc. of the 17th International Colloquium on Automata, Languages*

and Programming, volume 443 of *Lecture Notes in Computer Science*, pages 386–399. Springer-Verlag, Berlin, 1990.
26. P. Réty, C. Kirchner, H. Kirchner, and P. Lescanne. NARROWER: A new algorithm for unification and its applications to logic programming. In *Proc. of RTA'85*, volume 202 of *Lecture Notes in Computer Science*, pages 141–157. Springer-Verlag, Berlin, 1985.
27. J.H. Siekmann. Unification Theory. *Journal of Symbolic Computation*, 7:207–274, 1989.