# Optimization of Equational Logic Programs using Abstract Narrowing*

Germán Vidal †

**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 work, we define an *abstract narrower* for equational theories which can be used to obtain a finite description of the set of solutions. 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. We also introduce a parallel composition operator which can be used to perform computations and analysis incrementally in a Constraint Equational setting and show that the test of satisfiability in this setting can be done in parallel. Then, we define several applications of the framework. On one hand, we show a refined, but still complete, equation solving procedure which allows us to reduce the branching factor. On the other hand, we show how groundness and sharing information can be derived from the partial solutions of the abstract narrower. Finally, we give some benchmarks which show the usefulness of the method.

**Keywords:** Abstract interpretation, equational logic programming, term rewriting systems, universal unification, semantic analysis, compositionality, concurrency and parallelism, constraints.

---

*This work has been partially supported by CICYT under grant TIC 92-0793-C02-02.

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

# Contents

# 1 Introduction

The recent interest in logic programming with equations [18, 25, 30] has promoted much work on equational unification [21, 25, 44] and narrowing [25, 38]. Equational unification ($\mathcal{E}$-unification) characterizes the problem of solving equations *modulo* an equational theory $\mathcal{E}$. The narrowing mechanism is a powerful 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 [10, 25, 26, 38]. 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 [22, 25, 26, 38, 40]. In [10, 22] 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.

Compositionality is a desirable property which has been recognised as fundamental in the semantics of programming languages [20]. Compositionality has to do with a (syntactic) composition operator $\diamond$, and holds when the meaning (semantics) $\mathcal{S}(\mathbf{C}_1 \diamond \mathbf{C}_2)$ of a compound construct is defined by composing the meanings of the constituents $\mathcal{S}(\mathbf{C}_1)$ and $\mathcal{S}(\mathbf{C}_2)$, i.e. for a suitable function $\mathbf{f}_\diamond$, $\mathcal{S}(\mathbf{C}_1 \diamond \mathbf{C}_2) = \mathbf{f}_\diamond(\mathcal{S}(\mathbf{C}_1), \mathcal{S}(\mathbf{C}_2))$. In the case of logic programs [20], one could be concerned with AND-composition (of atoms in a goal or in a clause body) or with OR-composition, i.e. composition of (sets of) clauses. In the area of term rewriting systems there has been some investigation on which properties of the union of term rewriting systems are inherited from the corresponding properties of the separate constituents (see e.g. [31, 37]). In the case of equational unification, consideration has been given to the problem of determining the resulting *unification type* (unitary, finitary, infinitary, or nullary) for the union of two theories for which the unification type is known [44]. If the unification type of a theory is known, it is conceivable to apply special (and more efficient) algorithms to solve the equational unification problem. It has also been considered how to derive algorithms for unification in the union of (disjoint) equational theories (see e.g. [44]). In this paper, we address the problem of solving equations in equationally defined theories too but, unlike the above mentioned works, we want to define compositionally the meaning of the union of $\mathcal{E}$-unification problems. An $\mathcal{E}$-unification problem is a finite set of equations $\mathbf{E} = \{\mathbf{s}_1 = \mathbf{t}_1, \ldots, \mathbf{s_n} = \mathbf{t_n}\}$ together with an equational theory $\mathcal{E}$. Let a theory $\mathcal{E}$ be fixed and consider two finite equation sets $\mathbf{E}_1, \mathbf{E}_2$. We want

to define the meaning of $\mathbf{E}_1 \cup \mathbf{E}_2$ (with respect to $\mathcal{E}$) in terms of the meanings of $\mathbf{E}_1$ and $\mathbf{E}_2$.

The use of narrowing as the operational mechanism for computing leads to a powerful extension of ordinary logic programs [25, 42] and the computation model has many opportunities for parallelism. For example, [16, 17] describes a kind of OR-parallelism in which, for each position in the term and each rule in the program, alternative narrowings are explored concurrently according to some heuristic function. Our work concerns an AND-parallel computation model of equational Horn programs, where all subexpressions can be narrowed independently and the computed substitutions obtained so far can then be composed. This mechanism of computation was also mentioned in [42]. We extend the notion of *parallel composition* of substitutions introduced in [41] to the case of equational logic programs. Hence we show how complete sets of $\mathcal{E}$-unifiers for a given goal $\Leftarrow \mathbf{E}_1, \mathbf{E}_2$ can be generated by composing the complete sets of $\mathcal{E}$-unifiers computed by narrowing the separate subgoals $\Leftarrow \mathbf{E}_1$ and $\Leftarrow \mathbf{E}_2$. This allows us to model the combination of substitutions computed by AND-parallel 'narrowing processes', i.e. by agents which narrow subexpressions in parallel [9].

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. [6] 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, 5, 18, 19] 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 two ideas. First, the construction of a finite graph of terms built from the equational theory, which helps one to know the narrowing derivations which definitely terminate. Secondly, the approximation of the narrowing derivations by means of an abstract calculus whose derivations cover all concrete computations. [5, 6] 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 narrowing makes use of techniques for loop detection to ensure termination. Thus, when the abstract narrower terminates computing a "finitely failed tree", it states that the set of equations has no $\mathcal{E}$-unifier. Actually, a loop check on terms is a parameter for our construction [7]. We also present a specific strategy of

loop detection and show that the 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.

Recently, it has been defined an equational logic language [4] as an instance of Constraint Logic Programming [28], where the equations to be solved with respect to an equational theory are considered as constraints. For a computational step in this framework, it is essential to be able to (semi-)decide if a set of equations is satisfiable. The computation of complete sets of $\mathcal{E}$-unifiers is less striking in this context, while it is essential a mechanism to evaluate the satisfiability of the constraints incrementally. We show that a compositional narrowing can be taken as a basis for a compositional analysis for the problem of unsatisfiability. Then we show that compositionality leads, as a by-product, to an incremental implementation for the analysis. Therefore, while OR-compositionality, i.e. compositionality w.r.t. union of programs, has proven significant for programming with modules [20], we show that AND-compositionality can lead to incremental computations.

Then 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. 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.

This finite description of solutions of the set of $\mathcal{E}$-unifiers of $\mathbf{E}$ can also be used to accurately keep track of some non-trivial variable sharing information regarding the groundness and the independence of goal variables in the computation. This information is useful for a number of optimizations of equational Horn programs, notably in the exploitation of independent AND-parallelism, where different subexpressions can be narrowed in parallel if they are independent, i.e. they do not share unbound variables [27, 39], and the results of the analysis can avoid doing the corresponding checks at runtime.

The paper is organized as follows. After introducing some preliminary notions in Section 2, we present the basic solution procedure and its compositional version in Section 3. Section 4 recalls an abstract algorithm to analyze the unsatisfiability of equation sets [5, 6, 7], and give an incremental

definition of the algorithm using a compositional operator [9]. Section 5 defines a strategy that exploits the abstractions computed by this algorithm to reduce the alternatives that the solution procedure has to explore, and show a method to extract the groundness and sharing information from the abstract set. Finally, Section 6 presents some experimental results for our method and concludes.

## 2 Preliminaries

Let us first summarize some known results about equations, conditional rewrite systems and equational unification. For full definitions refer to [15, 31]. Throughout this paper, $\mathbf{V}$ will denote a countably infinite set of variables and $\Sigma$ 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 $\Sigma$ and $\mathbf{V}$, respectively. A $\Sigma$-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 [13]. 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**. Let $\mathbf{E}, \mathbf{E}'$ be equation sets, 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.

The equational representation of a substitution $\theta = \{\mathbf{x}_1/\mathbf{t}_1, \ldots, \mathbf{x}_\mathbf{n}/\mathbf{t}_\mathbf{n}\}$ is the set of equations $\widehat{\theta} = \{\mathbf{x}_1 = \mathbf{t}_1, \ldots, \mathbf{x}_\mathbf{n} = \mathbf{t}_\mathbf{n}\}$. The identity function on $\mathbf{V}$ is called the empty substitution and

denoted $\epsilon$.

We consider the usual preorder on substitutions $\leq$: $\theta \leq \sigma$ iff $\exists\gamma.\, \sigma \equiv \theta\gamma$. Note that $\theta \leq \sigma$ iff $\sigma \Rightarrow \theta$ [41]. A substitution $\{\mathbf{x_1}/\mathbf{t_1}, \ldots, \mathbf{x_n}/\mathbf{t_n}\}$ is a *unifier* of an equation set $\mathbf{E}$ iff $\{\mathbf{x_1} = \mathbf{t_1}, \ldots, \mathbf{x_n} = \mathbf{t_n}\} \Rightarrow \mathbf{E}$. 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 $\mathbf{E}$. In abuse of notation, we let **fail** denote failure when computing the $\mathbf{mgu}(\mathbf{E})$. While every unquantified equation set has a *most general unifier* [34], 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 $\Sigma$-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 $\Sigma$-equations. $\Sigma$-equations and $\Sigma$-theories will often be called equations and theories, respectively. An equational goal is an equational Horn clause with no head. We let *Goal* denote the set of equational goals. The set of *states* is defined by $\mathbf{State} = \mathbf{Goal} \times \mathbf{Sub}$.

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 $(\lambda \rightarrow \rho \Leftarrow \widetilde{\mathbf{e}})$, $\lambda, \rho \in \tau(\Sigma \cup \mathbf{V})$, $\lambda \notin \mathbf{V}$ 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 $(\Sigma, \mathcal{R})$. We let *Trs* denote the set of term rewriting systems.

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 [31].

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 $\Leftarrow \mathbf{g}$ conditionally narrows into a goal clause $\Leftarrow \mathbf{g'}$ (in symbols $\Leftarrow \mathbf{g} \stackrel{\theta}{\hookrightarrow} \Leftarrow \mathbf{g'}$) 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 $\theta$ such that $\theta = \mathbf{mgu}(\{\mathbf{e}_{|\mathbf{u}} = \lambda\})$ and $\mathbf{g}' = ((\mathbf{g} \sim \{\mathbf{e}\}) \cup \{\mathbf{e}[\rho]_{\mathbf{u}}\} \cup \tilde{\mathbf{e}})\theta$. A *narrowing derivation* is defined by $\Leftarrow \mathbf{g} \overset{\theta}{\hookrightarrow}^* \Leftarrow \mathbf{g}'$ iff $\exists \theta_1, \ldots, \theta_{\mathbf{n}}.\ \Leftarrow \mathbf{g} \overset{\theta_1}{\hookrightarrow} \ldots \overset{\theta_{\mathbf{n}}}{\hookrightarrow} \Leftarrow \mathbf{g}'$ and $\theta = \theta_1 \ldots \theta_{\mathbf{n}}$. If $\mathbf{n} = 0$ then $\theta = \epsilon$.

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

Each equational Horn theory $\mathcal{E}$ generates a smallest congruence relation $=_{\mathcal{E}}$ called $\mathcal{E}$-**equality** on the set of terms $\tau(\Sigma \cup \mathbf{V})$ (the least equational theory which contains all logic consequences of $\mathcal{E}$ under the entailment relation $\models$). $\mathcal{E}$ is a presentation or axiomatization of $=_{\mathcal{E}}$. In abuse of notation, we sometimes speak of the equational theory $\mathcal{E}$ to denote the theory axiomatized by $\mathcal{E}$. $\mathcal{E}$-equality is extended to substitutions by $\sigma =_{\mathcal{E}} \theta$ iff $\forall \mathbf{x} \in \mathbf{V}.\ \mathbf{x}\sigma =_{\mathcal{E}} \mathbf{x}\theta$.

A substitution $\sigma$ is an $\mathcal{E}$-**unifier** or a **solution** of the equation set $\mathbf{E}$ iff $\mathcal{E} \models (\widehat{\sigma} \Rightarrow \mathbf{E})$ [36]. The set $\mathcal{U}_{\mathcal{E}}(\mathbf{E})$ of all $\mathcal{E}$-**unifiers** of $\mathbf{E}$ is recursively enumerable [21, 25, 44].

For $\mathcal{E}$-**unification** problems, the notion of most general unifier generalizes to complete sets of minimal (incomparable) $\mathcal{E}$-**unifiers**. A set $\mathbf{S}$ of $\mathcal{E}$-unifiers of the equation set $\mathbf{E}$ is complete iff every $\mathcal{E}$-unifier $\sigma$ of $\mathbf{E}$ factors into $\sigma =_{\mathcal{E}} \theta\gamma$ for some substitutions $\theta \in \mathbf{S}$ and $\gamma$. A complete set of $\mathcal{E}$-unifiers of a system of equations may be infinite. Minimal complete sets $\mu\mathcal{U}_{\mathcal{E}}(\mathbf{E})$ of $\mathcal{E}$-**unifiers** of $\mathbf{E}$ do not always exist. An $\mathcal{E}$-**unification** procedure is complete if it generates a complete set of $\mathcal{E}$-**unifiers** for all input equation system. Conditional narrowing has been shown to be a complete $\mathcal{E}$-unification algorithm for canonical theories satisfying different restrictions [25, 38].

# 3 Narrowing Strategies

In this section, we introduce two refinements of the well-known narrowing algorithm. First, in Section 3.1 we recall the Basic Conditional Narrowing procedure, which has been proven to be a complete method for solving equations in the theory defined by a level-canonical conditional term rewriting system. Secondly, in Section 3.2, after introducing some preliminary notions about a parallel composition operator, we give a Compositional Conditional Narrowing procedure, which is proven sound and complete.

### 3.1 Basic Conditional Narrowing

Basic (conditional) narrowing is a restricted form of (conditional) narrowing where only terms at *basic* occurrences are considered to be narrowed [26, 38]. 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 [38].

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 [25]. The *skeleton* part is a set of equations $\mathbf{g}$ and the *environment* part is a substitution $\theta$. 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 $\theta$. 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 an initial state of the form $\langle \Leftarrow \mathbf{g}, \epsilon \rangle$ and tries to derive new states until a terminal state $\langle \Leftarrow \mathbf{true}, \sigma \rangle$ is reached. Each computed answer substitution $\sigma$ in a terminal state 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 state.

**Definition 3.1** Basic Conditional Narrowing calculus
*We formulate* **b***asic* $\mathcal{C}$*onditional* $\mathcal{N}$*arrowing calculus (*$\mathbf{b}\mathcal{CN}$*) as a transition system* $(\mathbf{State}, \leadsto_{\mathbf{b}\mathcal{CN}})$ *whose transition relation* $\leadsto_{\mathbf{b}\mathcal{CN}} \subseteq \mathbf{State} \times \mathbf{State}$ *is defined as the smallest relation satisfying the following rules:*

unification rule:

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

narrowing rule:

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

The operational semantics of Horn programs [14, 20, 25] is usually based on a transition system which describes how derivations (computations) are performed. In addition to the inference rules, an important aspect of the formalization is the concept of *observable*, that is the property we observe in a computation (e.g. the set of successes, the finite failure set, the infinite failure set, etc.). If we take the success set as observable, we can define (based on the above transition relation) the standard *success set* of an equational goal $\Leftarrow \mathbf{g}$ in the TRS $\mathcal{R}$ by a formal operational semantics $\mathcal{S} : \mathbf{Goal} \mapsto \wp\,\mathbf{Sub}$, as the set of the computed answers substitutions corresponding to all successful basic narrowing derivations for $\mathcal{R}$ with $\langle \Leftarrow \mathbf{g}, \epsilon \rangle$, that is,

$$\mathcal{S}(\Leftarrow \mathbf{g}) = \{\sigma \mid \langle \Leftarrow \mathbf{g}, \epsilon \rangle \leadsto^*_{\mathbf{bCN}} \langle \Leftarrow \mathbf{true}, \sigma \rangle\}$$

On the other hand, if we consider the set of all substitutions that may occur in any successful basic narrowing derivation (*partial success set*) as observable, we can introduce a new reference semantics which can be viewed as an extension of the standard semantics in order to gather essential informations about program executions. This *collecting semantics* is formalized in the following definition.

**Definition 3.2** Collecting semantics
*The collecting semantics for TRS $\mathcal{R}$ and initial state $\mathbf{s} = \langle \Leftarrow \mathbf{g}, \epsilon \rangle$ is the set of all substitutions in any state of a successful basic conditional narrowing derivation for $\mathcal{R}$ with $\mathbf{s}$, that is,*

$$\mathcal{C}(\Leftarrow \mathbf{g}) = \{\theta \mid \langle \Leftarrow \mathbf{g}, \epsilon \rangle \leadsto^*_{\mathbf{bCN}} \langle \Leftarrow \mathbf{g}', \theta \rangle \leadsto^*_{\mathbf{bCN}} \langle \Leftarrow \mathbf{true}, \sigma \rangle\}$$

## 3.2 Compositional Narrowing

In this section, we begin extending the parallel composition operator to the equational case, and follow giving a compositional version of the basic narrowing procedure we have just seen (Definition 3.1).

### 3.2.1 Equational Parallel Composition

In the following, we recall the notion of *parallel composition* of substitutions, denoted by $\Uparrow$. Parallel composition was proposed in [41] as a compositional characterization of the semantics of Horn Clause Logic. Roughly speaking, parallel composition is the operation of unification generalized to substitutions.

Parallel composition corresponds to one of the basic operations performed by the AND-parallel execution model of logic programs. Namely, when two subgoals (of the same goal) are run in parallel, the answer substitutions (computed independently) have to be combined to get the final result. This 'combination' can be done as follows [34, 41]. Given two idempotent substitutions $\theta_1$ and $\theta_2$, we let $\theta_1 \Uparrow \theta_2 = \mathbf{mgu}(\widehat{\theta_1} \cup \widehat{\theta_2})$. Parallel composition is idempotent, commutative, associative and has a null element **fail** and an identical element $\epsilon$ [41]. $\Uparrow$ is lifted to sets of substitutions by

$$\Theta_1 \Uparrow \Theta_2 = \begin{cases} \bigcup_{\theta_1 \in \Theta_1, \theta_2 \in \Theta_2} \theta_1 \Uparrow \theta_2 & \text{if it is different from } \{\mathbf{fail}\} \\ \emptyset & \text{otherwise.} \end{cases}$$

It is immediate to generalize the notion of parallel composition to the case when unification in equational theories is considered [9]. In the following definition we formalize the notion of *equational parallel composition*, denoted by $\Uparrow_{\mathcal{E}}$.

**Definition 3.3** *Let $\theta_1, \theta_2 \in \mathbf{Sub}$. We define the operator $\Uparrow_{\mathcal{E}} : \mathbf{Sub} \times \mathbf{Sub} \rightarrow \wp(\mathbf{Sub})$ as follows:*

$$\theta_1 \Uparrow_{\mathcal{E}} \theta_2 = \mathcal{U}_{\mathcal{E}}(\widehat{\theta_1} \cup \widehat{\theta_2}).$$

**Example 1** *Let $\mathcal{E} = \{\mathbf{f}(0) = 0,\ \mathbf{f}(\mathbf{g}(\mathbf{X})) = \mathbf{g}(\mathbf{X}),\ \mathbf{g}(0) = \mathbf{c}(0),\ \mathbf{g}(\mathbf{c}(\mathbf{X})) = \mathbf{g}(\mathbf{X})\}$.*

1. *Let $\theta_1 = \{\mathbf{X}/\mathbf{g}(\mathbf{Z})\}$ and $\theta_2 = \{\mathbf{X}/\mathbf{c}(\mathbf{Z})\}$. Then $\{\mathbf{X}/\mathbf{c}(0), \mathbf{Z}/0\} \in \theta_1 \Uparrow_{\mathcal{E}} \theta_2$.*
2. *Let $\theta_1 = \{\mathbf{X}/\mathbf{f}(0)\}$ and $\theta_2 = \{\mathbf{X}/\mathbf{g}(\mathbf{Z})\}$. Then $\theta_1 \Uparrow_{\mathcal{E}} \theta_2 = \emptyset$.*

It is straightforward to show that $\Uparrow_{\mathcal{E}}$ is commutative and associative and has a null element $\{\mathbf{fail}\}$ and an identical element $\{\epsilon\}$. The operator $\Uparrow_{\mathcal{E}}$ can be lifted to sets of substitutions as follows.

**Definition 3.4** *Given $\Theta_1, \Theta_2 \in \wp(\mathbf{Sub})$, let:*

$$\Theta_1 \Uparrow_{\mathcal{E}} \Theta_2 = \bigcup_{\theta_1 \in \Theta_1, \theta_2 \in \Theta_2} \theta_1 \Uparrow_{\mathcal{E}} \theta_2.$$

Given an $\mathcal{E}$-unification problem $\mathbf{E} = \mathbf{E}_1 \cup \mathbf{E}_2$, a compositional characterization of the set of all $\mathcal{E}$-unifiers of $\mathbf{E}$ is given in the following proposition.

**Proposition 3.5** *Let* $\mathbf{E} = \mathbf{E}_1 \cup \mathbf{E}_2$, $\Theta_1 = \mathcal{U}_\mathcal{E}(\mathbf{E}_1)$ *and* $\Theta_2 = \mathcal{U}_\mathcal{E}(\mathbf{E}_2)$. *Then* $\mathcal{U}_\mathcal{E}(\mathbf{E}) = \Theta_1 \Uparrow_\mathcal{E} \Theta_2$.

We note that it is much more complex to evaluate $\Uparrow_\mathcal{E}$ than the much simpler operation $\Uparrow$. However, equational parallel composition can be redefined in terms of 'most general' unifiers in the case of *finitary* theories, for which the solutions to an $\mathcal{E}$-unification problem $\mathbf{E}$ can always be represented by a complete and minimal finite set $\mu\mathcal{U}_\mathcal{E}(\mathbf{E})$ of (maximally general) $\mathcal{E}$-unifiers, which is unique up to equivalence [44]. Equational theories which are of finitary *unification type* play an important role in logic programming with equality [29]. In general, the unification type of an equational theory is undecidable. On the other hand, for a finitary theory the minimality requirement is often too strong, since an algorithm which generates a superset of $\mu\mathcal{U}_\mathcal{E}(\mathbf{E})$ may be far more efficient than a minimal one and hence sometimes preferable. In the next point, we will show that we can still work with ordinary parallel composition $\Uparrow$ when we consider the class of (level-)canonical equational theories, for which the problem of $\mathcal{E}$-unification reduces to ordinary (syntactic) unification plus narrowing [26].

### 3.2.2 Compositional Conditional Narrowing

We now give a compositional characterization of the operational semantics of equational Horn programs [9] in a style similar to that of [41] for logic programs. We define a new narrowing procedure by means of a transition relation from equational goals to equational goals, labeled on substitutions.

**Definition 3.6** Compositional Conditional Narrowing
*We define* compositional conditional narrowing *as a labelled transition system* $(\mathbf{Goal}, \mathbf{Sub}, \longmapsto)$ *whose transition relation* $\longmapsto \subseteq \mathbf{Goal} \times \mathbf{Sub} \times \mathbf{Goal}$ *is the smallest relation which satisfies*

$$(1)\quad \frac{\mathbf{u} \in \bar{\mathbf{O}}(\mathbf{e}) \ \wedge\ (\lambda \to \rho \Leftarrow \tilde{\mathbf{e}}) \ll \mathcal{R} \ \wedge\ \sigma = \mathbf{mgu}(\{\mathbf{e}_{|\mathbf{u}} = \lambda\})}{\Leftarrow \{\mathbf{e}\} \overset{\sigma}{\longmapsto} \Leftarrow \{\mathbf{e}[\rho]_\mathbf{u}\} \cup \tilde{\mathbf{e}}}$$

$$(2)\quad \frac{\Leftarrow \mathbf{g}_1 \overset{\theta_1}{\longmapsto} \Leftarrow \mathbf{g}_1' \ \wedge\ \Leftarrow \mathbf{g}_2 \overset{\theta_2}{\longmapsto} \Leftarrow \mathbf{g}_2'}{\Leftarrow \mathbf{g}_1, \mathbf{g}_2 \overset{\theta_1 \Uparrow \theta_2}{\longmapsto} \Leftarrow \mathbf{g}_1', \mathbf{g}_2'}$$

Roughly speaking, in the computation model formalized in the transition system above, all equations in the equational goal to be solved are reduced at the same time. Then, the substitution resulting from these local computations are combined by means of the operator of *parallel composition* to obtain the global result of the computation. By abuse, we consider $\Leftarrow \mathbf{true} \stackrel{\epsilon}{\longmapsto} \Leftarrow \mathbf{true}$.

The computation model formalized in Definition 3.6 could be taken as a basis for an AND-parallel computation model of equational Horn programs. We note that the model has not been devised to achieve maximal parallelism in the sense that not all redexes in a given goal are allowed to perform one narrowing step independently. Namely, redexes which occur in a same equation are not reduced in parallel, while they could. To overcome this problem, it suffices to introduce the following *unfolding* rule, which preserves the reachable solutions

$$(3) \quad \frac{\mathbf{e} \in \mathbf{g} \ \wedge \ \mathbf{u} \in \bar{\mathbf{O}}(\mathbf{e}) \ \wedge \ \mathbf{x} \text{ is a new variable}}{\Leftarrow \mathbf{g} \stackrel{\epsilon}{\longmapsto} \Leftarrow (\mathbf{g} \sim \{\mathbf{e}\}) \cup \{\mathbf{e}[\mathbf{x}]_{\mathbf{u}}\} \cup \{\mathbf{x} = \mathbf{e}_{|\mathbf{u}}\}}$$

provided that both $\mathbf{e}_{|\mathbf{u}}$ and $\mathbf{e}[\mathbf{x}]_{\mathbf{u}}$ contain at least one function symbol [40]. Note that we need not determine the level of granularity [33] (as it does not affect correctness); this we consider to be an implementation issue that could enable (more) effective parallelizations.

Our approach differs from other AND-parallel execution models, such as e.g. [33], where subexpressions are only narrowed in parallel if they are *independent*, i.e. if they do not share (unbound) variables. A 'need-driven' syncronization model is imposed which compels processes to wait for the value of a parallel subexpression if such a value is needed. In [32], a *dependent* AND-parallel execution model is exploited, but the imposed syncronization mechanisms produce too much overhead.

A new operational semantics of equational Horn programs can now be defined by

**Definition 3.7** $\mathcal{S}'(\Leftarrow \mathbf{g}) = \{\theta_1 \Uparrow \ldots \Uparrow \theta_{\mathbf{n}} \mid \Leftarrow \mathbf{g} \stackrel{\theta_1}{\longmapsto} \ldots \stackrel{\theta_{\mathbf{n}}}{\longmapsto} \Leftarrow \mathbf{true}\}$.

In Definition 3.7 we use parallel composition because, in the transition system in Definition 3.6, the computed substitution is not applied to either the derived goal or to the redex $\mathbf{e}_{|\mathbf{u}}$ selected to be narrowed, as opposed to the standard semantics. Therefore, the next computation step will not take this substitution into account and the next substitution that is computed has to be combined with the previous one.

The new success set semantics $\mathcal{S}'$ is compositional w.r.t. the AND operator. Formally,

**Theorem 3.8** $\mathcal{S}'(\Leftarrow \mathbf{g}_1, \mathbf{g}_2) = \mathcal{S}'(\Leftarrow \mathbf{g}_1) \Uparrow \mathcal{S}'(\Leftarrow \mathbf{g}_2)$.

The following result states that the compositional semantics and the standard basic conditional semantics coincide. The correspondence is restricted to successes, namely to the substitutions computed by all succesfully terminating derivations. The result is a consequence of Theorem 3.8 and the following lemma.

**Lemma 3.9** ([34, 41]) *Let $\vartheta_1$, $\vartheta_2$ be idempotent substitutions. Then*

1. *If $\vartheta_1 = \{\widetilde{\mathbf{x}}/\widetilde{\mathbf{s}}\}$, $\vartheta_2 = \{\widetilde{\mathbf{y}}/\widetilde{\mathbf{t}}\}$ and $\widetilde{\mathbf{x}} \cap \mathbf{Var}(\vartheta_2) \equiv \emptyset$, then $\widehat{\vartheta_1\vartheta_2} \Leftrightarrow \widehat{\vartheta_1} \cup \widehat{\vartheta_2}$.*
2. *$\vartheta_1 \Uparrow \vartheta_2 = \vartheta_1\mathbf{mgu}(\widehat{\vartheta_2}\vartheta_1) = \vartheta_2\mathbf{mgu}(\widehat{\vartheta_1}\vartheta_2)$.*

The following theorems establish the equivalence of the compositional operational semantics with the standard one for conditional narrowing.

**Corollary 3.10** $\langle \Leftarrow \mathbf{g}, \epsilon\rangle \leadsto^*_{\mathbf{bC}\mathcal{N}} \langle \Leftarrow \mathbf{true}, \theta\rangle$ *iff* $\Leftarrow \mathbf{g} \stackrel{\theta_1}{\longmapsto} \ldots \stackrel{\theta_\mathbf{n}}{\longmapsto} \Leftarrow \mathbf{true}$ *and* $\theta = \theta_1 \Uparrow \ldots \Uparrow \theta_\mathbf{n}$, $\theta \not\equiv \mathbf{fail}$.

As a consequence of Corollary 3.10, every solution found by basic conditional narrowing is found by compositional conditional narrowing as well. This establishes, for level-canonical systems, the completeness of the later procedure *via* the completeness of the former.

**Corollary 3.11** (completeness) *The set $\{\vartheta_{\restriction\mathbf{Var}(\mathbf{g})} \mid \vartheta \in \mathcal{S}'(\Leftarrow \mathbf{g})\}$ is a complete set of $\mathcal{E}$-unifiers of $\mathbf{g}$.*

Note that, by not using the information specified in the substitutions, compositional conditional narrowing may have to overcompute when compared to (basic) conditional narrowing. For example, suppose that the narrowings for $\Leftarrow \mathbf{g}_1$ and $\Leftarrow \mathbf{g}_2$ produce the substitutions $\theta_1$ and $\theta_2$, respectively. We can now compose $\theta_1$ and $\theta_2$ to compute the substitution $\theta = \theta_1 \Uparrow \theta_2$, which is a narrowing substitution for $\Leftarrow \mathbf{g}_1, \mathbf{g}_2$. The more specific subexpressions in $\Leftarrow \mathbf{g}_2\theta_1$ in general have fewer narrowings than the more general subexpressions in $\Leftarrow \mathbf{g}_2$.

From Corollary 3.10 it follows that, when considering successes, the standard operational semantics based on basic conditional narrowing is also compositional. In Section 4.2, we show how this semantics can be used for program analysis.

# 4 Abstract Interpretation

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 3.1. 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 [13].

## 4.1 Abstract Basic Conditional Narrowing

Most of the definitions at the beginning of this section are already in [5, 6] and are reported for completeness. A difference with respect to [5] is in the definition of abstract most general unifier (Definition 4.5) which has been revised and simplified. Another difference is that we present Definition 4.10 as parametric with respect to a functional dependency graph (or loop-check). The corresponding definition in [6] 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* $(\mathbf{C}, \leq)$ (a poset). When $\mathbf{C} = \mathbf{Eqn}$, $\mathbf{C} = \mathbf{Sub}$ or $\mathbf{C} = \mathbf{State}$, 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 $\gamma : \mathbf{D} \to \wp\mathbf{C}$. 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 [6].

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 [5, 6]. 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 4.1** term, equation poset
*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* $\mathbf{E}, \mathbf{E'}$*:*

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

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

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 [6, 35, 36]. Define $[\![\mathbf{E}]\!] = \mathbf{E'}$, where the n-tuple of occurrences of $\perp$ in $\mathbf{E}$ is replaced by an n-tuple of existentially quantified distinct fresh variables in $\mathbf{E'}$. We note that $\mathbf{E'} \preceq \mathbf{E}$ implies $[\![\mathbf{E}]\!] \Rightarrow [\![\mathbf{E'}]\!]$ [6].

**Definition 4.2** *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* $[\![\theta]\!] \Rightarrow [\![\kappa]\!]$.

Let us introduce the abstract domains which we will use in our analysis.

**Definition 4.3** upward

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

**Definition 4.4** term, equation, substitution, state description

*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* $\gamma : \mathcal{T}^{\perp} \rightarrow \wp\mathcal{T}$ *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* $\mathbf{Sub}^{\perp}$ *as* $\mathbf{State}^{\perp} = \{\langle \Leftarrow \mathbf{g}, \kappa\rangle \mid \mathbf{g} \in \mathbf{Eqn}^{\perp}, \kappa \in \mathbf{Sub}^{\perp}\}$.

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 [13]. A formal description can be found in [5, 6] and is omitted here for brevity.

An *abstract term rewriting system* is a finite set of abstract reduction rules of the form $(\lambda \to \rho \Leftarrow \widetilde{\mathbf{e}})$, $\lambda \in \tau(\Sigma \cup \mathbf{V})$, $\rho \in \tau(\Sigma \cup \mathbf{V} \cup \{\perp\})$, $\lambda \notin \mathbf{V}, \mathbf{Var}(\rho) \subseteq \mathbf{Var}(\lambda)$ and $\widetilde{\mathbf{e}} \in \mathbf{Eqn}^{\perp}$. We let $\mathit{Trs}^{\perp}$ denote the set of abstract term rewriting systems. 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 4.5** Abstract most general unifier
*Let* $\mathbf{solve}(\llbracket \mathbf{E}' \rrbracket) = \exists \mathbf{y_1}, \ldots, \mathbf{y_n}.\ \mathbf{E}$ *and* $\kappa = \{\mathbf{y}_1/\perp, \ldots, \mathbf{y_n}/\perp\}$. *Then,* $\mathbf{mgu}_{\mathcal{A}}(\mathbf{E}') = \mathbf{E}\kappa$.

**Example 2** *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 4.6** [6] $\forall \theta \in \mathbf{unif}([\![\mathbf{E}]\!]).\ \mathbf{mgu}_{\mathcal{A}}(\mathbf{E}) \preceq \theta$.

The safety of the abstract unification algorithm has been proven in [6] as stated by the following proposition.

**Proposition 4.7** [6] *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, 11, 12, 26, 43]. For example, [11, 43] 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 [11, 43] 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 4.8** 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:*

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

*then*

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

*(we refer to* $\langle \widehat{\mathbf{t}_\mathbf{i}}, \widehat{\mathbf{t}_\mathbf{i}} \rangle$ *as a 'cycle' of* $\mathcal{G}_{\mathcal{R}}^{+}$.*)*

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 as stated by the following proposition.

**Proposition 4.9** *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 4.10** 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*

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

We define the term rewriting systems description as follows.

**Definition 4.11** *Let $(\mathrm{Trs}, \subseteq)$ be the poset of term rewriting systems and $(\mathrm{Trs}^\bot, \subseteq)$ be the poset of abstract term rewriting systems ordered by set inclusion. The concretization function for abstract term rewriting system description is $\langle (\mathbf{Trs}^\bot, \subseteq), \gamma, (\mathbf{Trs}, \subseteq) \rangle$, where $\gamma : \mathrm{Trs}^\bot \to \wp(\mathrm{Trs})$ is:*

$$\gamma(\mathcal{R}_\mathcal{A}) = \{\mathcal{R} \in \mathrm{Trs} \mid (\lambda \to \rho_\mathcal{A} \Leftarrow \widetilde{\mathbf{e}}_\mathcal{A}) \in \mathcal{R}_\mathcal{A} \ \wedge \ \rho_\mathcal{A} = \mathbf{sh}(\rho) \ \wedge \ \widetilde{\mathbf{e}}_\mathcal{A} = \mathbf{sh}(\widetilde{\mathbf{e}}) \ \wedge \ (\lambda \to \rho \Leftarrow \widetilde{\mathbf{e}}) \in \mathcal{R}\}$$

We can now define abstract basic narrowing.

**Definition 4.12** *Let $\mathcal{R}_{\mathcal{A}}$ be an abstract* **TRS**. *We define* **a***bstract (basic)* $\mathcal{C}$*onditional* $\mathcal{N}$*arrowing as a transition system* $(\mathbf{State}^{\perp}, \leadsto_{\mathbf{a}\mathcal{C}\mathcal{N}})$ *whose transition relation* $\leadsto_{\mathbf{a}\mathcal{C}\mathcal{N}} \subseteq \mathbf{State}^{\perp} \times \mathbf{State}^{\perp}$ *is the smallest relation which satisfies:*

abstract unification rule:

$$\frac{\sigma = \mathbf{mgu}_{\mathcal{A}}(\mathbf{g}\kappa)}{\langle \Leftarrow \mathbf{g}, \kappa \rangle \quad \leadsto_{\mathbf{a}\mathcal{CN}} \quad \langle \Leftarrow \mathbf{true}, \kappa\sigma \rangle}$$

abstract narrowing rule:

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

abstract self rule:

$$\frac{}{\langle \Leftarrow \mathbf{g}, \kappa \rangle \quad \leadsto_{\mathbf{a}\mathcal{CN}} \quad \langle \Leftarrow \mathbf{g}, \kappa \rangle}$$

*Let* $\mathbf{s}, \mathbf{t} \in \mathbf{State}^{\perp}$. *We say that there is an* abstract (basic) narrowing *reduction from* $\mathbf{s}$ *to* $\mathbf{t}$ *iff* $\mathbf{s} \leadsto_{\mathbf{a}\mathcal{CN}} \mathbf{t}$.

The following lemma is proven in [6] and basically states that abstract (basic) narrowing reduction approximates (basic) narrowing reduction.

**Lemma 4.13** [6] *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 4.14** [6] *Let* $\mathcal{R}_{\mathcal{A}}$ *be an abstract TRS. The (abstract) transition system for* $\leadsto_{\mathbf{a}\mathcal{CN}}$, *state* $\mathbf{s}$ *and* $\mathbf{State}^{\perp}$ *has a finite number of nodes.*

The following auxiliary notions will be used below.

**Definition 4.15** *An abstract (basic) narrowing derivation for* $\mathcal{R}_{\mathcal{A}}$, $\mathbf{s}$ *and* $\mathbf{State}^{\perp}$ *is* relevant *if the last node in the derivation is labeled by* $\langle \Leftarrow \mathbf{true}, \kappa \rangle$.

**Definition 4.16** Abstract semantics

*Let* $\mathcal{R}_{\mathcal{A}}$ *be an abstract TRS. Define* $\Delta(\Leftarrow \mathbf{g})$ *to be the set of abstract substitutions* $\kappa$ *in the terminal states* $\langle \Leftarrow \mathbf{true}, \kappa \rangle$ *of the abstract (basic) narrowing derivations which are relevant for* $\mathcal{R}_{\mathcal{A}}$, $\langle \Leftarrow \mathbf{g}, \epsilon \rangle$ *and* $\mathbf{State}^{\perp}$. *Formally:*

$$\Delta(\Leftarrow \mathbf{g}) = \{\kappa \in \mathbf{Sub}^{\perp} \mid \langle \Leftarrow \mathbf{g}, \epsilon \rangle \leadsto^{*}_{\mathbf{a}\mathcal{CN}} \langle \Leftarrow \mathbf{true}, \kappa \rangle\}.$$

$\Delta(\Leftarrow \mathbf{g})$ is the set of the 'computed answer abstract substitutions' for $\Leftarrow \mathbf{g}$. In Section 5.1 we show that $\Delta(\Leftarrow \mathbf{g})$ 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 dependency graph $\mathcal{I}_{\mathcal{R}}$ that is 'induced' by a term rewriting system $\mathcal{R}$. Then we show that $\mathcal{I}_{\mathcal{R}}$ is a loop-check. Another different, less accurate, loop-check can be found in [5, 6].

**Definition 4.17** *Let* $\mathcal{R}$ *be a TRS. The following transformation defines a directed graph* $\mathcal{I}_{\mathcal{R}}$ *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* $\mathcal{I}_{\mathcal{R}}$, *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.*

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

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

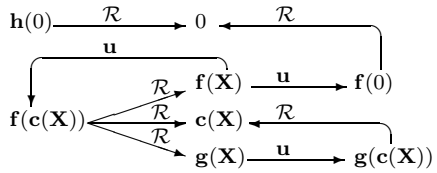


Figure 1: *Graph of functional dependencies*

Termination of this calculus is ensured since the number of terms occurring in the rules in $\mathcal{R}$ is finite. Roughly speaking, in Definition 4.17, 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 $\rho$ or in $\widetilde{\mathbf{e}}$, rule (1) adds an arrow $\lambda \stackrel{\mathcal{R}}{\to} \mathbf{f}(\lfloor \mathbf{t_1} \rfloor, \ldots, \lfloor \mathbf{t_n} \rfloor)$ to $\mathcal{I}_\mathcal{R}$. Rule (2) adds an arrow $\mathbf{r} \stackrel{\mathbf{u}}{\to} \lambda'$ between the right-hand side $\mathbf{r}$ of an arrow $\lambda \stackrel{\mathcal{R}}{\to} \mathbf{r}$ in $\mathcal{I}_\mathcal{R}$ and the left-hand side $\lambda'$ of each arrow $\lambda' \stackrel{\mathcal{R}}{\to} \mathbf{r}'$ with which $\mathbf{r}$ unifies. Note that $\mathcal{I}_\mathcal{R}$ 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 [11]. Also note that $\mathcal{I}_\mathcal{R}$ consists of the graph of top symbols in [5, 6] 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 $\stackrel{\mathcal{R}}{\to}$ and arrows $\stackrel{\mathbf{u}}{\to}$.

The following proposition shows that the induced dependency graph $\mathcal{I}_\mathcal{R}$ is a loop-check.

**Proposition 4.18** *Let $\mathcal{R}$ be a TRS and $\mathcal{I}_\mathcal{R}$ be the graph of terms induced by $\mathcal{R}$. Then $\mathcal{I}_\mathcal{R}$ is a loop-check.*

**Example 3** *Let us consider the following level-canonical TRS $\mathcal{R}$ and its abstraction $\mathcal{R}_\mathcal{A}$.*

$\mathcal{R} = \{$

$\mathbf{r}1)\ \mathbf{h}(0) \to 0.$

$\mathbf{r}2)\ \mathbf{f}(0) \to 0.$

$\mathbf{r}3)\ \mathbf{f}(\mathbf{c}(\mathbf{X})) \to \mathbf{c}(\mathbf{f}(\mathbf{X})) \Leftarrow \mathbf{g}(\mathbf{X}) = \mathbf{X}.$

$\mathbf{r}4)\ \mathbf{g}(\mathbf{c}(\mathbf{X})) \to \mathbf{c}(\mathbf{X}).\ \}$

$\mathcal{R}_\mathcal{A} = \{$

$\mathbf{r}1_\mathcal{A})\ \mathbf{h}(0) \to 0.$

$\mathbf{r}2_\mathcal{A})\ \mathbf{f}(0) \to 0.$

$\mathbf{r}3_\mathcal{A})\ \mathbf{f}(\mathbf{c}(\mathbf{X})) \to \mathbf{c}(\bot) \Leftarrow \mathbf{g}(\mathbf{X}) = \mathbf{X}.$

$\mathbf{r}4_\mathcal{A})\ \mathbf{g}(\mathbf{c}(\mathbf{X})) \to \mathbf{c}(\mathbf{X}).\ \}$

*Note that $\mathcal{R}_\mathcal{A}$ yields no infinite abstract basic narrowing sequences. We depict in Figure 1 the dependency graph induced by $\mathcal{R}$. There is a cycle $\mathbf{f}(\mathbf{X}) \to^* \mathbf{f}(\mathbf{X}) \to^* \ldots$ in the graph.*

From Lemma 4.13, 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.

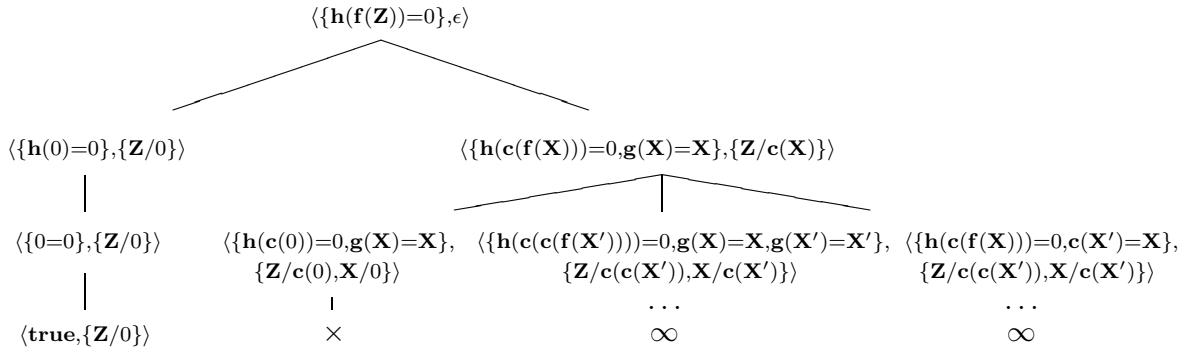


Figure 2: *Basic narrowing*

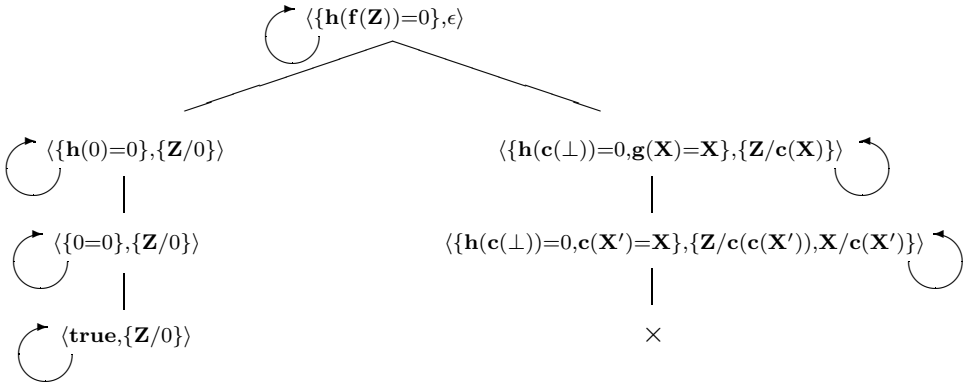


Figure 3: *Abstract basic narrowing*

**Example 4** *Consider again the TRS $\mathcal{R}$ from Example 3. Figures 2 and 3 depict the search space of basic narrowing and abstract basic narrowing with initial state* $\mathbf{s} = \langle \Leftarrow \mathbf{h}(\mathbf{f}(\mathbf{Z})) = 0, \epsilon \rangle$. *The leftmost derivation in Figure 3 is* relevant. $\Delta(\Leftarrow \mathbf{h}(\mathbf{f}(\mathbf{Z})) = 0) = \{\mathbf{Z}/0\}$.

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

**Theorem 4.19** *Let $\mathcal{R}$ be a TRS and $\mathbf{g}' \propto \mathbf{g}$. Then, for every computed answer substitution $\theta \in \mathcal{S}(\Leftarrow \mathbf{g})$ there exists $\kappa \in \Delta(\Leftarrow \mathbf{g}')$ such that $\kappa \propto \theta$.*

**Corollary 4.20** *If $\Delta(\Leftarrow \mathbf{g}) = \emptyset$, then $\mathbf{g}$ is unsatisfiable.*

## 4.2 Incremental Equational Analyzer

We now 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 $\Uparrow_{\mathcal{A}}$.

**Definition 4.21** *Let $\kappa_1, \kappa_2 \in \mathbf{Sub}^{\perp}$. We define the abstract parallel composition $\kappa_1 \Uparrow_{\mathcal{A}} \kappa_2$ by:*

$$\kappa_1 \Uparrow_{\mathcal{A}} \kappa_2 = \mathbf{mgu}_{\mathcal{A}}(\widehat{\kappa}_1 \cup \widehat{\kappa}_2).$$

This test can be lifted in the standard way to sets of abstract substitutions. The following theorem constitutes the main result in this section and basically states that abstract basic narrowing is compositional w.r.t. the AND operator.

**Theorem 4.22** $\Delta(\Leftarrow \mathbf{g}_1, \mathbf{g}_2) = \Delta(\Leftarrow \mathbf{g}_1) \Uparrow_{\mathcal{A}} \Delta(\Leftarrow \mathbf{g}_2)$.

In the following we formalize the idea that the compositionality of the abstract semantics w.r.t the union of $\mathcal{E}$-unification problems, as formalized in Theorem 4.22, provides for incrementality when dealing with constraint satisfaction problems in the framework of constraint logic programming, where sets of constraints are incrementally added to a solver.

In the context of constraint logic programming [24, 28], incremental search consists of proving the solvability of a sequence of constraint problems by transforming the existing solution to each previously solved problem into a solution to the next problem [23].

When dealing with equational constraints [4], the tests of solvability can be extremely redundant. Termination is not even guaranteed. In [5, 6] we propose a lazy resolution procedure [24] which incorporates an analysis of unsatisfiability which allows us to avoid some useless computations. To achieve efficiency, the analyses also need to be incremental, that is, when adding a new equation set $\widetilde{\mathbf{c}}$ to an already tested set $\mathbf{c}$ of constraints, the analysis should not start checking the accumulated constraint $\mathbf{c} \cup \widetilde{\mathbf{c}}$ from scratch. In this section, we formulate an incremental algorithm for analyzing the unsatisfiability of equation sets within a constraint setting [4]. The kernel of the algorithm is the calculus of abstract basic narrowing reduction as formulated in Section 4.1.

We assume that constraints monotonically grow as long as the computation proceeds, and the question we consider is how to deal efficiently with the test of unsatisfiability for the accumulated constraints as long as new equations are added.

**Definition 4.23** Incremental constraint satisfaction problem

*Let $\mathbf{c}_0, \mathbf{c}_1, \widetilde{\mathbf{c}}_1, \ldots, \mathbf{c}_\mathbf{n}, \widetilde{\mathbf{c}}_\mathbf{n}$ be constraints, where $\mathbf{c}_\mathbf{i} = \mathbf{c}_{\mathbf{i}-1} \cup \widetilde{\mathbf{c}}_\mathbf{i}$. The incremental constraint satisfaction*

*problem consists of (efficiently) checking the (un)satisfiability of* $\mathbf{c_i}$ *by using some information from the computations of* $\mathbf{c}_0, \ldots, \mathbf{c}_{\mathbf{i}-1}$, $\mathbf{i} = 1, \ldots, \mathbf{n}$.

The idea here is to compute the abstract success set of $\Leftarrow \mathbf{c} \cup \widetilde{\mathbf{c}}$ by combining the sets $\Delta(\Leftarrow \mathbf{c})$ and $\Delta(\Leftarrow \widetilde{\mathbf{c}})$ which describe the successes of $\Leftarrow \mathbf{c}$ and $\Leftarrow \widetilde{\mathbf{c}}$, respectively.

We define an **i***ncremental* **E***quational* **A***nalyzer (***iEA***)* as follows.

**Definition 4.24** *An* **iEA***-state is a pair* $\langle \mathbf{c}, \Theta \rangle$*, where* $\mathbf{c}$ *is a constraint and* $\Theta$ *is a set of substitutions. The empty* **iEA***-state is* $\langle \emptyset, \emptyset \rangle$.

**Definition 4.25** *(***iEA** transition relation $\xrightarrow{\tilde{\mathbf{c}}}_{\mathbf{iEA}}$*)*

$$\frac{\Theta' = \Theta \Uparrow \Delta(\Leftarrow \widetilde{\mathbf{c}})}{\langle \mathbf{c}, \Theta \rangle \quad \xrightarrow{\tilde{\mathbf{c}}}_{\mathbf{iEA}} \quad \langle \mathbf{c} \cup \widetilde{\mathbf{c}}, \Theta' \rangle}$$

We note that, if the accumulated abstract success set $\Theta' = \emptyset$ then $\mathbf{c} \cup \widetilde{\mathbf{c}}$ is unsatisfiable by Corollary 4.20. Our strategy proves the unsatisfiability of $\mathbf{c} \cup \widetilde{\mathbf{c}}$, or it builds the (non-empty) abstract success set $\Delta(\Leftarrow \mathbf{c} \cup \widetilde{\mathbf{c}})$, as stated by:

**Theorem 4.26** *Let* $\mathbf{c}$ *be a constraint and* $\Theta = \Delta(\Leftarrow \mathbf{c}) \not\equiv \emptyset$*. Then,*

1. *if a transition* $\langle \mathbf{c}, \Theta \rangle \quad \xrightarrow{\tilde{\mathbf{c}}}_{\mathbf{iEA}} \quad \langle \mathbf{c} \cup \widetilde{\mathbf{c}}, \Theta' \rangle$ *is proven, then* $\Theta' = \Delta(\Leftarrow \mathbf{c} \cup \widetilde{\mathbf{c}})$*;*
2. *if a transition* $\langle \mathbf{c}, \Theta \rangle \quad \xrightarrow{\tilde{\mathbf{c}}}_{\mathbf{iEA}} \quad \langle \mathbf{c} \cup \widetilde{\mathbf{c}}, \emptyset \rangle$ *is proven, then the constraint* $\mathbf{c} \cup \widetilde{\mathbf{c}}$ *is unsatisfiable.*

We note that the computed abstract answer set $\Delta(\Leftarrow \mathbf{c} \cup \widetilde{\mathbf{c}})$ can be used to guide the final execution of a 'full' narrower which can find the concrete solutions and possibly recognize the unsatisfiability not detected by this lazy procedure, as described in [7].

# 5 Applications of Abstract Narrowing

In this part, we show two possible applications of the framework we have defined in the last section. First, we define a refined narrowing which is able to prune branches which do not lead to any solution. Secondly, we show how groundness and sharing information can be derived from the abstract success set obtained by abstract narrowing.

## 5.1 Refined Narrowing

Theorem 4.19 suggests that $\Delta(\Leftarrow \mathbf{g}')$ 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 $\Delta(\Leftarrow \mathbf{g}')$ with concrete substitutions.

**Definition 5.1** *Let $\phi, \kappa \in \mathbf{Sub}^{\perp}$. We define the relation $\phi \diamondsuit \kappa$ by:*

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

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 this section we formulate an algorithm which decides this relation.

From Theorem 4.19 we derive a stronger characterization of *successful* narrowing derivations. Actually, successful derivations are distinguishable by the following property, which characterizes our analysis as a finitely computable approximation of the collecting semantics, as formalized in Definition 3.2.

**Theorem 5.2** *Let $\mathcal{R}$ be a TRS and $\langle \Leftarrow \mathbf{g}, \epsilon \rangle$ be an initial state. Then, for every $\theta \in \mathcal{C}(\Leftarrow \mathbf{g})$, there exists $\kappa \in \Delta(\Leftarrow \mathbf{g})$ such that $\kappa \diamondsuit \theta$.*

The following result follows immediately from Theorem 5.2 and represents the basis for our optimization:

**Corollary 5.3** *Let $\mathcal{R}$ be a TRS and $\mathbf{s} = \langle \Leftarrow \mathbf{g}, \epsilon \rangle$ be an initial state. Let $\langle \Leftarrow \mathbf{g}', \theta \rangle$ be any state of the (basic) conditional narrowing tree for $\mathcal{R}$ with $\mathbf{s}$. If there is no $\kappa \in \Delta(\Leftarrow \mathbf{g})$ such that $\kappa \diamondsuit \theta$, then there is no successful basic narrowing derivation for $\langle \Leftarrow \mathbf{g}', \theta \rangle$.*

Since the subtree originating in any node which satisfies the requirements of Corollary 5.3 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. The basic idea is to prune from the narrowing search tree all branches whose root satisfies the requirements of Corollary 5.3.

Now, we refine the *basic $\mathcal{C}$onditional $\mathcal{N}$arrowing* calculus of Section 3.1 by considering the abstract substitutions which can be gathered by **a**bstract $\mathcal{C}$onditional $\mathcal{N}$arrowing, as formalized in Definition 4.16.

**Definition 5.4** refined $\mathcal{C}$onditional $\mathcal{N}$arrowing calculus *($\mathbf{r}\mathcal{CN}$)*
*Let $\mathcal{R}$ be a level-canonical TRS. Let $\mathbf{s} \in \mathbf{State}$ be an initial state. We define the $\mathbf{r}$efined $\mathcal{C}$onditional $\mathcal{N}$arrowing as a transition system $(\mathbf{State}, \leadsto_{\mathbf{r}\mathcal{CN}})$ whose transition relation $\leadsto_{\mathbf{r}\mathcal{CN}} \subseteq \mathbf{State} \times \mathbf{State}$ is the smallest relation which satisfies:*

unification rule:

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

narrowing rule:

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

*To solve the equation set $\mathbf{g}_0$, the algorithm starts with the initial state $\mathbf{s} = \langle \Leftarrow \mathbf{g}_0, \epsilon\rangle$ and tries to derive new goals until a terminal state $\langle \Leftarrow \mathbf{true}, \sigma\rangle$ is reached.*

We give an example which illustrates how the optimization works.

**Example 5** *Consider the TRS $\mathcal{R}$ from Example 4. Applying $\mathbf{r}\mathcal{CN}$ for $\mathcal{R}$ with initial state $\langle \Leftarrow \mathbf{h}(\mathbf{f}(\mathbf{Z})) = 0, \epsilon\rangle$ 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* [11]*, unification rules* [19]*, operator joinability* [18] *or eager normalization* [2, 18] *techniques.*

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

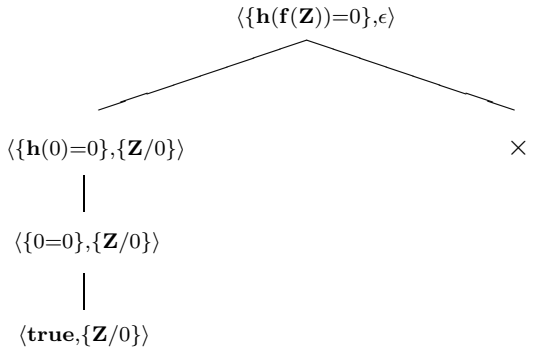


Figure 4: *Refined basic narrowing*

**Theorem 5.5** *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* $\{\sigma_{\restriction \mathbf{Var(g)}} \mid \langle \Leftarrow \mathbf{g}, \epsilon\rangle \leadsto^*_{\mathbf{r}\mathcal{CN}} \langle \Leftarrow \mathbf{true}, \sigma\rangle\}$ *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. The compatibility $\phi \diamondsuit \kappa$ of two abstract substitutions exactly corresponds to check that $\phi \Uparrow_{\mathcal{A}} \kappa$ is not **fail** as we can see in the next proposition.

**Proposition 5.6** *Let* $\phi, \kappa \in \mathbf{Sub}^{\perp}$*. Then* $\phi \diamondsuit \kappa$ iff $(\phi \Uparrow \kappa) \not\equiv \mathbf{fail}$.

## 5.2 Sharing Analysis

As formalized in Definition 4.12 and 4.16, our algorithm manages information about all possible derivations for a program and a given query, and computes a set of (abstract) substitutions which can be used to accurately keep track of some non-trivial variable sharing information regarding the groundness and the independence of goal variables in the computation. This information is useful for a number of optimizations of equational Horn programs, notably in the exploitation of independent AND-parallelism, where different subexpressions can be narrowed in parallel if they are independent, i.e. they do not share unbound variables [27, 39], and the results of the analysis can avoid doing the corresponding checks at runtime.

A variable $\mathbf{x}$ is ground under a substitution $\theta$ iff $\mathbf{Var}(\mathbf{x}\theta) = \emptyset$. A pair of variables are independent under $\theta$ iff the terms to which these variables are bound by $\theta$ do not share any variable, which follows iff $\mathbf{Var}(\mathbf{x}\theta) \cap \mathbf{Var}(\mathbf{y}\theta) = \emptyset$. Otherwise, $\mathbf{x}$ and $\mathbf{y}$ are said to be aliased by $\theta$. Groundness and

independence information can be derived from the abstract semantics for a program and a given goal as follows.

We define a function *ground*, which takes an abstract substitution $\kappa$ and returns the set of variables which are ground under $\kappa$. Formally,

$$\mathbf{ground}(\kappa) = \{\mathbf{x} \mid \mathbf{Var}([\![\mathbf{x}\kappa]\!]) = \emptyset\}$$

The function *alias* takes an abstract substitution $\kappa$ and returns a set of pairs of variables which *share* under $\kappa$, i.e. which are bound by $\kappa$ to terms which share a variable. Formally,

$$\mathbf{alias}(\kappa) = \{(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \neq \mathbf{y} \ \wedge \ \mathbf{Var}([\![\mathbf{x}\kappa]\!]) \cap \mathbf{Var}([\![\mathbf{y}\kappa]\!]) = \emptyset\}$$

Note that two variables $\mathbf{x}$ and $\mathbf{y}$ are independent under $\kappa$ iff the pair $(\mathbf{x}, \mathbf{y}) \notin \mathbf{alias}(\kappa)$.

Groundness and independence information about the variables of a given goal $\Leftarrow \mathbf{g}$ can be derived from all substitutions in $\Delta(\Leftarrow \mathbf{g})$ as described by the following sets:

$$\mathbf{SGV}(\Leftarrow \mathbf{g}) = \bigcap\nolimits_{\kappa \in \Delta(\Leftarrow \mathbf{g})} \{\mathbf{ground}(\kappa)\}$$

$$\mathbf{SIV}(\Leftarrow \mathbf{g}) = \bigcap\nolimits_{\kappa \in \Delta(\Leftarrow \mathbf{g})} \{(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \notin \mathbf{alias}(\kappa)\}$$

which specify the set of definitely ground variables and the set of definitely independent variables inferred from $\Delta(\Leftarrow \mathbf{g})$, respectively.

The following example illustrates the derivation of sharing information from the abstract semantics of a given goal.

**Example 6** *Consider the following program $\mathcal{R}$, which computes the number of elements of a list, and its abstraction $\mathcal{R}_\mathcal{A}$:*

$$\mathcal{R} = \{ \ \mathbf{noe}([\,]) = 0. \qquad \mathbf{noe}([\mathbf{H} \mid \mathbf{T}]) = \mathbf{succ}(\mathbf{noe}(\mathbf{T})). \ \}$$

$$\mathcal{R}_\mathcal{A} = \{ \ \mathbf{noe}([\,]) = 0. \qquad \mathbf{noe}([\mathbf{H} \mid \mathbf{T}]) = \mathbf{succ}(\bot). \ \}$$

*The abstract semantics for the initial goal $\Leftarrow \mathbf{g} \equiv \Leftarrow \mathbf{noe}(\mathbf{L}) = \mathbf{succ}(0), \mathbf{noe}(\mathbf{LL}) = \mathbf{X}, \mathbf{L} = [\mathbf{a} \,|\, \mathbf{LL}]$ is $\Delta(\langle \Leftarrow \mathbf{g}, \epsilon \rangle) = \{\{\mathbf{X}/0, \mathbf{L}/[\mathbf{a}], \mathbf{LL}/[\,]\}\}$. Then $\mathbf{SGV}(\Leftarrow \mathbf{g}) = \{\mathbf{X}, \mathbf{L}, \mathbf{LL}\}$. The analysis shows that, at the end of any possible computation for a given goal, all variables are ground and hence independent.*

The following proposition shows that the inference of sharing information from the abstract semantics for a given goal is safe.

**Proposition 5.7** *Let $\mathcal{R}$ be a TRS and let $\Leftarrow \mathbf{g}$ be a goal.*

1. *If $\mathbf{x} \in \mathbf{SGV}(\Leftarrow \mathbf{g})$ then any successful derivation for $\mathcal{R}$ with $\Leftarrow \mathbf{g}$ grounds $\mathbf{x}$.*

2. *If $(\mathbf{x}, \mathbf{y}) \in \mathbf{SIV}(\Leftarrow \mathbf{g})$ then there is no solution for $\Leftarrow \mathbf{g}$ in $\mathcal{R}$ under which $\mathbf{x}$ and $\mathbf{y}$ share.*

# 6 Conclusion and further research

We have presented a formal compositional semantics for the success set of equational logic programs which is suitable for AND-parallel implementations.

We have then shown that this semantics leads to compositional analysis and have given an example of analysis of unsatisfiability which is suitable for theories where equations are considered as constraints. Some benchmarks can be found in [8].

The approach which we have taken is of general interest. In particular it applies to any kind of analysis where we look for properties which are satisfied by all (or some) success paths. For specific analyses, it will be necessary to provide the appropriate abstract domains and approximation of the term rewriting system.

We have also 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.

The optimization can be used to prune useless paths in the search tree for a given set of equations. It may as well be used for constraint languages like CLP($\mathcal{H}/\mathcal{E}$) [3, 4] which integrates equational and logic programming as an instance of the CLP scheme [28]. A CLP($\mathcal{H}/\mathcal{E}$) program $\mathcal{P} \cup \mathcal{E}$ is a CLP program $\mathcal{P}$ augmented by a level-canonical Horn equational theory $\mathcal{E}$. The equations to be solved with respect to the equational theory are considered as constraints. For a computational step in this framework, it is essential to be able to (semi-)decide if a set of equations is satisfiable. We can use our refined conditional narrowing to evaluate the satisfiability of a set of equations.

A prototype system based on the ideas described in this work has been implemented. In spite of

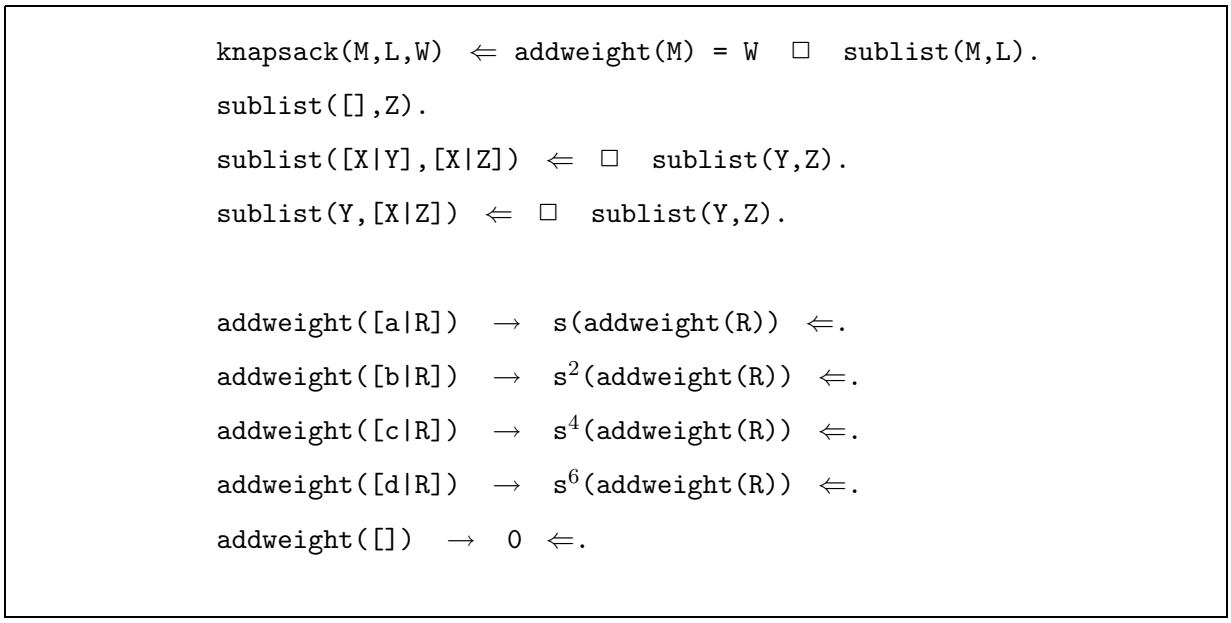


Figure 5: A CLP($\mathcal{H}/\mathcal{E}$) program. `knapsack(M,L,W)` states that the items in the sublist `M` of the list `L` can be packed into a knapsack such that it weights exactly `W`.

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 [4, 22, 25] 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 CLP($\mathcal{H}/\mathcal{E}$) program knapsack (Figure 5). In these tests we are able to avoid up to 85% of unnecessary narrowing attempts, with no prohibitive time overhead due to **a**bstract $\mathcal{C}$onditional $\mathcal{N}$arrowing (column *abstract*). Although more experimentation is needed, our experiments indicate that the strategy can be a useful tool in the optimization of equational logic programs.

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 to improve our method by strengthening the construction of the dependency graph. This seems a difficult task if we want to maintain goal independence. Another topic of further research concerns the parallel implementation of the consistency check.

| constraint | solution | basic | refined | abstract |
|---|---|---|---|---|
| `addweight(X) = 0.` | X = nil | 1.32 | 0.26 | 0.26 |
| | termination | $\infty$ | 0 | |
| `addweight(X) = s`$^2$`(0).` | X = [b] | 4.34 | 3.82 | 0.32 |
| | X = [a,a] | 9.58 | 5.46 | |
| | termination | $\infty$ | $\infty$ | |
| `addweight(X) = s`$^3$`(0).` | X = [a,b] | 11.56 | 7.58 | 0.34 |
| | X = [b,a] | 17.64 | 13.76 | |
| | X = [a,a,a] | 46.82 | 22.68 | |
| | termination | $\infty$ | $\infty$ | |
| `addweight(X) = 0,` `addweight([a\|X]) = Y.` | X = nil, Y = s(0) | 16.68 | 1.22 | 0.82 |
| | termination | $\infty$ | 0 | |

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

# 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 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 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*.

[5] 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.

[6] 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.

[7] M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. Narrowing Aproximations as an optimization for Equational Logic Programs. In M. Bruynooghe and J. Penjam, editors, *Proc. of PLILP'93*, volume 714 of *Lecture Notes in Computer Science*, pages 391–409. Springer-Verlag, Berlin, 1993.

[8] M. Alpuente, M. Falaschi, and G. Vidal. Incremental Equational Constraint Analyses. In V. Dahl and D. Miller, editors, *Proc. of ILPS'93*. MIT Press, Cambridge, Mass., 1993. Poster.

[9] M. Alpuente, M. Falaschi, and G. Vidal. Semantics-Based Compositional Analysis for Equational Horn Programs. Technical Report DSIC-II/5/93, UPV, 1993. Submitted for publication.

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

[11] 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.

[12] 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.

[13] 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.

[14] M. Codish, M. Falaschi, and K. Marriott. Suspension Analyses for Concurrent Logic Programs. Technical Report TR 12/92, Dipartimento di Informatica, Università di Pisa, May 1992. Submitted for publication.

[15] 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.

[16] N. Dershowitz and N. Lindenstrauss. An Abstract Concurrent Machine for Rewriting. In H. Kirchner and W. Wechler, editors, *Proc. Second Int'l Conf. on Algebraic and Logic Programming*, volume 463 of *Lecture Notes in Computer Science*, pages 318–331. Springer-Verlag, Berlin, 1990.

[17] N. Dershowitz and N. Lindenstrauss. A Parallel Implementation of Equational Programming. In *Proc.* $5^{\mathbf{th}}$ *Jerusalem Conf. on Information Technology*, pages 426–435. IEEE Comp. Soc. Press, 1990.

[18] 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.

[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] M. Gabbrielli and G. Levi. On the Semantics of Logic Programs. In J. Leach Albert, B. Monien, and M. Rodriguez Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium*, volume 510 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, Berlin, 1991.

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

[22] 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.

[23] P. Van Hentenryck and T. Le Provost. Incremental Search in Constraint Logic Programming. *New Generation Computing*, 9:257–275, 1991.

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

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

[26] 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.

[27] D. Jacobs and A. Langen. Accurate and Efficient Approximation of Variable Aliasing in Logic Programs. In E. Lusk and R. Overbeck, editors, *Proc. North American Conf. on Logic Programming'89*, pages 154–165. The MIT Press, Cambridge, Mass., 1989.

[28] 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.

[29] J. Jaffar, J.-L. Lassez, and M.J. Maher. A theory of complete logic programs with equality. *Journal of Logic Programming*, 3:211–223, 1984.

[30] 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.

[31] 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.

[32] H. Kuchen and W. Hans. An AND-Parallel Implementation of the Functional Logic Language Babel. In *Aachener Informatik-Bericht*, volume 12, pages 119–139, RWTH Aachen, 1991.

[33] H. Kuchen, J.J. Moreno-Navarro, and M. Hermenegildo. Independent AND-Parallel Implementation of Narrowing. In M. Bruynooghe and M. Wirsing, editors, *Proc. of PLILP'92, Leuven (Belgium)*, volume 631 of *Lecture Notes in Computer Science*, pages 24–38. Springer-Verlag, Berlin, 1992.

[34] 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.

[35] 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.

[36] M. J. Maher. On parameterized substitutions. Technical Report RC 16042, IBM - T.J. Watson Research Center, Yorktown Heights, NY, 1990.

[37] A. Middeldorp. Modular Properties of Conditional Term Rewriting Systems. Technical Report CS-R9105, Centre for Mathematics and Computer Science, Amsterdam, 1991. To appear in Information and Computation.

[38] 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.

[39] K. Mutkumar and M. Hermenegildo. Determination of Variable Dependence Information through Abstract Interpretation. In E. Lusk and R. Overbeck, editors, *Proc. North American Conf. on Logic Programming'89*, pages 166–185. The MIT Press, Cambridge, Mass., 1989.

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

[41] 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.

[42] 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.

[43] 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.

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

# Appendix

PROOF. [proof of Proposition 3.5]

1. ($\Rightarrow$) Let $\vartheta \in \mathcal{U}_{\mathcal{E}}(\mathbf{E}_1 \cup \mathbf{E}_2)$. Then $\vartheta \in \mathcal{U}_{\mathcal{E}}(\mathbf{E}_1) = \Theta_1$ and $\vartheta \in \mathcal{U}_{\mathcal{E}}(\mathbf{E}_2) = \Theta_2$. Thus $\vartheta \in \vartheta \Uparrow_{\mathcal{E}} \vartheta \subseteq \Theta_1 \Uparrow_{\mathcal{E}} \Theta_2$.

2. ($\Leftarrow$) Let $\theta \in \Theta_1 \Uparrow_{\mathcal{E}} \Theta_2$. Then, by Definition 3.4, $\exists \theta_1 \in \Theta_1, \exists \theta_2 \in \Theta_2$ such that $\theta \in \mathcal{U}_{\mathcal{E}}(\widehat{\theta_1} \cup \widehat{\theta_2})$. Since

   $$\theta \in \mathcal{U}_{\mathcal{E}}(\widehat{\theta_1}) \Rightarrow \theta \in \mathcal{U}_{\mathcal{E}}(\mathbf{E}_1)$$
   $$\theta \in \mathcal{U}_{\mathcal{E}}(\widehat{\theta_2}) \Rightarrow \theta \in \mathcal{U}_{\mathcal{E}}(\mathbf{E}_2)$$

   then $\theta \in \mathcal{U}_{\mathcal{E}}(\mathbf{E}_1 \cup \mathbf{E}_2)$.

□

PROOF. [proof of Theorem 3.8]

1. ($\Leftarrow$) Let $\theta \in \mathcal{S}'(\Leftarrow \mathbf{g}_1) \Uparrow \mathcal{S}'(\Leftarrow \mathbf{g}_2)$. From the definition of parallel composition, there exists $\theta_1 \in \mathcal{S}'(\Leftarrow \mathbf{g}_1)$ and $\theta_2 \in \mathcal{S}'(\Leftarrow \mathbf{g}_2)$ such that $\theta = \theta_1 \Uparrow \theta_2$. From Definition 3.7, there exist $\theta_{11}, \theta_{12}, \ldots, \theta_{1\mathbf{n}_1}$, $\mathbf{n}_1 \geq 1$ and $\theta_{21}, \theta_{22}, \ldots, \theta_{2\mathbf{n}_2}$, $\mathbf{n}_2 \geq 1$ such that $\Leftarrow \mathbf{g}_1 \stackrel{\theta_{11}}{\longmapsto} \Leftarrow \mathbf{g}'_1 \stackrel{\theta_{12}}{\longmapsto} \ldots \stackrel{\theta_{1\mathbf{n}_1}}{\longmapsto} \Leftarrow \mathbf{true}$ and $\theta_1 = \theta_{11} \Uparrow \ldots \Uparrow \theta_{1\mathbf{n}_1}$, and $\Leftarrow \mathbf{g}_2 \stackrel{\theta_{21}}{\longmapsto} \Leftarrow \mathbf{g}'_2 \stackrel{\theta_{22}}{\longmapsto} \ldots \stackrel{\theta_{2\mathbf{n}_2}}{\longmapsto} \Leftarrow \mathbf{true}$ and $\theta_2 = \theta_{21} \Uparrow \ldots \Uparrow \theta_{2\mathbf{n}_2}$. To simplify the proof, we suppose $\mathbf{n}_1 = \mathbf{n}_2 = \mathbf{n}$ (we assume as many transitions $\Leftarrow \mathbf{true} \stackrel{\epsilon}{\longmapsto} \Leftarrow \mathbf{true}$ as needed). We prove the assertion by induction on the length $\mathbf{n}$ of the derivation.

   If $\mathbf{n} = 1$, then $\Leftarrow \mathbf{g}_1 \stackrel{\theta_{11}}{\longmapsto} \Leftarrow \mathbf{true}$ and $\Leftarrow \mathbf{g}_2 \stackrel{\theta_{21}}{\longmapsto} \Leftarrow \mathbf{true}$. Hence, from Definition 3.6, $\Leftarrow \mathbf{g}_1, \mathbf{g}_2 \stackrel{\theta_{11} \Uparrow \theta_{21}}{\longmapsto} \Leftarrow \mathbf{true}, \mathbf{true}$. Therefore, the goal is solved and $\theta_{11} \Uparrow \theta_{21} = \theta_1 \Uparrow \theta_2 \in \mathcal{S}'(\Leftarrow \mathbf{g}_1, \mathbf{g}_2)$.

   Let us consider the inductive case. Let $\theta'_1 = \theta_{12} \Uparrow \ldots \Uparrow \theta_{1\mathbf{n}}$ and $\theta'_2 = \theta_{22} \Uparrow \ldots \Uparrow \theta_{2\mathbf{n}}$. Then $\theta'_1 \Uparrow \theta'_2 \in \mathcal{S}'(\Leftarrow \mathbf{g}'_1) \Uparrow \mathcal{S}'(\Leftarrow \mathbf{g}'_2)$ and, by the inductive hypothesis, $\theta'_1 \Uparrow \theta'_2 \in \mathcal{S}'(\Leftarrow \mathbf{g}'_1, \mathbf{g}'_2)$. From Definition 3.6, $\Leftarrow \mathbf{g}_1, \mathbf{g}_2 \stackrel{\theta_{11} \Uparrow \theta_{21}}{\longmapsto} \Leftarrow \mathbf{g}'_1, \mathbf{g}'_2$ and, by the commutativity and the associativity of the operator $\Uparrow$ of parallel composition, $\theta_1 \Uparrow \theta_2 = \theta_{11} \Uparrow \theta_{21} \Uparrow \theta'_1 \Uparrow \theta'_2 \in \mathcal{S}'(\Leftarrow \mathbf{g}_1, \mathbf{g}_2)$.

2. ($\Rightarrow$) Let $\theta \in \mathcal{S}'(\Leftarrow \mathbf{g}_1, \mathbf{g}_2)$. From Definition 3.7, there exist $\theta_1, \ldots, \theta_{\mathbf{n}}$ such that $\Leftarrow \mathbf{g}_1, \mathbf{g}_2 \stackrel{\theta_1}{\longmapsto} \ldots \stackrel{\theta_{\mathbf{n}}}{\longmapsto} \Leftarrow \mathbf{true}$ and $\theta = \theta_1 \Uparrow \ldots \Uparrow \theta_{\mathbf{n}}$. We prove the assertion by induction on the length $\mathbf{n}$ of the derivation.

   Let $\mathbf{n} = 1$ and assume $\mathbf{g}_1 \cup \mathbf{g}_2 = \bigcup_{\mathbf{k}=1,\ldots,\mathbf{m}} \{\mathbf{e_k}\}$, $\mathbf{g}_1 = \bigcup_{\mathbf{i}=1,\ldots,\mathbf{k}_1} \{\mathbf{e_i}\}$, and $\mathbf{g}_2 = \bigcup_{\mathbf{j}=\mathbf{k}_1+1,\ldots,\mathbf{m}} \{\mathbf{e_j}\}$. If $\Leftarrow \mathbf{g}_1, \mathbf{g}_2 \stackrel{\theta_1}{\longmapsto} \Leftarrow \mathbf{true}$ then, from Definition 3.7, for all $\mathbf{k} = 1, \ldots, \mathbf{m}$, $\Leftarrow \{\mathbf{e_k}\} \stackrel{\vartheta_{\mathbf{k}}}{\longmapsto} \Leftarrow \mathbf{true}$, and $\theta_1 = \vartheta_1 \Uparrow \ldots \Uparrow \vartheta_{\mathbf{m}}$. Since $\Leftarrow \{\mathbf{e_i}\} \stackrel{\vartheta_{\mathbf{i}}}{\longmapsto} \Leftarrow \mathbf{true}$, $\mathbf{i} = 1, \ldots, \mathbf{k}_1$ and $\Leftarrow \{\mathbf{e_j}\} \stackrel{\vartheta_{\mathbf{j}}}{\longmapsto} \Leftarrow \mathbf{true}$, $\mathbf{j} = \mathbf{k}_1 + 1, \ldots, \mathbf{m}$, then $\vartheta_1 \Uparrow \ldots \Uparrow \vartheta_{\mathbf{k}_1} \in \mathcal{S}'(\Leftarrow \mathbf{g}_1)$ and $\vartheta_{\mathbf{k}_1+1} \Uparrow \ldots \Uparrow \vartheta_{\mathbf{m}} \in \mathcal{S}'(\Leftarrow \mathbf{g}_2)$. Therefore, $\theta_1 = \vartheta_1 \Uparrow \ldots \Uparrow \vartheta_{\mathbf{m}} \in \mathcal{S}'(\Leftarrow \mathbf{g}_1) \Uparrow \mathcal{S}'(\Leftarrow \mathbf{g}_2)$.

   Let us consider the inductive case. Let consider $\theta_{11}, \theta_{12}$ such that $\Leftarrow \mathbf{g}_1 \stackrel{\theta_{11}}{\longmapsto} \Leftarrow \mathbf{g}'_1$, $\Leftarrow \mathbf{g}_2 \stackrel{\theta_{12}}{\longmapsto} \Leftarrow \mathbf{g}'_2$, $\theta_1 = \theta_{11} \Uparrow \theta_{12}$, and $\Leftarrow \mathbf{g}_1, \mathbf{g}_2 \stackrel{\theta_{11} \Uparrow \theta_{12}}{\longmapsto} \Leftarrow \mathbf{g}'_1, \mathbf{g}'_2 \stackrel{\theta_2}{\longmapsto} \ldots \stackrel{\theta_{\mathbf{n}}}{\longmapsto} \Leftarrow \mathbf{true}$. Then, by the

inductive hypothesis, $\theta_2 \Uparrow \ldots \Uparrow \theta_{\mathbf{n}} \in \mathcal{S}'(\Leftarrow \mathbf{g}'_1) \Uparrow \mathcal{S}'(\Leftarrow \mathbf{g}'_2)$. From the definition of parallel composition, there exists $\theta'_1 \in \mathcal{S}'(\Leftarrow \mathbf{g}_1)$ and $\theta'_2 \in \mathcal{S}'(\Leftarrow \mathbf{g}_2)$ such that $\theta_2 \Uparrow \ldots \Uparrow \theta_{\mathbf{n}} = \theta'_1 \Uparrow \theta'_2$. Hence, from Definition 3.6, $\theta_{11} \Uparrow \theta'_1 \in \mathcal{S}'(\Leftarrow \mathbf{g}_1)$ and $\theta_{12} \Uparrow \theta'_2 \in \mathcal{S}'(\Leftarrow \mathbf{g}_2)$. By the commutativity and associativity of the operator $\Uparrow$ of parallel composition, $\theta_{11} \Uparrow \theta'_1 \Uparrow \theta_{12} \Uparrow \theta'_2 = \theta_{11} \Uparrow \theta_{12} \Uparrow \theta'_1 \Uparrow \theta'_2 = \theta_{11} \Uparrow \theta_{12} \Uparrow \theta_2 \Uparrow \ldots \Uparrow \theta_{\mathbf{n}} = \theta_1 \Uparrow \theta_2 \Uparrow \ldots \Uparrow \theta_{\mathbf{n}} \in \mathcal{S}'(\Leftarrow \mathbf{g}_1) \Uparrow \mathcal{S}'(\Leftarrow \mathbf{g}_2)$.

□

PROOF. [proof of Corollary 3.10]

1. ($\Leftarrow$) We prove the more general assertion that, for every compositional conditional narrowing sequence $\Leftarrow \mathbf{g}_0 \overset{\theta_1}{\longmapsto} \ldots \overset{\theta_\mathbf{i}}{\longmapsto} \Leftarrow \mathbf{g_i} \overset{\theta_{\mathbf{i}+1}}{\longmapsto} \ldots \overset{\theta_\mathbf{n}}{\longmapsto} \Leftarrow \mathbf{g_n}$, there exists a corresponding basic conditional narrowing sequence $\langle \Leftarrow \mathbf{g}'_0, \vartheta_0\rangle \rightsquigarrow \ldots \rightsquigarrow \langle \Leftarrow \mathbf{g}'_{\mathbf{m_i}}, \vartheta_{\mathbf{m_i}}\rangle \rightsquigarrow \ldots \rightsquigarrow \langle \Leftarrow \mathbf{g}'_{\mathbf{m_n}}, \vartheta_{\mathbf{m_n}}\rangle$ such that $\mathbf{g}'_{\mathbf{m_n}} = \mathbf{g_n}$, $\vartheta_{\mathbf{m_n}} = \theta_1 \Uparrow \ldots \Uparrow \theta_\mathbf{n}$ and $\vartheta_{\mathbf{m_n}} \not\equiv \mathbf{fail}$. The proof is done by induction on the length $\mathbf{n}$ of the former derivation. For the sake of simplicity, we assume that the clause renaming chosen for basic conditional narrowing reduction is the same as in compositional narrowing reduction.

   Let $\mathbf{n} = 1$ and assume $\mathbf{g}_0 = \bigcup_{\mathbf{i}=1,\ldots,\mathbf{k}} \{\mathbf{e_i}\}$, $\mathbf{k} \geq 1$. If $\Leftarrow \mathbf{g}_0 \overset{\theta_1}{\longmapsto} \Leftarrow \mathbf{g}_1$ then, by rules (1) and (2) of Definition 3.6, $\mathbf{g}_1 = \bigcup_{\mathbf{i}=1,\ldots,\mathbf{k}} \{\mathbf{e_i}[\rho_\mathbf{i}]_{\mathbf{u_i}}\} \cup \widetilde{\mathbf{e}}_\mathbf{i}$, where $\mathbf{u_i} \in \bar{\mathbf{O}}(\mathbf{e_i})$, $(\lambda_\mathbf{i} \to \rho_\mathbf{i} \Leftarrow \widetilde{\mathbf{e}}_\mathbf{i}) \ll \mathcal{R}$ and $\sigma_\mathbf{i} = \mathbf{mgu}(\{\mathbf{e}_{\mathbf{i}|\mathbf{u_i}} = \lambda_\mathbf{i}\})$, and $\theta_1 = \sigma_1 \Uparrow \ldots \Uparrow \sigma_\mathbf{k}$. If $\theta_1 \not\equiv \mathbf{fail}$ then, by the definition of basic conditional narrowing, there exists $\langle \Leftarrow \mathbf{g}_0, \epsilon\rangle = \langle \Leftarrow \mathbf{g}'_0, \vartheta_0\rangle \rightsquigarrow \langle \Leftarrow \mathbf{g}'_1, \vartheta_1\rangle \rightsquigarrow \ldots \rightsquigarrow \langle \Leftarrow \mathbf{g}'_\mathbf{k}, \vartheta_\mathbf{k}\rangle$ such that, for all $\mathbf{i} = 1, \ldots, \mathbf{k}$, $\mathbf{g}'_\mathbf{i} = (\mathbf{g}'_{\mathbf{i}-1} \sim \{\mathbf{e_i}\}) \cup \{\mathbf{e_i}[\rho_\mathbf{i}]_{\mathbf{u_i}}\} \cup \widetilde{\mathbf{e}}_\mathbf{i}$, $\vartheta_\mathbf{i} = \vartheta_{\mathbf{i}-1}\sigma'_\mathbf{i}$ and $\sigma'_\mathbf{i} = \mathbf{mgu}(\{(\mathbf{e}_{\mathbf{i}|\mathbf{u_i}})\vartheta_{\mathbf{i}-1} = \lambda_\mathbf{i}\}) \not\equiv \mathbf{fail}$. Hence $\mathbf{g}'_\mathbf{k} = \mathbf{g}_1$ and, since $\vartheta_0 = \epsilon$ then $\sigma'_1 = \sigma_1$, and thus $\vartheta_\mathbf{k} = \sigma_1\sigma'_2 \ldots \sigma'_\mathbf{k} = \sigma_1 \Uparrow \ldots \Uparrow \sigma_\mathbf{k} = \theta_1$, by Lemma 3.9.

   Let us consider the inductive case. If $\mathbf{n} \geq 1$ then $\Leftarrow \mathbf{g}_0 \overset{\theta_1}{\longmapsto} \ldots \overset{\theta_{\mathbf{n}-1}}{\longmapsto} \Leftarrow \mathbf{g_{n-1}} \overset{\theta_\mathbf{n}}{\longmapsto} \Leftarrow \mathbf{g_n}$ and, by the inductive hypothesis, there exists $\langle \Leftarrow \mathbf{g}'_0, \epsilon\rangle \rightsquigarrow \ldots \rightsquigarrow \langle \Leftarrow \mathbf{g}'_{\mathbf{m_{n-1}}}, \vartheta\rangle$ such that $\mathbf{g}'_{\mathbf{m_{n-1}}} = \mathbf{g_{n-1}}$, $\vartheta = \theta_1 \Uparrow \ldots \Uparrow \theta_{\mathbf{n}-1}$ and $\vartheta \not\equiv \mathbf{fail}$. Let assume $\mathbf{g_{n-1}} = \bigcup_{\mathbf{i}=1,\ldots,\mathbf{k}} \{\mathbf{e_i}\}$, $\mathbf{k} \geq 1$. If $\Leftarrow \mathbf{g_{n-1}} \overset{\theta_\mathbf{n}}{\longmapsto} \Leftarrow \mathbf{g_n}$ then, by rules (1) and (2) of Definition 3.6, $\mathbf{g_n} = \bigcup_{\mathbf{i}=1,\ldots,\mathbf{k}} \{\mathbf{e_i}[\rho_\mathbf{i}]_{\mathbf{u_i}}\} \cup \widetilde{\mathbf{e}}_\mathbf{i}$, where $\mathbf{u_i} \in \bar{\mathbf{O}}(\mathbf{e_i})$, $(\lambda_\mathbf{i} \to \rho_\mathbf{i} \Leftarrow \widetilde{\mathbf{e}}_\mathbf{i}) \ll \mathcal{R}$ and $\sigma_\mathbf{i} = \mathbf{mgu}(\{\mathbf{e}_{\mathbf{i}|\mathbf{u_i}} = \lambda_\mathbf{i}\})$, and $\theta_\mathbf{n} = \sigma_1 \Uparrow \ldots \Uparrow \sigma_\mathbf{k}$. If $\theta_\mathbf{n} \not\equiv \mathbf{fail}$ then, by an argument similar to the basic case, $\langle \Leftarrow \mathbf{g}'_{\mathbf{m_{n-1}}}, \vartheta\rangle \rightsquigarrow \ldots \rightsquigarrow \langle \Leftarrow \mathbf{g}'_{\mathbf{m_{n-1}+k}}, \theta\rangle = \langle \Leftarrow \mathbf{g}'_{\mathbf{m_n}}, \theta\rangle$, $\mathbf{g}'_{\mathbf{m_n}} = \mathbf{g_n}$, $\theta = \vartheta \Uparrow \sigma_1 \Uparrow \ldots \Uparrow \sigma_\mathbf{k} = \theta_1 \Uparrow \ldots \Uparrow \theta_{\mathbf{n}-1} \Uparrow \theta_\mathbf{n}$, and $\theta \not\equiv \mathbf{fail}$.

2. ($\Rightarrow$) Now we prove that, for every successful basic narrowing sequence $\langle \Leftarrow \mathbf{g}'_0, \epsilon\rangle \rightsquigarrow \langle \Leftarrow \mathbf{g}'_1, \vartheta_1\rangle \rightsquigarrow \ldots \rightsquigarrow \langle \Leftarrow \mathbf{g}'_\mathbf{m}, \vartheta_\mathbf{m}\rangle = \langle \Leftarrow \mathbf{true}, \vartheta_\mathbf{m}\rangle$ there exists a corresponding successful compositional conditional narrowing sequence $\Leftarrow \mathbf{g}_0 = \mathbf{g}'_0 \overset{\theta_1}{\longmapsto} \ldots \overset{\theta_{\mathbf{n_m}}}{\longmapsto} \Leftarrow \mathbf{true}$ such that $\vartheta_\mathbf{m} = \theta_1 \Uparrow \ldots \Uparrow \theta_{\mathbf{n_m}}$ and $\vartheta_\mathbf{m} \not\equiv \mathbf{fail}$. The proof is done by induction on the length $\mathbf{m}$ of the

former derivation. We make the same assumption about clauses renaming than before.

Let $\mathbf{m} = 1$. If $\langle \Leftarrow \mathbf{g}'_0, \epsilon \rangle \rightsquigarrow \langle \Leftarrow \mathbf{true}, \vartheta_1 \rangle$, then the rule $\mathbf{x} = \mathbf{x} \rightarrow \mathbf{true}$ has been applied, and there exists an equation $\mathbf{e}$ in $\mathbf{g}'_0$ such that $\mathbf{g}'_0 = \{\mathbf{e}\}$ and $\vartheta_1 = \mathbf{mgu}(\{\mathbf{e}\})$. Let consider $\mathbf{g}_0 = \mathbf{g}'_0 = \{\mathbf{e}\}$. By rule (1) of Definition 3.6, $\Leftarrow \mathbf{g}_0 = \Leftarrow \{\mathbf{e}\} \stackrel{\sigma}{\longmapsto} \Leftarrow \mathbf{true}$, with $\sigma = \mathbf{mgu}(\{\mathbf{e}\}) = \vartheta_1$.

Now we consider the inductive case. Let asume $\mathbf{g}_0 = \mathbf{g}'_0 = \{\mathbf{e}\} \cup \mathbf{g}$. If $\mathbf{m} \geq 1$ and $\langle \Leftarrow \mathbf{g}'_0, \epsilon \rangle \rightsquigarrow \ldots \rightsquigarrow \langle \Leftarrow \mathbf{g}'_\mathbf{m}, \vartheta_\mathbf{m} \rangle = \langle \Leftarrow \mathbf{true}, \vartheta_\mathbf{m} \rangle$, then it is immediate to prove that there exist $\langle \Leftarrow \{\mathbf{e}\}, \epsilon \rangle \rightsquigarrow \ldots \rightsquigarrow \langle \Leftarrow \mathbf{true}, \upsilon_\mathbf{i} \rangle$ and $\langle \Leftarrow \mathbf{g}, \epsilon \rangle \rightsquigarrow \ldots \rightsquigarrow \langle \Leftarrow \mathbf{true}, \upsilon'_\mathbf{k} \rangle$ such that $\mathbf{i}, \mathbf{k}$ in $\{1, \ldots, \mathbf{m} - 1\}$ and $\vartheta_\mathbf{m} = \upsilon_\mathbf{i} \Uparrow \upsilon'_\mathbf{k}$, $\vartheta_\mathbf{m} \not\equiv \mathbf{fail}$. By the inductive hypothesis, there exist $\Leftarrow \{\mathbf{e}\} \stackrel{\eta_1}{\longmapsto} \ldots \stackrel{\eta_{\mathbf{n_i}}}{\longmapsto} \Leftarrow \mathbf{true}$ and $\Leftarrow \mathbf{g} \stackrel{\eta'_1}{\longmapsto} \ldots \stackrel{\eta'_{\mathbf{n_k}}}{\longmapsto} \Leftarrow \mathbf{true}$ such that $\upsilon_\mathbf{i} = \eta_1 \Uparrow \ldots \Uparrow \eta_{\mathbf{n_i}}$ and $\upsilon'_\mathbf{k} = \eta'_1 \Uparrow \ldots \Uparrow \eta'_{\mathbf{n_k}}$. From Theorem 3.8, $\vartheta_\mathbf{m} = \upsilon_\mathbf{i} \Uparrow \upsilon'_\mathbf{k} \in \mathcal{S}'(\Leftarrow \{\mathbf{e}\}) \Uparrow \mathcal{S}'(\Leftarrow \mathbf{g}) = \mathcal{S}'(\Leftarrow \{\mathbf{e}\} \cup \mathbf{g}) = \mathcal{S}'(\Leftarrow \mathbf{g}_0)$. Hence, there exists $\theta_1, \ldots, \theta_{\mathbf{n_m}}$ such that $\Leftarrow \mathbf{g}_0 \stackrel{\theta_1}{\longmapsto} \ldots \stackrel{\theta_{\mathbf{nm}}}{\longmapsto} \Leftarrow \mathbf{true}$ and $\vartheta_\mathbf{m} = \theta_1 \Uparrow \ldots \Uparrow \theta_{\mathbf{n_m}}$, $\vartheta_\mathbf{m} \not\equiv \mathbf{fail}$.

□

PROOF. [proof of Corollary 3.11] Immediate from Corollary 3.10. □

PROOF. [proof of Proposition 4.9] Immediate from Definition 4.8. □

PROOF. [proof of Proposition 4.18] The proof is based on a well-founded complexity measure on goals which is decreased by narrowing steps. Consider the infinite derivation:

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

and assume, by contradiction, that $\mathcal{I}_{\mathcal{R}}$ has no cycle associated with any of the terms occurring in $\mathcal{D}$. For $\mathbf{t} \in \mathcal{T}$, let $\mathbf{m}(\mathbf{t})$ denote the maximal length of the paths in $\mathcal{I}_{\mathcal{R}}$ starting from any node $\mathbf{t}'$ in the left-hand side of an arrow $\stackrel{\mathcal{R}}{\rightarrow}$ such that $\mathbf{t}'$ unifies with $\widehat{\mathbf{t}}$. We associate with each $\langle \Leftarrow \mathbf{g_i}, \theta_\mathbf{i} \rangle$ in the derivation a set $\mathcal{H}_\mathbf{i} = \{\langle \mathbf{e}, \mathbf{u}, \mathbf{n} \rangle \mid \mathbf{e} \in \mathbf{g_i}, \mathbf{u} \in \bar{\mathbf{O}}(\mathbf{e}),\ \mathbf{n} = \mathbf{m}(\mathbf{e}_{|\mathbf{u}}\theta_\mathbf{i})\}$, $\mathbf{i} \geq 0$. We define the complexity $\mathcal{M}_\mathbf{i}$ of the goal $\langle \Leftarrow \mathbf{g_i}, \theta_\mathbf{i} \rangle$ as the finite multiset of natural numbers consisting of the third components of the triples in the set $\mathcal{H}_\mathbf{i}$. Since $\mathcal{I}_{\mathcal{R}}$ has no cycle associated with any term occurring in $\mathcal{D}$, these third components are finite for all $\mathbf{i} \geq 0$.

Let us define a well-founded total ordering $<_{\mathbf{mul}}$ over multiset complexities by extending the well-founded ordering $<$ on $\mathbf{N}$ to the set $\mathbf{M}(\mathbf{N})$ of finite multisets over $\mathbf{N}$. The set $\mathbf{M}(\mathbf{N})$ is well-founded under the ordering $<_{\mathbf{mul}}$ since $\mathbf{N}$ is well-founded under $<$ [15, 31]. Let $\mathcal{M}, \mathcal{M}'$ be multiset complexities. $\mathcal{M} <_{\mathbf{mul}} \mathcal{M}' \leftrightarrow \exists \mathbf{X} \subseteq \mathcal{M}, \mathbf{X}' \subseteq \mathcal{M}'$ such that $\mathcal{M} = (\mathcal{M}' \sim \mathbf{X}') \cup \mathbf{X}$ and $\forall \mathbf{n} \in \mathbf{X}\ \exists \mathbf{n}' \in \mathbf{X}'.\ \mathbf{n} < \mathbf{n}'$.

From the definition of basic conditional narrowing, at each derivation step $\langle \Leftarrow \mathbf{g_i}, \theta_\mathbf{i} \rangle \leadsto_{\mathbf{bCN}} \langle \Leftarrow \mathbf{g_{i+1}}, \theta_{\mathbf{i}+1} \rangle$, the selected occurrence either comes from the nonvariable occurrences of the equations in $\mathbf{g}_0$ or it comes from the right-hand side or from the condition of a rewrite rule used in a previous step. Let $\mathbf{e}$ and $\mathbf{u}$ denote, respectively, the selected equation and occurrence in $\mathbf{g_i}$. By definition of $\mathcal{I}_{\mathcal{R}}$, for all $\mathbf{i}$, $\mathbf{i} \geq 0$, $\forall \langle \mathbf{e}', \mathbf{u}', \mathbf{n}' \rangle \in (\mathcal{H}_{\mathbf{i}+1} \sim \mathcal{H}_\mathbf{i})$. $\mathbf{n}' < \mathbf{m}(\mathbf{e}_{|\mathbf{u}}\theta_\mathbf{i})$. Hence, by the definition of the ordering $<_{\mathbf{mul}}$, it is immediate that for all $\mathbf{i}$, $\mathbf{i} \geq 0$, $\mathcal{M}_{\mathbf{i}+1} <_{\mathbf{mul}} \mathcal{M}_\mathbf{i}$. From the definition of conditional basic narrowing, if there is a $\mathbf{k} > 0$ such that every element in $\mathcal{M}_\mathbf{k}$ is 0 then no more narrowing steps are possible. Thus, by the well-foundness of the multiset ordering $<_{\mathbf{mul}}$ over $\mathbf{M}(\mathbf{N})$, only finite decreasing chains are possible and the narrowing derivation terminates. □

PROOF. [proof of Theorem 4.19]

This theorem can easily be proved as a consequence of Lemma 4.13. We prove the theorem by induction on the length $\mathbf{n}$ of the derivation.

If $\mathbf{n} = 1$ then $\langle \Leftarrow \mathbf{g}, \epsilon \rangle \leadsto_{\mathbf{bCN}} \langle \Leftarrow \mathbf{true}, \theta \rangle$. Since $\mathbf{g}' \propto \mathbf{g}$, by Lemma 4.13, there exists an abstract basic narrowing reduction from $\langle \Leftarrow \mathbf{g}', \epsilon \rangle$ to some $\langle \Leftarrow \mathbf{g}'', \kappa \rangle$ such that $\langle \Leftarrow \mathbf{g}'', \kappa \rangle \propto \langle \Leftarrow \mathbf{true}, \theta \rangle$. Hence, there exists a node labeled by $\langle \Leftarrow \mathbf{g}'', \kappa \rangle$ such that $\mathbf{g}'' \propto \mathbf{true}$ and $\kappa \propto \theta$. Since $\mathbf{g}'' \propto \mathbf{true}$, then $\mathbf{g}'' = \mathbf{true}$ and $\kappa \in \Delta(\Leftarrow \mathbf{g}')$.

Let us consider the inductive case. If $\mathbf{n} > 1$ then: $\langle \Leftarrow \mathbf{g}, \epsilon \rangle \leadsto_{\mathbf{bCN}} \ldots \leadsto_{\mathbf{bCN}} \langle \Leftarrow \mathbf{g_{n-1}}, \theta_{\mathbf{n-1}} \rangle \leadsto_{\mathbf{bCN}} \langle \Leftarrow \mathbf{true}, \theta_{\mathbf{n}} \rangle$. Since $\mathbf{g}' \propto \mathbf{g}$, by Lemma 4.13 and the inductive hypothesis, there exists an abstract basic narrowing derivation $\langle \mathbf{g}', \epsilon \rangle \leadsto_{\mathbf{aCN}} \ldots \leadsto_{\mathbf{aCN}} \langle \Leftarrow \mathbf{g'_{n-1}}, \kappa_{\mathbf{n-1}} \rangle$ such that $\langle \Leftarrow \mathbf{g'_i}, \kappa_{\mathbf{i}} \rangle \propto \langle \Leftarrow \mathbf{g_i}, \theta_{\mathbf{i}} \rangle$, $\mathbf{i} = 1, \ldots, \mathbf{n} - 1$. Since $\langle \Leftarrow \mathbf{g'_{n-1}}, \kappa_{\mathbf{n-1}} \rangle \propto \langle \Leftarrow \mathbf{g_{n-1}}, \theta_{\mathbf{n-1}} \rangle$, by Lemma 4.13 there exists $\langle \Leftarrow \mathbf{g'_n}, \kappa_{\mathbf{n}} \rangle$ such that $\langle \Leftarrow \mathbf{g'_n}, \kappa_{\mathbf{n}} \rangle \propto \langle \Leftarrow \mathbf{true}, \theta_{\mathbf{n}} \rangle$. Therefore $\kappa_{\mathbf{n}} \propto \theta_{\mathbf{n}}$ and $\mathbf{g'_n} \propto \mathbf{true}$, and thus $\mathbf{g'_n} = \mathbf{true}$ and $\kappa_{\mathbf{n}} \in \Delta(\Leftarrow \mathbf{g}')$. □

PROOF. [proof of Theorem 5.2]

To prove this theorem we need to reexamine the proof of Theorem 4.19. Let $\langle \Leftarrow \mathbf{g}, \epsilon \rangle \leadsto_{\mathbf{b}\mathcal{CN}} \ldots \leadsto_{\mathbf{b}\mathcal{CN}} \langle \Leftarrow \mathbf{true}, \theta_{\mathbf{n}} \rangle$ be any successful basic narrowing derivation and let $\langle \Leftarrow \mathbf{g}', \epsilon \rangle \leadsto_{\mathbf{a}\mathcal{CN}} \ldots \leadsto_{\mathbf{a}\mathcal{CN}} \langle \Leftarrow \mathbf{true}, \kappa_{\mathbf{n}} \rangle$ be the corresponding *relevant* abstract basic narrowing derivation. Then we have that $\kappa_{\mathbf{n}} \in \Delta(\Leftarrow \mathbf{g})$, by Theorem 4.19, and for all $\mathbf{i}$, $\mathbf{i} = 1, \ldots, \mathbf{n}$,

- $\kappa_{\mathbf{i}} \preceq \theta_{\mathbf{i}}$
- $\theta_{\mathbf{i}} \leq \theta_{\mathbf{n}}$

By Definition 5.1, it follows that $\kappa_{\mathbf{n}} \diamondsuit \theta_{\mathbf{i}}$, $\mathbf{i} = 1, \ldots, \mathbf{n}$. □

PROOF. [proof of Corollary 5.3] The proof is done by contradiction. Let us assume that there is a successful basic narrowing derivation for $\langle \Leftarrow \mathbf{g}', \theta \rangle$. Then, by Theorem 5.2, there exists $\kappa \in \Delta(\Leftarrow \mathbf{g})$ such that $\kappa \diamondsuit \theta$, which contradicts the hypothesis. □

PROOF. [proof of Corollary 4.20] Immediate from Theorem 4.19. □

PROOF. [proof of Theorem 4.22] Immediate from Theorem 4.19 and the fact that $\mathcal{S}$ is AND-compositional. □

PROOF. [proof of Theorem 4.26]

1. From Definition 4.25, $\langle \mathbf{c}, \Theta \rangle \xrightarrow{\tilde{\mathbf{c}}}_{\mathbf{iEA}} \langle \mathbf{c} \cup \widetilde{\mathbf{c}}, \Theta \Uparrow \Delta(\Leftarrow \widetilde{\mathbf{c}}) \rangle = \langle \mathbf{c} \cup \widetilde{\mathbf{c}}, \Delta(\Leftarrow \mathbf{c}) \Uparrow \Delta(\Leftarrow \widetilde{\mathbf{c}}) \rangle = \langle \mathbf{c} \cup \widetilde{\mathbf{c}}, \Delta(\Leftarrow \mathbf{c} \cup \widetilde{\mathbf{c}}) \rangle$, by Theorem 4.22.

2. Assume, by contradiction, that the constraint $\mathbf{c} \cup \widetilde{\mathbf{c}}$ is satisfiable. It suffices to show that the transition $\langle \mathbf{c}, \Theta \rangle \xrightarrow{\tilde{\mathbf{c}}}_{\mathbf{iEA}} \langle \mathbf{c} \cup \widetilde{\mathbf{c}}, \emptyset \rangle$ cannot be proven for the hypothesis to be contradicted. By case 1, $\langle \mathbf{c}, \Theta \rangle \xrightarrow{\tilde{\mathbf{c}}}_{\mathbf{iEA}} \langle \mathbf{c} \cup \widetilde{\mathbf{c}}, \Delta(\Leftarrow \mathbf{c} \cup \widetilde{\mathbf{c}}) \rangle$. By Corollary 4.20 and the assumption that $\mathbf{c} \cup \widetilde{\mathbf{c}}$ is satisfiable, $\Delta(\Leftarrow \mathbf{c} \cup \widetilde{\mathbf{c}}) \not\equiv \emptyset$.

□

PROOF. [proof of Theorem 5.5] By Theorem 5.2, for every substitution $\theta \in \mathcal{C}(\Leftarrow \mathbf{g})$, there exists $\kappa$ in $\Delta(\Leftarrow \mathbf{g})$ such that $\kappa \diamondsuit \theta$. From Definition 5.4, every narrowing step leading to a node $\langle \Leftarrow \mathbf{g}', \theta \rangle$, with $\kappa \diamondsuit \theta$ for some $\kappa$ in $\Delta(\Leftarrow \mathbf{g})$, is proven. Hence, $\langle \Leftarrow \mathbf{g}, \epsilon \rangle \leadsto^*_{\mathbf{r}\mathcal{CN}} \langle \Leftarrow \mathbf{true}, \sigma \rangle$ iff $\langle \Leftarrow \mathbf{g}, \epsilon \rangle \leadsto^*_{\mathbf{b}\mathcal{CN}} \langle \Leftarrow \mathbf{true}, \sigma \rangle$. Therefore, from the correctness and completeness of $\leadsto_{\mathbf{b}\mathcal{CN}}$, $\{\sigma_{\restriction \mathbf{Var(g)}} \mid \langle \Leftarrow \mathbf{g}, \epsilon \rangle \leadsto^*_{\mathbf{r}\mathcal{CN}} \langle \Leftarrow \mathbf{true}, \sigma \rangle\}$ is a complete set of $\mathcal{E}$-unifiers of $\mathbf{g}$. □

PROOF. [proof of Proposition 5.6]

| | | |
|---|---|---|
| $\phi \diamondsuit \kappa$ | iff | (by Definition 5.1) |
| $\exists\theta.\ \phi \preceq \theta \ and\ \kappa \preceq \theta$ | iff | (by Definition 4.2) |
| $\exists\theta.\ \theta \Rightarrow [\![\phi]\!] \ and\ \theta \Rightarrow [\![\kappa]\!]$ | iff | |
| $\exists\theta.\ \theta \in \mathbf{unif}([\![\phi]\!]) \ and\ \theta \in \mathbf{unif}([\![\kappa]\!])$ | iff | |
| $\exists\theta.\ \theta \in \mathbf{unif}([\![\phi]\!]) \cap \mathbf{unif}([\![\kappa]\!])$ | iff | |
| $\exists\theta.\ \theta \in \mathbf{unif}([\![\phi \cup \kappa]\!])$ | iff | (by Proposition 4.6) |
| $\mathbf{mgu}_{\mathcal{A}}(\phi \cup \kappa) \not\equiv \mathit{fail}.$ | | |

□