ANALYSES OF UNSATISFIABILITY FOR EQUATIONAL LOGIC PROGRAMMING *

MARIA ALPUENTE, MORENO FALASCHI, AND FERDINANDO MANZO

▷ The problem of unifying pairs of terms with respect to an equational theory (as well as detecting the unsatisfiability of a system of equations) is in general undecidable. In this work we define a framework based on abstract interpretation for the (static) analysis of the unsatisfiability of equation sets. The main idea behind the method is to abstract the process of semantic unification of equation sets based on narrowing. The method consists of building an *abstract narrower* for equational theories and executing the sets of equations to be detected for unsatisfiability in the approximated narrower. As an instance of our framework we define a new analysis whose accuracy is enhanced by some simple loop-checking technique. This analysis can also be actively used for pruning the search tree of an incremental equational constraint solver and can be integrated with other methods in the literature. Standard methods are shown to be an instance of our framework. To the best of our knowledge, this is the first framework proposed for approximating equational unification.

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

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

Address correspondence to Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n, Apdo. 22012, 46020 Valencia, Spain, e.mail: alpuente@dsic.upv.es

Address correspondence to Dipartimento di Elettronica e Informatica, Università di Padova, Via Gradenigo 6/A, 35131 Padova, Italy e.mail: falaschi@di.unipi.it

Address correspondence to Dipartimento di Informatica, Università di Pisa, Corso Italia, 40, 56125 Pisa, Italy

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/93/$3.50

1. INTRODUCTION

Equational Logic Programming [30, 34] is a relevant extension of the logic programming paradigm where logic programs are augmented by Horn equational theories. The operational semantics of an equational logic language is some special form of equational resolution (such as SLDE-resolution [26, 30, 34]), which is in turn based upon equational unification [23, 26, 48]. Equational unification ($\mathcal{E}$-*unification* [26, 30]) is the problem of unifying a pair of terms with respect to an equational theory $\mathcal{E}$. There is in general no single most general $\mathcal{E}$-*unifier* of two terms. A complete set of $\mathcal{E}$-*unifiers* of a pair of terms may even be infinite. The set of $\mathcal{E}$-*unifiers* of a pair of terms is semidecidable, even for unconditional and canonical equational theories. The unsatisfiability of a system of equations is also undecidable. A number of $\mathcal{E}$-*unification* procedures have been developed [23, 26, 30, 32, 36]. Conditional narrowing, e.g., has been shown to be complete for equational theories satisfying different restrictions [30, 32, 36, 42].

The definition of any equational logic language must necessarily address the problem of the completeness and termination of $\mathcal{E}$-unification. In order to overcome the latter problem, lazy strategies relying on delayed unification have been defined [2, 14, 30]. In [14, 30] only a partial unifier of the equations is computed and the unsolved equations are added as residuum to the goal clause to be solved later. In [2] unificands are reduced before they are submitted to the unification algorithm. Equations which are not ready for evaluation are postponed until they can be decided. Verified ground equations are then discarded.

Several methods to improve the termination of $\mathcal{E}$-*unification* algorithms have been reported [1, 15, 47]. These algorithms are guided by a graph of terms to detect and remove some loops of the search tree which do not lead to any solution. The improved algorithms are still complete but termination is only guaranteed for theories which satisfy a given condition.

The ability to detect the unsatisfiability of a set of equations is also very important to prune useless paths from the search tree and save a lot of unnecessary computation. The standard methods to recognize failure are based on the idea that two terms headed by different irreducible symbols can never be equal [2, 25]. [22] introduces another decidable notion to detect unsatisfiability which is based on an analysis of joinability of the outermost function symbols of each equation. A close idea is developed in [11] for constructor-based programs, i.e. programs which obey the constructor discipline and where all functions are defined completely. This analysis relies on comparing the sets of possible constructor normal forms of the two sides of each equation, which are approximated by the notion of *abstract rewriting* [11].

In this paper we define a framework based on abstract interpretation for the (static) analysis of the unsatisfiability of equation sets with respect to a given equational theory. We give a formalization of the classical methods defined in [2, 22, 25] as an instance of our framework and show that their correctness can be proven as a consequence of a general theorem (Theorem 4.6) and a specific (simple) lemma (Lemma 6.3) to be proven for each case. We also define a new algorithm for detecting statically when a set of equations is unsatisfiable with respect to an equational theory. This algorithm is based on the idea that rewrite rules can be abstracted and an approximated narrowing can be performed and it is an instance of our framework. 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. Existentially quantified variables can be introduced by our abstract narrower as a result of the abstraction made over the terms appearing in the rewriting rules. We make use of a simple technique of loop detection to build a graph of functional dependencies and improve our method. 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 analysis. We discuss the relation of our method with a similar mechanism to detect equational unsatisfiability as defined in [22].

The algorithm 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 instance of the CLP scheme [33]. In this language equations are treated as constraints and are incrementally added to the "store" to be checked for solvability. This test can be very costly (not to mention termination is not guaranteed) and, as pointed out in [29], partial but efficient constraint solvers turn out to be valuable. Our algorithm always terminates and says that the set of equations is unsatisfiable or returns a finite description of the set of its $\mathcal{E}$-unifiers which can be effectively used to prune the search tree incrementally. The set of answers has the property that each concrete solution is an instance of one of these answers. It is conceivable a lazy procedure for CLP($\mathcal{H}/\mathcal{E}$) which does not prove the solvability of a set of equations c but just checks that c is not "unsatisfiable" by means of our algorithm and then uses the collected set of answers for pruning the search tree of the constraint solver incrementally. A final guided execution of the "full" narrower may find the concrete solutions and possibly detect the unsatisfiability not detected by this lazy procedure. This methodology is similar to that followed by [29]. In the framework of CLP, [29] introduces a relaxation function which associates with each constraint c another constraint (that is implied by c) for which solvability can be easily tested. The unsolvability of the relaxed constraint implies the unsolvability of c. [29] mainly considers constraints over finite domains, the case of equational theories is not considered.

We formalize the idea of approximated narrowing in the framework of abstract interpretation [16, 18]. We prove that our analysis of unsatisfiability always terminates and is correct, i.e. if it terminates with failure the constraint is unsatisfiable. Clearly, since the problem is undecidable the analysis may sometimes fail to detect unsatisfiability.

Interesting to note we abstract term rewriting systems and obtain a form of compiled simplified "program" which always terminates and in which we can execute a compiled query efficiently. Differently to the standard approach to abstract interpretation [41] where all the work of abstraction is left to the procedure which computes the abstract most general unifier of two terms.

Our equational analyzer is incremental. When new equations are added to a set of equations which has already been checked, it is possible to 'merge' the result of the previous analysis with the new equations and hence make an analysis which is simpler than 'restarting from scratch'.

The structure of the paper is as follows. After introducing some preliminary notions in Section 2 we present the concrete basic narrowing in Section 3. In Section 4 we first recall the concept of abstract transition system [16], then we define a finitely failed abstract transition system and give a general correctness theorem of the framework. In Section 5 we define an algorithm to detect unsatisfiable equations and present the (extension and) formalization in our framework of other methods in the literature like [2, 22, 25]. In Section 6 we prove that the analyses presented in this paper are correct. Section 7 shows an incremental equational analyzer which incrementally builds the search tree as long as new constraints are added. In Section 8 we present some optimizations for our analyzer and make a comparison with some related work [2, 11, 22, 25, 30] which we show how it can be suitably integrated with our method.

A preliminary (short) version of this paper appeared in [5].

2. PRELIMINARIES

By Σ, Π and V (possibly subscripted) we denote denumerable (disjoint) collections of function symbols, predicate symbols and variable symbols with their signatures. $\tau(\Sigma \cup V)$ and $\tau(\Sigma)$ denote the sets of terms and ground terms built on Σ and V, respectively. $\tau(\Sigma)$ is usually called the Herbrand Universe ($\mathcal{H}$) over Σ. A Σ-equation $s = t$ is a pair of terms $s, t \in \tau(\Sigma \cup V)$. A (Π, Σ)-atom is an element $p(t_1, \ldots, t_n)$ where $p \in \Pi$ is n-ary and $t_i \in \tau(\Sigma \cup V)$, $i = 1, \ldots, n$. A (Π_C, Σ)-constraint is a (possibly empty) finite set of (Π_C, Σ)-atoms, with $\Pi_C \subseteq \Pi$. Throughout the paper we assume $\Pi_C = \{=\}$. $Var(s)$ is the set of distinct variables occurring in the syntactic object s. A fresh variable is a variable which appears nowhere else. Identity of syntactic objects is denoted by $\equiv$. The symbol $\tilde{}$ denotes a finite sequence of symbols.

Let us briefly recall some basic notions about equations, conditional rewrite systems and universal unification. For full definitions refer to [19, 37]. Terms are viewed as labelled trees in the usual way. The depth of t, written $depth(t)$, is defined recursively as follows. If t is a constant or a variable, then $depth(t)$ is 1. Also, the depth of $f(t_1, \ldots, t_n)$ is $1 + max(\{depth(t_1), \ldots, depth(t_n)\})$. Occurrences are represented by sequences, possibly empty, of natural numbers used to address subterms of t, and they are ordered by the prefix ordering $u \leq v$ if there exists a w such that $uw = v$. $O(t)$ denotes the set of occurrences of a term t. $\bar{O}(t)$ denotes the set of nonvariable occurrences of a term t. $t|_u$ is the subterm at the occurrence u of t. $t[r]_u$ is the term t with the subterm at the occurrence u replaced with r. $t[u]$ denotes the label in t at occurrence $u \in O(t)$. These notions extend to equations in a natural way. A substitution is defined as an almost identical mapping from the set of variables V into the set of terms $\tau(\Sigma \cup V)$. It naturally extends to a mapping from terms to terms or from equations over terms to equations over terms. We restrict our interest to the set of idempotent substitutions over $\tau(\Sigma \cup V)$, which is denoted by Sub. In abuse of notation, $Dom(\sigma) = \{x \in V \mid x\sigma \not\equiv x\}$ is called the domain of σ. The equational representation of a substitution $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is the set of equations, $\{x_1 = t_1, \ldots, x_n = t_n\}$. The empty substitution is denoted by ϵ. We describe the lattice of equation sets following [17]. Also see [39, 40]. We let Eqn denote the set of possibly existentially quantified finite sets of equations over terms. Elements of Eqn are regarded as (quantified) conjunctions of equations and treated modulo logical equivalence. We let $fail$ denote the unsatisfiable equation set, which (logically) implies all other equation sets. Likewise, the empty equation set, denoted by $true$, is implied by all elements of Eqn. We write $E \leq E'$ if E' logically implies 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

$\exists y_1 \ldots \exists y_m . \{x_1 = t_1, \ldots, x_n = t_n\}$ where each x_i is a distinct variable not occurring in any of the terms t_i and each y_i occurs in some t_j. Every set of equations has at least one (logically) equivalent solved form. We let the non-deterministic function $solve(E)$ choose one of these solved forms of equation sets E. For example, a solved form of $\exists x.\{x = f(y), z = y\}$ is $\{z = y\}$ and $\exists y.\{x = f(y), z = y\}$ is in solved form. There is a natural isomorphism between substitutions and unquantified equation sets.

We use the same notation for a substitution $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, the corresponding set of equations $\{x_1 = t_1, \ldots, x_n = t_n\}$ and the corresponding conjunction of equations $x_1 = t_1 \wedge \ldots \wedge x_n = t_n$. For example we may write $\theta \wedge (y = t\theta)$, where the first occurrence of θ is a conjunction of equations and the second occurrence is the application of a substitution.

The notions of application, composition and relative generality are defined in the usual way [9]. We consider the usual preorder on substitutions $\leq$: $\theta \leq \sigma$ iff $\exists \gamma . \sigma \equiv \theta\gamma$. Note that $\theta \leq \sigma$ iff $\sigma \Rightarrow \theta$ [45]. A substitution $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is a *unifier* of an equation set E if $E \leq \{x_1 = t_1, \ldots, x_n = t_n\}$, i.e. $\{x_1 = t_1, \ldots, x_n = t_n\} \Rightarrow E$. We denote the set of unifiers of E by $unif(E)$ and $mgu(E)$ denotes the *most general unifier* of the unquantified equation set E.

While every unquantified equation set has a *most general unifier* [38], this is not true in general for equation sets with existentially quantified variables. For example, the equation set $\exists y.\{x = f(y)\}$ does not have a most general unifier.

We note that when dealing with the concrete operational semantics the extra machinery of quantified variables is not needed, all equations can be thought of as unquantified, and hence as isomorphic to a most general unifier. In the abstract semantics our 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. Hence, it is only when arguing the relation between the concrete and abstract operational semantics that we need to consider quantified equation sets.

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

A Term Rewriting System (TRS for short) is a pair $(\Sigma, \mathcal{R})$ where $\mathcal{R}$ is a finite set of (conditional) reduction (or rewrite) rule schemes of the form $(\lambda \rightarrow \rho \Leftarrow C)$, $\lambda, \, \rho \in \tau(\Sigma \cup V), \; \lambda \notin V$ and $Var(\rho) \subseteq Var(\lambda)$. The condition C is a possibly empty conjunction $e_1, \ldots, e_n, \;\; n \geq 0$, of equations. Variables in C that do not occur in λ are called extra-variables. If a rewrite rule has no condition we write $\lambda \rightarrow \rho$. When the condition in every rule in $\mathcal{R}$ is empty the system is said to be unconditional. Otherwise, it is said to be conditional. We will often write just $\mathcal{R}$ instead of $(\Sigma, \mathcal{R})$.

A Horn equational theory $\mathcal{E}$ which satisfies the above assumptions can be viewed as a term rewriting system $\mathcal{R}$ where the rules are the heads (implicitly 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 unless we explicitly state otherwise. 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 [37]. For syntactical characterizations of confluent conditional theories refer to [10, 13, 43].

A term s conditionally rewrites to a term t, written $s \rightarrow_{\mathcal{R}} t$, if there exists a rule

$(\lambda \to \rho \;\Leftarrow\; s_1 = t_1, \ldots, s_n = t_n) \in \mathcal{R}$, an occurrence $u \in O(s)$, and a substitution σ such that $s|_u = \lambda\sigma$, $t = s[\rho\sigma]_u$ and, for each $i = 1, \ldots, n$, there exists a term w_i such that $s_i\sigma \to^*_{\mathcal{R}} w_i$ and $t_i\sigma \to^*_{\mathcal{R}} w_i$. Two terms s and t are *joinable*, $s \downarrow_{\mathcal{R}} t$ if there exists a term w such that $s \to^*_{\mathcal{R}} w$ and $t \to^*_{\mathcal{R}} w$. When no confusion can arise, we omit the subscript $\mathcal{R}$.

For TRS $\mathcal{R}$, $r \ll \mathcal{R}$ denotes that r is a new variant of a rule in $\mathcal{R}$ such that r contains no variable previously met during computation (standardised apart). The instantiated left-hand side $\lambda\sigma$ of a reduction rule $(\lambda \to \rho \;\Leftarrow\; C)$ is called a *redex* (*red*ucible *ex*pression) with *contractum* $\rho\sigma$.

Let $\mathcal{R}$ be a TRS. A function symbol $f \in \Sigma$ is irreducible iff there is no rule $(\lambda \to \rho \;\Leftarrow\; C) \in \mathcal{R}$ such that f occurs as the outermost function symbol in λ, otherwise it is a defined function symbol. In theories where the above distinction is made, the signature Σ is partitioned as $\Sigma = \mathcal{C} \biguplus \mathcal{F}$, where $\mathcal{C}$ is the set of irreducible function symbols and $\mathcal{F}$ is the set of defined function symbols. The elements of $\mathcal{C}$ are often called constructors.

An important kind of Horn equational theories are those which are level-confluent [27, 42]. Let $\to_{\mathcal{R}}$ be the conditional rewrite relation corresponding to a Horn equational theory $\mathcal{E}$. Then $\to_{\mathcal{R}}$ is equivalent to $\to_{\mathcal{R}} = \cup_{i \geq 0}\{\to_{\mathcal{R}_i}\}$, where:

(1) $\to_{\mathcal{R}_0} = \emptyset$; and

(2) $t \to_{\mathcal{R}_{n+1}} t'$ holds iff there is a rule $(\lambda \to \rho \;\Leftarrow\; C) \in \mathcal{R}$, a non-variable occurrence u of t and a substitution σ such that $t|_u = \lambda\sigma$, $t' = t[\rho\sigma]_u$ and $s\sigma \downarrow_{\mathcal{R}_n} s'\sigma$ for all $(s = s') \in C$. $\to_{\mathcal{R}}$ is called level-confluent [27] iff, for each $n \geq 0$, the relation $\to_{\mathcal{R}_n}$ is confluent. Level confluence can still be ensured by syntactic conditions [27]. Of course, level confluence implies confluence. We call $\mathcal{R}$ level-canonical if $\to_{\mathcal{R}}$ is level-confluent and noetherian. This is equivalent to say that each $\to_{\mathcal{R}_n}$ is canonical. The notion of level-canonicity of a conditional term rewriting system extends in the obvious way to Horn equational theories.

Each Horn equational theory $\mathcal{E}$ generates a smallest congruence relation $=_{\mathcal{E}}$ called *$\mathcal{E}$-equality* on the set of terms $\tau(\Sigma \cup V)$ (the least equational theory which contains all logical consequences of $\mathcal{E}$ under the entailment relation $\models$ obeying the axioms of equality for $\mathcal{E}$). $\mathcal{E}$ 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 $\mathcal{E}$.

Given a finite set of equations $E = \{s_1 = t_1, \ldots, s_n = t_n\}$, we say that E is *$\mathcal{E}$-unifiable* iff there exists a substitution σ such that $s_i\sigma =_{\mathcal{E}} t_i\sigma$ for each i. The substitution σ is called an *$\mathcal{E}$-unifier* of E. *$\mathcal{E}$-unification* is semidecidable.

For an equational theory, the notion of most general unifier generalizes to complete sets of minimal (incomparable) *$\mathcal{E}$-unifiers*. A set S of $\mathcal{E}$-unifiers of the equation set E is complete iff every $\mathcal{E}$-unifier σ of E factors into $\sigma =_{\mathcal{E}} \theta\gamma$ for some substitutions $\theta \in S$ and γ. A complete set of $\mathcal{E}$-unifiers of a system of equations may be infinite. Minimal complete sets of *$\mathcal{E}$-unifiers* 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 sets. By applying narrowing or some refinement or variant thereof, it is possible to construct a complete set of $\mathcal{E}$-unifiers for a given set of equations in canonical theories satisfying different restrictions [30, 32, 42].

3. BASIC CONDITIONAL NARROWING

Given a conditional TRS $\mathcal{R}$, an equational goal clause g conditionally narrows into a goal clause g' if there exists an equation $e \in g$, $u \in \bar{O}(e)$, a (standardised apart) variant $(\lambda \to \rho \Leftarrow C) \ll \mathcal{R}$ and a substitution σ such that $\sigma = mgu(\{e|_u = \lambda\})$ and $g' = ((g \sim \{e\}) \cup \{e[\rho]_u\} \cup C)\sigma$. s is called a (narrowing) *redex* iff there exists a new variant $(\lambda \to \rho \Leftarrow C)$ of a reduction rule in $\mathcal{R}$ and a substitution σ such that $s\sigma \equiv \lambda\sigma$.

Basic (conditional) narrowing is a restricted form of (conditional) narrowing where only terms at *basic* occurrences are considered to be narrowed [31, 42]. 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 preceding narrowing step. The idea behind the concept of *basic* is to avoid narrowing steps on subterms that are introduced by instantiation. Basic Conditional Narrowing (BCN) is a complete *$\mathcal{E}$-unification* algorithm for level-canonical Horn equational theories [42].

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 [30].

The *skeleton* part is a set of equations g and the *environment* part is a substitution θ. Substitutions are composed in the environment part, but are not applied to the terms in the skeleton part. Due to this representation, the basic occurrences in $g\theta$ are all in g, whereas the non-basic occurrences are all in the codomain of θ. The set of goals (*states*) is denoted by *State*. 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 reduction between states. Thus reduction sequences correspond to paths in the graph from the source. To solve the equation set g, the algorithm starts with the initial state $\langle g, \epsilon\rangle$ and tries to derive new goals until a terminal goal $\langle true, \phi\rangle$ is reached. Each substitution ϕ in a terminal goal is an *$\mathcal{E}$-unifier* of g. By abuse of notation, it is often called solution. The BCN calculus is defined by the two following rules:

BCN unification rule:

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

BCN narrowing rule:

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

In the following section we recall a general method to approximate transition systems [16]. We then show how to use this method for approximating term rewriting systems and narrowers.

4. APPROXIMATING TRANSITION SYSTEMS

4.1. Abstract Interpretation

Abstract interpretation [18] formalizes the concept of a semantic based analysis of programs. The computations are performed with respect to descriptions (i.e. approximations) of data rather than the data itself. Thus, an analysis is a computation performed with respect to a *non-standard interpretation* of data and operators in the program. The *standard interpretation* gives the usual behavior of the program.

We follow the approach of [16]. Thus we define the "approximation relation" in terms of a "concretization function" which maps elements in a non-standard domain to those elements in the standard domain which they describe. The approximation relation is then lifted from the "base" domains to relations and tuples.

Definition 4.1. [16] A *description* $Desc = \langle D, \gamma, E\rangle$ consists of a *description domain* (a poset) D, a *data domain* (a poset) E, and a monotonic *concretization function* $\gamma : D \rightarrow \wp\, E$. When $E = Eqn$, $E = Sub$ or $E = State$, the description is called an *equation description*, a *substitution description* or a *state description*, respectively.

We say that d γ-approximates e, written $d \propto_\gamma e$, iff $e \in \gamma(d)$. The approximation relation is lifted to binary relations and cross products as follows:

- Let $\langle D_1, \gamma_1, E_1\rangle$ and $\langle D_2, \gamma_2, E_2\rangle$ be descriptions, and $F \subseteq D_1 \times D_2$ and $F' \subseteq E_1 \times E_2$ be relations. Then $F \propto F'$ iff $\forall d \in D_1.\ \forall e \in E_1.\ d \propto_{\gamma_1} e \wedge \langle e, e'\rangle \in F' \Rightarrow \exists \langle d, d'\rangle \in F \wedge d' \propto_{\gamma_2} e'$.
- Let $\langle D_1, \gamma_1, E_1\rangle$ and $\langle D_2, \gamma_2, E_2\rangle$ be descriptions, and $(d_1, d_2) : D_1 \times D_2$ and $(e_1, e_2) : E_1 \times E_2$. Then $(d_1, d_2) \propto (e_1, e_2)$ iff $d_1 \propto_{\gamma_1} e_1 \wedge d_2 \propto_{\gamma_2} e_2$.

When clear from the context we say that d approximates e and write $d \propto e$. By an abuse of notation we will sometimes let $Desc$ denote both the description and the description domain.

Given an equation and a substitution description, we can define the concept of "abstract transition system". We have chosen the approach of abstracting transition systems, similarly to [16], for two reasons. The method results to be quite simple and general since standard formal operational semantics for narrowing are often given in terms of transition systems. Besides, the abstract algorithm requires the collection of all intermediate states for proving the unsatisfiability of a set of equations.

4.2. Abstract Transition System

Given an equation description and a substitution description, an associated state description is constructed as follows:

Definition 4.2. Let $Eqn_{\mathcal{A}}$ be an equation description, $Sub_{\mathcal{A}}$ a substitution description. Define the *abstract state domain*, $State_{\mathcal{A}}$ induced by $Eqn_{\mathcal{A}}$ and $Sub_{\mathcal{A}}$, to be $State_{\mathcal{A}} = \{\langle g, \kappa\rangle \mid g \in Eqn_{\mathcal{A}}, \kappa \in Sub_{\mathcal{A}}\}$. Associated with $State_{\mathcal{A}}$ are extraction functions $sub : State_{\mathcal{A}} \rightarrow Sub_{\mathcal{A}}$ and $eqn : State_{\mathcal{A}} \rightarrow Eqn_{\mathcal{A}}$ defined by $eqn(t) = g$, if $t = \langle g, \kappa \rangle$ and $sub(t) = \kappa$, if $t = \langle g, \kappa \rangle$. The definition of eqn and sub extend naturally to any set of pairs.

Abstract states describe states which are equivalent modulo variable renaming. This is because the operational behaviour preserves this type of equivalence, and so it is not necessary to distinguish between such states. Furthermore, to ensure termination of analyses we do not distinguish between abstract states which describe the same set of states.

Definition 4.3. [16] induced state description
Let $State_{\mathcal{A}}$ be an abstract state domain induced from an equation description $Eqn_{\mathcal{A}}$ and a substitution description $Sub_{\mathcal{A}}$. Let $t \in State_{\mathcal{A}}$, $s \in State$ and ρ be a renaming. Then $t \propto_{\rho} s$ iff:

1. $sub(t) \propto \rho(sub(s))$; and
2. $eqn(t) \propto \rho(eqn(s))$.

We write $t \propto s$ iff $\exists \rho . t \propto_{\rho} s$. Define $\gamma : State_{\mathcal{A}} \rightarrow \wp\, State$ by $\gamma(t) = \{s \mid t \propto s\}$, and let $\simeq$ be the equivalence relation on elements of $State_{\mathcal{A}}$ induced by γ; that is $t_1 \simeq t_2$ iff $\gamma(t_1) = \gamma(t_2)$. We let $State_{\mathcal{A}}/\simeq$ denote the equivalence classes of $State_{\mathcal{A}}$ and rep be a function returning a representative of an equivalence class. The induced state description for $Eqn_{\mathcal{A}}$ and $Sub_{\mathcal{A}}$ is $\langle State_{\mathcal{A}}/\simeq, \gamma \circ rep, State \rangle$.

Let us now define an abstract transition system. The definition is parametric with respect to a generic TRS, the set of abstract states and a reduction relation between abstract states. These notions have to be instantiated in the specialization process.

Definition 4.4. abstract transition system
Let T be a TRS, s a state and $State_{\mathcal{A}}$ an induced state description. Let $\overset{\mathcal{A}}{\hookrightarrow}_T \subseteq State_{\mathcal{A}} \times State_{\mathcal{A}}$ be a relation (depending on T). An abstract transition system for $\overset{\mathcal{A}}{\hookrightarrow}_T$, s and $State_{\mathcal{A}}$ is a transition graph $\mathcal{G}$ with elements of $State_{\mathcal{A}}/\simeq$ as nodes which is defined in terms of the abstract reduction relation $\overset{\mathcal{A}}{\hookrightarrow}_T$. The graph $\mathcal{G}$ must satisfy:

- The source of $\mathcal{G}$ is an approximation of s.
- Let $\hat{t} \in State_{\mathcal{A}}/\simeq$ denote the equivalence class of $t \in State_{\mathcal{A}}$. There is an edge from $\hat{t}$ to $\hat{t}'$ iff $t \overset{\mathcal{A}}{\hookrightarrow}_T t'$.

An abstract transition system should approximate the concrete transition system semantics BCN we introduced in Section 3. The basic idea is to approximate the behaviour of a TRS and initial state by a *finite* transition graph with nodes labeled by state descriptions so that the non-existence of a node labeled by $\langle true, \kappa \rangle$ ensures the unsatisfiability of the equation set. This motivates the following definition.

Definition 4.5. A (finite) abstract transition system for $\overset{\mathcal{A}}{\hookrightarrow}_T$, s and $State_{\mathcal{A}}$ is *finitely failed* iff it does not contain leaves of the form $\langle true, \kappa \rangle$.

The relevance of this notion is clarified by the following theorem.

Theorem 4.6. (Correctness) Let $\mathcal{E}$ be a level-canonical Horn equational theory with associated TRS $\mathcal{R}$. Let $\propto$ be an approximation relation such that $\forall d \in Eqn_{\mathcal{A}}.(d \propto true \Rightarrow d \equiv true)$. Let T be a TRS such that $\overset{\mathcal{A}}{\hookrightarrow}_T \ \propto \leadsto_{\mathcal{R}}$. If the abstract transition system for $\overset{\mathcal{A}}{\hookrightarrow}_T$, state $s \propto \langle g, \epsilon\rangle$ and $State_{\mathcal{A}}$ is finitely failed then g is unsatisfiable in $\mathcal{E}$.

Proof. We prove the theorem by contradiction. We prove that if the set of equations g in the state $\langle g, \epsilon\rangle$ is satisfiable then, in any abstract transition system for state $s \propto \langle g, \epsilon\rangle$ there is a leaf of the form $\langle true, \kappa\rangle$. If g is satisfiable in $\mathcal{E}$ then there exists an $\mathcal{E}$-unifier of g. Thus, by the completeness of Basic Conditional Narrowing, there exists a path $\langle g, \epsilon\rangle \leadsto_{\mathcal{R}} \ldots \leadsto_{\mathcal{R}} \langle true, \sigma\rangle$. We prove the theorem by induction on the length n of the path.

If $n = 1$ then $\langle g, \epsilon\rangle \leadsto_{\mathcal{R}} \langle true, \sigma\rangle$ where $\sigma = mgu(g)$. Since $s \propto \langle g, \epsilon\rangle$ and $\overset{\mathcal{A}}{\hookrightarrow}_T \ \propto \leadsto_{\mathcal{R}}$, by Definition 4.1, there exists an abstract reduction from s to some $t' = \langle g', \kappa\rangle$ such that $t' \propto \langle true, \sigma\rangle$. Hence, there exists a node $\langle g', \kappa\rangle$ such that $g' \propto true$ and $\kappa \propto \sigma$. Since $g' \propto true$ then $g' = true$.

Let us consider the inductive case. If $n > 1$ then $\langle g, \epsilon\rangle = \langle g_1, \theta_1\rangle \leadsto_{\mathcal{R}} \ldots \leadsto_{\mathcal{R}} \langle g_{n-1}, \theta_{n-1}\rangle \leadsto_{\mathcal{R}} \langle true, \sigma\rangle$. Since $s \propto \langle g, \epsilon\rangle$ and $\overset{\mathcal{A}}{\hookrightarrow}_T \propto \leadsto_{\mathcal{R}}$, there exists an abstract reduction derivation $s = \langle g'_1, \theta'_1\rangle \overset{\mathcal{A}}{\hookrightarrow}_T \ldots \overset{\mathcal{A}}{\hookrightarrow}_T \langle g'_{n-1}, \theta'_{n-1}\rangle$ such that $\langle g'_i, \theta'_i\rangle \propto \langle g_i, \theta_i\rangle$, $1 \leq i \leq n-1$. Since $\langle g'_{n-1}, \theta'_{n-1}\rangle \propto \langle g_{n-1}, \theta_{n-1}\rangle$, by Definition 4.1 there exists $\langle g'_n, \theta'_n\rangle$ s.t. $\langle g'_{n-1}, \theta'_{n-1}\rangle \overset{\mathcal{A}}{\hookrightarrow}_T \langle g'_n, \theta'_n\rangle$ and $\langle g'_n, \theta'_n\rangle \propto \langle true, \sigma\rangle$. Hence $g'_n \propto true$ and therefore $g'_n = true$. $\square$

In the next section we present our analysis of unsatisfiability. 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. ANALYSES OF EQUATIONAL UNSATISFIABILITY

5.1. *Abstract (Basic) Narrowing Analysis*

In this section we present an abstract algorithm to prove that a set of equations is unsatisfiable in a (level-canonical) Horn equational theory. We define the descriptions for terms, equations, substitutions and term rewriting systems. We need some technical definitions first. Notice that we do not distinguish between equivalence classes and representatives. Moreover, by abuse of notation, we denote in the same way a preorder and the corresponding partial ordering induced on the equivalence classes of the equivalence relation associated with the preorder.

By $\mathcal{T} = (\tau(\Sigma \cup V), \leq)$ 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 $\mathcal{T}^{\perp} = (\tau(\Sigma \cup V \cup \{\perp\}), \preceq)$ be the domain of terms over the signature augmented by $\perp$, where the partial order $\preceq$ is defined as follows:

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

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

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

1) $S' \preceq S$ iff $\forall e' \in S'.\exists e \in S$ such that $e' \preceq e$.

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

We note that $S' \preceq true \Rightarrow S' \equiv true$. Intuitively, $S' \sqsubseteq S$ means that either S' contains less information than S, or if they have the same information, then S' expresses it by less elements.

Define $[\![S]\!] = S'$, where the n-tuple of occurrences of $\perp$ in S is replaced by an n-tuple of existentially quantified distinct fresh variables in S'.

The relation between the order $\preceq$ over terms, equations and sets of equations and the logical order over terms, equations and sets of equations is stated in the following lemmata and proposition.

Lemma 5.1. Let t, t' be terms. $t' \preceq t$ iff $\forall u \in O(t).(\exists w \leq u.t'[w] \equiv \perp$ or $u \in O(t') \wedge t[u] \equiv t'[u])$

PROOF.

($\Rightarrow$) By structural induction on t.
If $depth(t) = 1$, then $t' \equiv \perp$ or $t' \equiv t$.
If $depth(t) = k+1$, $k \geq 0$, and the lemma holds for all t such that $depth(t) \leq k$, then $t \equiv f(t_1, \ldots, t_n)$ and $t' \equiv \perp$ or $t' \equiv f(t'_1, \ldots, t'_n)$ and $t'_1 \preceq t_1 \wedge \ldots \wedge t'_n \preceq t_n$. Let us consider the second case since the case when $t' \equiv \perp$ is immediate. Since $O(t) = \{i.O(t_i) \mid i = 1, \ldots, n\}$ and $O(t') = \{i.O(t'_i) \mid i = 1, \ldots, n\}$, then the lemma follows by the inductive hypothesis, since $\forall i = 1, \ldots, n.depth(t_i) \leq k$.

($\Leftarrow$) Straightforward.

□

Lemma 5.2. Let e, e' be two equations such that $e' \preceq e$. Then $[\![e]\!] \Rightarrow [\![e']\!]$.

PROOF. By induction on the number n of occurrences of $\perp$ in e'.

Base case. Let $n = 1$. Assume that $(e'|_u) \equiv \perp$ and $(e|_u) \equiv t$. Since there is only one occurrence of $\perp$ in e' then by Lemma 5.1 $e[y]_u \equiv e'[y]_u$, where $y \notin Var(e) \cup Var(e')$. Thus $[\![e]\!] \Leftrightarrow [\![\exists x.(e[x]_u \wedge x = t)]\!]$ (where $x \notin Var(e) \cup Var(e')$) $\Leftrightarrow \exists x.(e[x]_u \wedge [\![x = t]\!]) \Rightarrow \exists x.(e[x]_u \wedge \exists y.x = y)$ (where $y \notin Var(e) \cup Var(e') \cup \{x\}$) $\Leftrightarrow \exists y.(e[x]_u \wedge \exists x.x = y) \Rightarrow \exists y.e[y]_u \Leftrightarrow [\![e']\!]$.

Inductive case. Let $n > 1$ and assume that the claim holds for $k < n$. Assume that $(e'|_u) \equiv \perp$ and $(e|_u) \equiv t$.

$[\![e]\!] \Leftrightarrow [\![\exists x.(e[x]_u \wedge x = t)]\!]$ (where $x \notin Var(e) \cup Var(e')$) $\Leftrightarrow \exists x.([\![e[x]_u]\!] \wedge [\![x = t]\!]) \Rightarrow \exists x.([\![e'[x]_u]\!] \wedge \exists y.x = y)$ (where $y \notin Var(e) \cup Var(e') \cup \{x\}$ and by inductive hypothesis since $e'[x]_u$ contains $n-1$ occurrences of $\perp$ and $e'[x]_u \preceq e[x]_u$) $\Leftrightarrow \exists y.([\![e'[y]_u]\!] \wedge \exists x.x = y) \Rightarrow \exists y.[\![e'[y]_u]\!] \Leftrightarrow [\![e']\!]$. □

Proposition 5.3. Let S, S' be two equation sets such that $S' \preceq S$. Then $[\![S]\!] \Rightarrow [\![S']\!]$.

PROOF. Let $soln(S)$ denote the set of solutions of the equation set S, i.e. the set of all ground unifiers of S. $(soln([\![S]\!]) \subseteq soln([\![S']\!])) \Leftrightarrow ([\![S]\!] \Rightarrow [\![S']\!])$. We thus show that $soln([\![S]\!]) \subseteq soln([\![S']\!])$. If $S' \preceq S$ then for every $e' \in S'.\exists e \in S$ s.t. $e' \preceq e$.

Thus there exists a function $f : S' \to S$ such that $\forall e' \in S'.e' \preceq f(e')$. Then, we have that

$soln(\llbracket S' \rrbracket) = \bigcap_{e'_i \in S'} soln(\llbracket e'_i \rrbracket) \supseteq \bigcap_{e'_i \in S'} soln(\llbracket f(e'_i) \rrbracket)$ (by Lemma 5.2) $=$
$\bigcap_{e_i \in S \cap f(S')} soln(\llbracket e_i \rrbracket) \supseteq \bigcap_{e \in S} soln(\llbracket e \rrbracket) = soln(\llbracket S \rrbracket)$. □

Definition 5.4. An abstract substitution is a set of the form $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where, for each $i = 1, \ldots, n$, x_i is a distinct variable in V not occurring in any of the terms $t_1, \ldots, t_n$ and $t_i \in \tau(\Sigma \cup V \cup \{\bot\})$. We define the following preorder on (abstract) substitutions: let $\theta, \kappa \in Sub^{\bot}$, $\kappa \preceq \theta$ iff $\llbracket \theta \rrbracket \Rightarrow \llbracket \kappa \rrbracket$.

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

Definition 5.5. upward
Let $(X, \leq)$ be a poset and let $Y \subseteq X$. Define $upw(Y) = \{x \in X | \exists y \in Y.y \leq x\}$.

We can now define the term, equation and substitution descriptions. It is easy to verify that the concretization function associated with each description is monotonic.

Definition 5.6. Let $\mathcal{T} = (\tau(\Sigma \cup V), \leq)$ and $\mathcal{T}^{\bot} = (\tau(\Sigma \cup V \cup \{\bot\}), \preceq)$. The *term description* is $\langle \mathcal{T}^{\bot}, \gamma, \mathcal{T} \rangle$ where $\gamma : \mathcal{T}^{\bot} \to \wp\,\mathcal{T}$ is defined by: $\gamma(t') = \{t \in \mathcal{T} \mid t \in upw(\{t'\})\}$.

Definition 5.7. Let Eqn be the set of finite sets of equations over $\tau(\Sigma \cup V)$ and $Eqn^{\bot}$ be the set of finite sets of equations over $\tau(\Sigma \cup V \cup \{\bot\})$. The *equation description* is $\langle (Eqn^{\bot}, \sqsubseteq), \gamma, (Eqn, \leq) \rangle$, where $\gamma : Eqn^{\bot} \to \wp\, Eqn$ is defined by: $\gamma(g') = \{g \in Eqn \mid g \in upw(\{g'\})$ and g is unquantified$\}$.

Definition 5.8. Let Sub be the set of substitutions over $\tau(\Sigma \cup V)$ and $Sub^{\bot}$ be the set of abstract substitutions over $\tau(\Sigma \cup V \cup \{\bot\})$. The *substitution description* is $\langle (Sub^{\bot}, \preceq), \gamma, (Sub, \leq) \rangle$, where $\gamma : Sub^{\bot} \to \wp\, Sub$ is defined by: $\gamma(\kappa) = \{\theta \in Sub \mid \theta \in upw(\{\kappa\})\}$.

The behaviour of the symbol $\bot$ from a programming viewpoint resembles that of an "anonymous" variable in Prolog. From a logical viewpoint, $\bot$ stands for an existentially quantified variable [39, 40]. Thus the unification algorithm has to be modified accordingly. We define the abstract most general unifier for an equation set $E' \in Eqn^{\bot}$ as follows. First replace all occurrences of $\bot$ in E' by distinct fresh existentially quantified variables. Then take a solved form of the resulting quantified equation set and finally replace the existentially quantified variables again by $\bot$. Formally:

Definition 5.9. (Abstract most general unifier, $mgu_{\mathcal{A}}$)
Let $\exists y_1 \ldots y_n.E = solve(\llbracket E' \rrbracket)$ and $\kappa = \{y_1 \mapsto \bot, \ldots, y_n \mapsto \bot\}$. Then

$mgu_{\mathcal{A}}(E') = E\kappa$

Example 5.10. Let $E = \{f(g(\bot),\ h(z,z)) = f(x, h(x, g(a))),\ y = g(\bot)\}$. Then $mgu_{\mathcal{A}}(E) = \{x \mapsto g(a), y \mapsto g(\bot),\ z \mapsto g(a)\}$.

The following proposition shows in what sense the abstract mgu is the "most general" unifier.

Proposition 5.11. $\forall\theta \in unif([\![E]\!]).mgu_{\mathcal{A}}(E) \preceq \theta$.

PROOF. By Definition 5.9, $[\![mgu_{\mathcal{A}}(E)]\!] \Leftrightarrow solve([\![E]\!])$, hence $unif([\![mgu_{\mathcal{A}}(E)]\!]) = unif(solve([\![E]\!])) = unif([\![E]\!])$. Let $\theta \in unif([\![mgu_{\mathcal{A}}(E)]\!])$, then $\theta \Rightarrow [\![mgu_{\mathcal{A}}(E)]\!]$, i.e. $mgu_{\mathcal{A}}(E) \preceq \theta$. Therefore, $\forall\theta \in unif([\![mgu_{\mathcal{A}}(E)]\!]).mgu_{\mathcal{A}}(E) \preceq \theta$. □

Lemma 5.12. Let g, g' be (unquantified) finite sets of equations over $\tau(\Sigma \cup V)$ and $\tau(\Sigma \cup V \cup \{\bot\})$, respectively. If $g \Rightarrow [\![g']\!]$, then there exists $mgu_{\mathcal{A}}(g')$ where $mgu_{\mathcal{A}}(g') \propto mgu(g)$.

PROOF. We have to prove that $mgu(g) \Rightarrow [\![mgu_{\mathcal{A}}(g')]\!]$. Since $g \Rightarrow [\![g']\!]$, then $solve(g) \Rightarrow solve([\![g']\!]) \Leftrightarrow [\![mgu_{\mathcal{A}}(g')]\!]$. □

The following corollary expresses the "safety" of abstract mgu.

Corollary 5.13. Let g, g' be (unquantified) finite sets of equations over $\tau(\Sigma \cup V)$ and $\tau(\Sigma \cup V \cup \{\bot\})$, respectively. If $g' \propto g$ and there exists $mgu(g) = \sigma$, then there exists $mgu_{\mathcal{A}}(g') = \sigma_{\mathcal{A}}$ where $\sigma_{\mathcal{A}} \propto \sigma$.

PROOF. Immediate from Proposition 5.3 and Lemma 5.12 □

The graph $\mathcal{G}_{\mathcal{R}}$ of functional dependencies is constructed to perform a simple loop check over the function symbols involved in the term rewriting system $\mathcal{R}$ as formalized by the following definition.

Definition 5.14. Graph of functional dependencies

Let $(\mathcal{C} \uplus \mathcal{F}, \mathcal{R})$ be a term rewriting system. The following transformation defines the directed graph $\mathcal{G}_{\mathcal{R}}$ of functional dependencies induced by $\mathcal{R}$. To build $\mathcal{G}_{\mathcal{R}}$, the algorithm starts with $\langle\mathcal{R}, \emptyset\rangle$ and applies the inference rule as long as the terminal configuration $\langle\emptyset, \mathcal{G}_{\mathcal{R}}\rangle$ has not been reached. The symbol $\cup$ stands for set union.

$$\frac{r = (f(t_1,\ldots,t_n) \to \rho \ \Leftarrow\ C) \in \mathcal{R}}{\begin{array}{lll}\langle\mathcal{R}, \mathcal{G}_{\mathcal{R}}\rangle & \longmapsto & \langle\mathcal{R} \sim \{r\}, \mathcal{G}_{\mathcal{R}} \cup \{f \stackrel{dep}{\to} h \mid\ u \in \bar{O}(\rho)\ \wedge\ \rho[u] = h \in \mathcal{F}\} \\ & & \cup\, \{f \stackrel{dep}{\to} h \mid\ e \in C\ \wedge\ u \in (\bar{O}(e) \sim \{\Lambda\})\ \wedge\ e[u] = h \in \mathcal{F}\}\rangle\end{array}}$$

Roughly speaking, for each rule $(f(t_1,\ldots,t_n) \to \rho \ \Leftarrow\ C)$ in $\mathcal{R}$ and for each function symbol h occurring in ρ or C we add an arc $f \stackrel{dep}{\to} h$ to $\mathcal{G}_{\mathcal{R}}$. Note that we do not need to consider variable occurrences of terms since we are only concerned with basic derivations. The termination of the calculus is ensured since the number of reduction rules in $\mathcal{R}$ is finite and a rule is dropped at each step. Clearly if there is an infinite basic narrowing derivation for a given goal, then there is a cycle in the graph of functional dependencies. This graph can be useful to prove the termination of basic narrowing derivations for simple cases as stated by the following:

Proposition 5.15. Let $\mathcal{R}$ be a term rewriting system, $\mathcal{G}_{\mathcal{R}}$ be the corresponding graph of functional dependencies and g_0 be an equational goal. Let $\mathcal{D}$ be the set of defined function symbols occurring in g_0. If the paths in $\mathcal{G}_{\mathcal{R}}$ starting from each $f \in \mathcal{D}$ contain no cycles, then any basic narrowing derivation issuing from g_0 terminates.

PROOF. The proof is based on a well-founded complexity measure on goals which is decreased by narrowing steps. For $f \in \mathcal{D}$, let $m(f)$ denote the maximal length of the paths in $\mathcal{G}_{\mathcal{R}}$ starting from the vertex f, or zero if f is an irreducible function symbol. Since the paths starting from each $f \in \mathcal{D}$ have no loop and the number of (distinct) defined function symbols is finite, $m(f)$ is finite for all $f \in \mathcal{D}$. Let us consider a basic derivation $\langle g_0, \theta_0\rangle \leadsto_{\mathcal{R}} \ldots \leadsto_{\mathcal{R}} \langle g_p, \theta_p\rangle \ldots$ issuing from g_0, $p > 0$. We associate with each goal g_i in the derivation a set $\mathcal{H}_i = \{\langle e, u, n\rangle \mid e \in g_i, u \in \bar{O}(e), n = m(e[u])\}$, $i \geq 0$. We define the complexity $\mathcal{M}_i$ of the goal g_i as the finite multiset of natural numbers consisting of the third components of the triples in the set $\mathcal{H}_i$.

Let us define a well-founded total ordering $<_{mul}$ over multiset complexities by extending the well-founded ordering $<$ on $I\!N$ to the set $M(I\!N)$ of finite multisets over $I\!N$. The set $M(I\!N)$ is well-founded under the ordering $<_{mul}$ since $I\!N$ is well-founded under $<$ [19, 37]. Let $\mathcal{M}, \mathcal{M}'$ be multiset complexities. $\mathcal{M} <_{mul} \mathcal{M}' \leftrightarrow \exists X \subseteq \mathcal{M}, X' \subseteq \mathcal{M}'$ such that $\mathcal{M} = (\mathcal{M}' \sim X') \cup X$ and $\forall n \in X\ \exists n' \in X'.\ n < n'$.

¿From the definition of basic conditional narrowing, at each derivation step $\langle g_i, \theta_i\rangle \leadsto_{\mathcal{R}} \langle g_{i+1}, \theta_{i+1}\rangle$, the selected occurrence either comes from the nonvariable occurrences of the equations in g_0 or it comes from the right-hand side or from the condition of a rewrite rule used in a previous step. Let e and u denote, respectively, the selected equation and occurrence in g_i. By construction of $\mathcal{G}_{\mathcal{R}}$, for all i, $i \geq 0$, $\forall \langle e', u', n'\rangle \in (\mathcal{H}_{i+1} \sim \mathcal{H}_i).n' < m(e[u])$. Hence, by the definition of the ordering $<_{mul}$, it is immediate that for all i, $i \geq 0$, $\mathcal{M}_{i+1} <_{mul} \mathcal{M}_i$. From the definition of conditional basic narrowing, if there is a $k > 0$ such that every element in $\mathcal{M}_k$ is 0 then no more narrowing steps are possible. Thus, by the well-foundness of the multiset ordering $<_{mul}$ over $M(I\!N)$, only finite decreasing chains are possible and the narrowing derivation terminates. $\Box$

The following definition represents the core of our algorithm. A term rewriting system 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. Irreducible symbols are kept, while some of the terms whose main functor is a defined symbol are drastically simplified by replacing them by $\perp$. Note that we do not perform this approximation when for the function symbol f which we consider there is no cycle in $\mathcal{G}_{\mathcal{R}}$. In this case we can be more accurate and retain the subterm originating in f.

Definition 5.16. Abstract term rewriting system
Let $(\mathcal{C} \uplus \mathcal{F}, \mathcal{R})$ be a TRS. Let $\mathcal{G}_{\mathcal{R}}$ be the corresponding graph of functional dependencies. We define the abstraction of $\mathcal{R}$ as follows:
$\mathcal{R}^{\perp} = \{\lambda \rightarrow sh(\rho) \Leftarrow sh(C) \mid (\lambda \rightarrow \rho \Leftarrow C) \in \mathcal{R}\}$, *where the shell $sh(s)$ of an*

expression s is defined inductively

$$sh(s) = \begin{cases} x & \textit{if } s = x \in V \\ c(sh(t_1), \ldots, sh(t_m)) & \textit{if } s = c(t_1, \ldots, t_m), c \in \mathcal{C} \\ f(sh(t_1), \ldots, sh(t_k)) & \textit{if } s = f(t_1, \ldots, t_k), f \in \mathcal{F} \textit{ and there is no cycle} \\ & \quad \textit{in } \mathcal{G}_{\mathcal{R}} \textit{ starting from } f \\ sh(l) = sh(r) & \textit{if } s = (l = r) \\ sh(e_1), \ldots, sh(e_n) & \textit{if } s = e_1, \ldots, e_n \\ \bot & \textit{otherwise} \end{cases}$$

Note that a corresponding concretization function for abstract term rewriting systems can be easily defined [6].

Example 5.17. Let us consider the following level-canonical TRS $\mathcal{R}$ and its abstraction $\mathcal{R}^{\perp}$. We depict in Figure 1 the dependency graph built from $\mathcal{R}$. There is a cycle: $f \overset{dep}{\rightarrow} f \overset{dep}{\rightarrow} \ldots$ in the graph.

$$\begin{array}{llll} \mathcal{R} = \{ & r1) & h(X) & \rightarrow f(X). \\ & r2) & f(0) & \rightarrow 0. \\ & r3) & f(c(X)) & \rightarrow c(f(X)) \\ & & \quad \Leftarrow g(X) = X. & \\ & r4) & g(c(X)) & \rightarrow c(X). \} \end{array} \qquad \begin{array}{llll} \mathcal{R}_{\mathcal{A}} = \{ & r1_{\mathcal{A}}) & h(X) & \rightarrow f(X). \\ & r2_{\mathcal{A}}) & f(0) & \rightarrow 0. \\ & r3_{\mathcal{A}}) & f(c(X)) & \rightarrow c(\bot) \\ & & \quad \Leftarrow g(X) = X. & \\ & r4_{\mathcal{A}}) & g(c(X)) & \rightarrow c(X). \} \end{array}$$

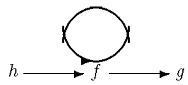

Figure 1. *Graph of functional dependencies*

In [6] we propose a generic technique of loop detection and a notion of abstract term rewriting system which is parametric w.r.t. a loop check. We also instantiate our construction with another, more accurate, loop check.

In the following we formalize the idea that abstract narrowing reduction approximates (basic) narrowing reduction with abstract states, abstract unification and abstract term rewriting systems replacing concrete states, unification and term rewriting systems.

Definition 5.18. Let $\mathcal{R}^{\perp}$ be an abstract *TRS*, and $State^{\perp}$ the abstract state domain induced by the equation description $Eqn^{\perp}$ and the substitution description $Sub^{\perp}$. We define *abstract (basic) narrowing* as a transition system $(State^{\perp}, \overset{\mathcal{A}}{\leadsto}_{\mathcal{R}^{\perp}})$ whose transition relation $\overset{\mathcal{A}}{\leadsto}_{\mathcal{R}^{\perp}} \subseteq State^{\perp} \times State^{\perp}$ is defined as follows.

Let $t, t' \in State^{\perp}$. There is an *abstract (basic) narrowing* reduction from t to t', and we write $t \overset{\mathcal{A}}{\leadsto}_{\mathcal{R}^{\perp}} t'$, iff:

1) $\exists(\lambda \rightarrow \rho \Leftarrow C) \ll \mathcal{R}^{\perp}$ such that $t = \langle g, \kappa \rangle$ and $t' = \langle g', \kappa' \rangle$ where $g' = (g \sim \{e\}) \cup \{e[\rho]_u\} \cup C$, $e \in g$, $u \in \bar{O}(e)$, $e|_u \not\equiv \bot$, $\sigma = mgu_{\mathcal{A}}(\{(e|_u)\kappa = \lambda\})$ and $\kappa' = \kappa\sigma$; or,

2) $t = \langle g, \kappa\rangle$, $\sigma = mgu_{\mathcal{A}}(g\kappa)$ and $t' = \langle true, \kappa\sigma\rangle$; or,
3) $t = t'$.

We note that Rule (3) in Definition 5.18 is only necessary because in Definition 4.1 we require that every concrete transition is approximated by an abstract transition. Since in our construction the abstract calculus in general contains less redexes than the concrete one, we have to add this generic rule.

We give an example that illustrates the definitions above.

Example 5.19. Let us consider the following level-canonical TRS $\mathcal{R}$ and its abstraction $\mathcal{R}^{\perp}$.

$$\begin{array}{llll}
\mathcal{R} = \{ & & & \\
r1)\ h(0) & \rightarrow & s(0). & \\
r2)\ h(c(0)) & \rightarrow & 0. & \\
r3)\ h(f(X)) & \rightarrow & X & \\
\quad\Leftarrow g(X) = X. & & & \\
r4)\ f(0) & \rightarrow & c(0). & \\
r5)\ f(c(X)) & \rightarrow & c(f(X)). & \\
r6)\ g(0) & \rightarrow & 0.\} &
\end{array}
\qquad
\begin{array}{lll}
\mathcal{R}^{\perp} = \{ & & \\
r1_{\mathcal{A}})\ h(0) & \rightarrow & s(0). \\
r2_{\mathcal{A}})\ h(c(0)) & \rightarrow & 0. \\
r3_{\mathcal{A}})\ h(f(X)) & \rightarrow & X \\
\quad\Leftarrow g(X) = X. & & \\
r4_{\mathcal{A}})\ f(0) & \rightarrow & c(0). \\
r5_{\mathcal{A}})\ f(c(X)) & \rightarrow & c(\perp). \\
r6_{\mathcal{A}})\ g(0) & \rightarrow & 0.\}
\end{array}$$

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. Figures 2 and 3 depict the search tree of basic narrowing and abstract basic narrowing for the (unsatisfiable) equation $h(f(Z)) = s(0)$. Let us note that the unsatisfiability of the equation cannot be determined either using methods based on loop-checking [15, 47], unification rules [14, 25], operator joinability [22], eager normalization [2, 22] or by abstract rewriting [11].

The following proposition shows that our analysis terminates.

Proposition 5.20. (Termination of the analysis) Let $\mathcal{R}$ be a TRS. The abstract transition system for $\overset{\mathcal{A}}{\leadsto}_{\mathcal{R}^{\perp}}$, state s and $State^{\perp}$ has a finite number of nodes.

Proof. Let $\mathcal{D}$ be the following abstract narrowing derivation for $\mathcal{R}^{\perp}$ and state s (i.e. a path starting from the source of the graph): $s = \langle g'_0, \theta'_0\rangle \overset{\mathcal{A}}{\leadsto}_{\mathcal{R}^{\perp}} \langle g'_1, \theta'_1\rangle \overset{\mathcal{A}}{\leadsto}_{\mathcal{R}^{\perp}} \ldots \overset{\mathcal{A}}{\leadsto}_{\mathcal{R}^{\perp}} \langle g'_n, \theta'_n\rangle \overset{\mathcal{A}}{\leadsto}_{\mathcal{R}^{\perp}} \ldots$, $n > 0$. We have to prove that the number of (different) states in the derivation is finite. We can assume that each transition $\langle g'_j, \theta'_j\rangle \overset{\mathcal{A}}{\leadsto}_{\mathcal{R}^{\perp}} \langle g'_{j+1}, \theta'_{j+1}\rangle, 0 \leq j < n$, has been proven by applying Rule (1) of Definition 5.18, as this suffices to prove the claim. In fact, Rule (3) does not lead to any different state, and Rule (2) leads to states of the form $\langle true, \kappa\rangle$ which cannot be further reduced.

Let us assume, by contradiction, that the derivation $\mathcal{D}$ above does not terminate. We associate with each abstract state $\langle g'_i, \theta'_i\rangle$ in $\mathcal{D}$ a state $\langle g_i, \theta_i\rangle$ where $[\![\langle g'_i, \theta'_i\rangle]\!] = \exists y_1 \ldots \exists y_k.\langle g_i, \theta_i\rangle$, $0 \leq i \leq n$. Define $[\![r]\!] = r'$, where the n-tuple of occurrences of $\perp$ in r is replaced by a n-tuple of distinct fresh variables in r'. We define the term

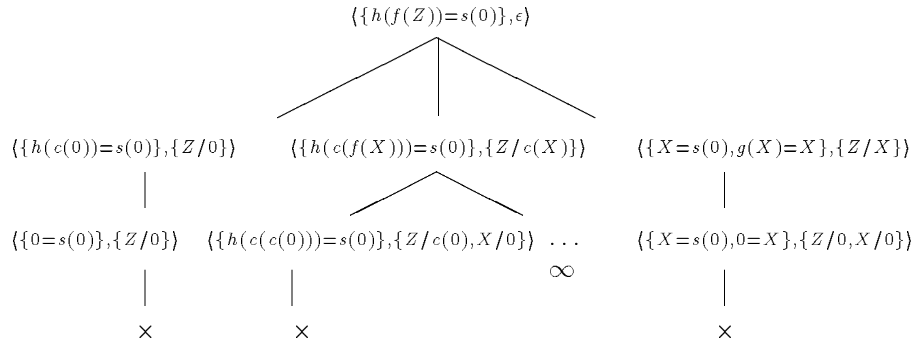

Figure 2. *Basic narrowing*

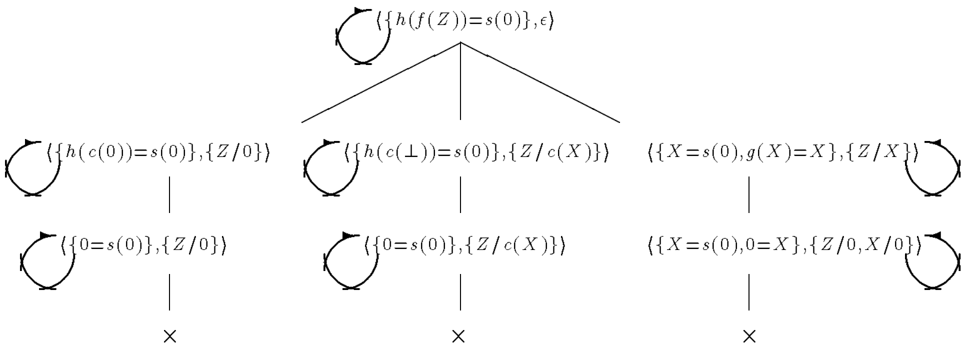

Figure 3. *Abstract basic narrowing*

rewriting system[1] $T = \{[\![r_{\mathcal{A}}]\!] \mid r_{\mathcal{A}} \in \mathcal{R}_{\mathcal{A}}\}$. By construction of T, it is clear that the graph $\mathcal{G}_T$ of functional dependencies does not contain any cycle. We can easily prove that the sequence $s = \langle g_0, \theta_0\rangle \leadsto_{\mathcal{R}} \langle g_1, \theta_1\rangle \leadsto_{\mathcal{R}} \ldots \leadsto_{\mathcal{R}} \langle g_n, \theta_n\rangle \leadsto_{\mathcal{R}} \ldots$, $n > 0$, is a basic narrowing derivation for TRS T and state s. Hence, by Proposition 5.15, this derivation terminates, which yields the required contradiction. □

In equational logic programming different strategies to detect equational unsatisfiability have been previously proposed. The distinction between defined and irreducible function symbols has been proven to be extremely useful for recognizing failures. The analyses in [14, 23, 24, 25, 35] basically exploit the idea that two terms whose outermost functors are different irreducible symbols cannot be unified. In [2, 30] an additional term rewriting rule is used to strengthen the analysis and further simplify the equations. The notion of *operator joinability* in [22] provides the basis for another method to detect the unsolvability of an equation set. In the following we formalize these standard methods following the approach of abstracting transition systems.

5.2. Non-unifiability Analysis

This method essentially relies on the application of two of the rules of the standard unification algorithm [9] which get restricted to deal only with irreducible symbols; namely, two terms which are headed by different irreducible symbols cannot be unified (*failure rule*) and, when headed by the same irreducible function symbol, they are equal if, and only if, their respective arguments are equal (*decomposition rule*) [2, 14, 30, 32]. These rules can be built in within the (equational) unification algorithm, automatically generated and added to the equational theory [35], or handled as 'eager' reduction rules as described in [21, 25]. In the following we show that this method can be formalized in our framework and that the corresponding analysis is less precise than the one which we have defined in Section 5.1.

We need not establish here any different domain of descriptions, since this analysis can be carried out in the same way described by Definition 5.18, except for a different approximation to the TRS $\mathcal{R}$. We call $\mathcal{R}^u$ this new abstraction of $\mathcal{R}$ as formalized in the following definition and denote by $\overset{\mathcal{A}}{\leadsto}_{\mathcal{R}^u}$ the resulting reduction relation, which we call *unifiability* reduction.

Definition 5.21. Abstract term rewriting system
Let $(\Sigma, \mathcal{R})$ be a TRS. We define the abstraction $\mathcal{R}^u$ of $\mathcal{R}$ as follows:
$\mathcal{R}^u = \{f(\bot, \ldots, \bot)^2 \to \bot \mid (f(t_1, \ldots, t_n) \to \rho \;\Leftarrow\; C) \in \mathcal{R}\}$.

The following example illustrates how the *non-unifiability* analysis works.

Example 5.22. Consider the TRS $\mathcal{R}$ from Example 5.19 with initial state $s = \langle\{s(f(Y)) = c(Y)\}, \epsilon\rangle$. In Figure 4, the abstract transition graph on the right describes the *non-unifiability* analysis for $\mathcal{R}^u = \{h(\bot) \to \bot,\ f(\bot) \to \bot,\ g(\bot) \to \bot\}$ and (initial state) s. The left graph represents the corresponding

[1] We note that the rules $(\lambda \to \rho \;\Leftarrow\; C) \in T$ do not satisfy the condition of non-extra variables in the right-hand-side of the rules, i.e. $Var(\rho) \not\subseteq Var(\lambda)$.

[2] We assume that f has the same arity as the abstracted function symbol.

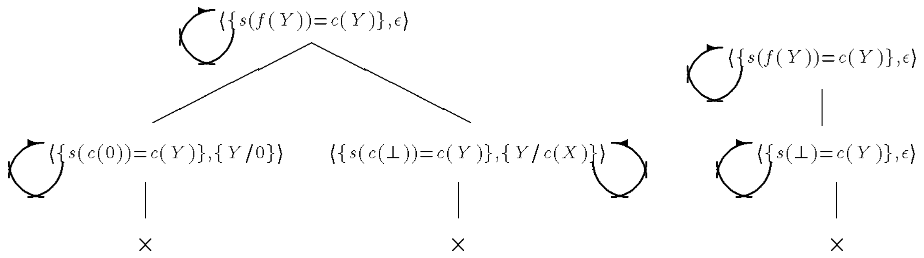

Figure 4. *Abstract (basic) narrowing* analysis *vs. non-unifiability* analysis

abstract (basic) narrowing analysis of ours. Both analyses show the unsatisfiability of the equation since none of the graphs contain any leaves of the form $\langle true, \kappa\rangle$.

Proposition 5.23. (Termination of the analysis) Let $\mathcal{R}$ be a TRS. The abstract transition system for $\overset{\mathcal{A}}{\leadsto}_{\mathcal{R}^u}$, state s and $State^{\perp}$ has a finite number of nodes.

Proof. Immediate, since the number of function symbols in s is finite and (at least) one function symbol is dropped at each derivation step. □

We present now an alternative characterization of the non-unifiability analysis. Let us define the function $|\ |$, a slight variant of the function *shell* in Definition 5.16, which assigns to any expression (term, or equation, or equation set) an approximation of it (in the sense of Definition 5.7) which is obtained by inductively replacing by $\perp$ any term whose outermost functor is not an irreducible symbol, i.e.,

$$|s| = \begin{cases} x & \text{if } s = x \in V \\ c(|t_1|, \ldots, |t_m|) & \text{if } s = c(t_1, \ldots, t_m),\, c \in C \\ |l| = |r| & \text{if } s = (l = r) \\ |e_1|, \ldots, |e_n| & \text{if } s = e_1, \ldots, e_n \\ \perp & \text{otherwise} \end{cases}$$

For *non-unifiability* analyses it is immediate to see that, for any program $\mathcal{R}$ with initial state $\langle g, \epsilon\rangle$, the associated transition graph has a single leaf, which is either $\langle true, \epsilon\rangle$ or contains a non-unifiable set of equations which coincides with $|g|$. Redundancy in the graph is avoided by collapsing all nodes, except for $\langle true, \epsilon\rangle$, in the source of the graph. This is easily obtained by taking $|g|$ as the source of the graph. The correctness of this method, as stated by the following theorem, is straightforward.

Theorem 5.24. Let $\mathcal{E}$ be a level-canonical Horn equational theory. If the abstract transition system for $\overset{\mathcal{A}}{\leadsto}_{\emptyset}$, state $\langle |g|, \epsilon\rangle$ and $State^{\perp}$ is finitely failed then g is unsatisfiable in $\mathcal{E}$.

The following analysis is based on the idea that more precision may be achieved by firstly reducing unificands up to their irreducible form and then applying the *non-unifiability* analysis to the resulting expression [2, 22].

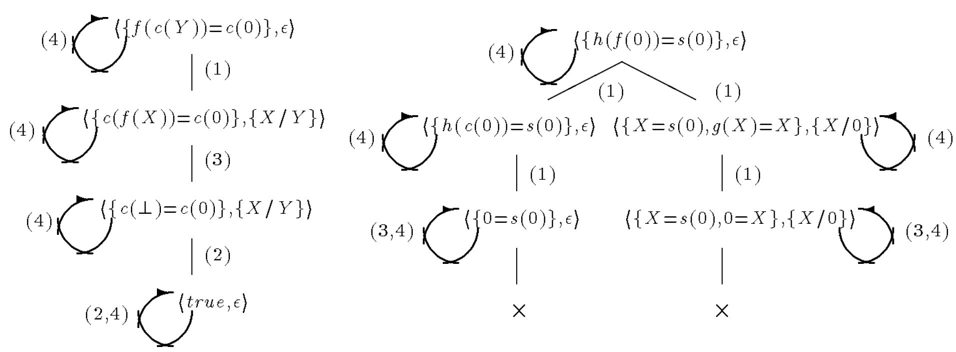

Figure 5. *Eager normalization* analyses

5.3. Eager Normalization Analysis

Definition 5.25. Let $\mathcal{R}$ be a *TRS*. *Eager normalization* reduction is a relation

$\xrightarrow{\mathcal{A}}_{\mathcal{R}} \subseteq State^{\perp} \times State^{\perp}$, defined as follows.

Let $t, t' \in State^{\perp}$. $t \xrightarrow{\mathcal{A}}_{\mathcal{R}} t'$ iff:

1) $\exists(\lambda \to \rho \Leftarrow C) \ll \mathcal{R}$ such that $t = \langle g, \kappa \rangle$ and $t' = \langle g', \kappa' \rangle$ where $g' = (g \sim \{e\}) \cup \{e[\rho]_u\} \cup C$, $e \in g$, $u \in \bar{O}(e)$, $(e|_u)\kappa \equiv \lambda\sigma$ and $\kappa' = \kappa\sigma$; or,
2) $t = \langle g, \kappa \rangle$, $\sigma = mgu_{\mathcal{A}}(g\kappa)$ and $t' = \langle true, \kappa\sigma \rangle$; or,
3) $t = \langle g, \kappa \rangle$, $t' = \langle |g\kappa|, \kappa \rangle$ and rules (1) and (2) do not apply to t; or,
4) $t = t'$

Some example analyses based on *eager normalization* are now given.

Example 5.26. Consider again the TRS $\mathcal{R}$ from Example 5.19. Applying the *eager normalization* analysis for $\mathcal{R}$ with the initial states $\langle \{f(c(Y)) = c(0)\}, \epsilon \rangle$ and $\langle \{h(f(0)) = s(0)\}, \epsilon \rangle$ respectively results in the graphs which are depicted in Figure 5. For the sake of simplicity, the number of the rule which proves each transition labels the corresponding edge. The analysis on the right proves the equation $h(f(0)) = s(0)$ to be unsatisfiable. The analysis on the left does not detect the unsatisfiability of $f(c(Y)) = c(0)$. We note that the *non-unifiability* analysis would not recognize the unsatisfiability of both equations.

It is immediate to see that the *eager normalization* analysis is strictly more powerful than the *non-unifiability* analysis, in the sense that it is able to recognize any failure that the latter does. Of course, earlier detection of failures may be possible by testing the nodes of the graph for *non-unifiability* before carrying on with the normalization, as in [30]. Efficiency can also be gained by making "deterministic" rewriting (leading to no branching in the search tree) instead of "nondeterministic" rewriting (leading to branching in the search tree), as in [2]. We note that TRS's are required to be decreasing [20, 42], since this is essential to ensure the termination of the analysis.

Also note that the *abstract (basic) narrowing* and the *eager normalization* analyses are not comparable in the sense that they do not subsume one another. It would be worth investigating how to integrate the abstract basic narrowing analysis and the eager normalization analysis, (i.e. how to approximate normalizing basic narrowing [30, 44, 46]) in order to define more accurate analyses.

Example 5.27. Consider the level-canonical TRS $\mathcal{R} = \{g(X) \to 0,\ g(s(X)) \to g(X)\}$ with initial state $s = \langle\{g(Y) = 1\}, \epsilon\rangle$. The unsatisfiability of the equation $g(Y) = 1$ in s is detected by *eager normalization* while *abstract (basic) narrowing* does not detect it.

Proposition 5.28. (Termination of the analysis) Let $\mathcal{R}$ be a decreasing TRS. The abstract transition system for $\xrightarrow{\mathcal{A}}_{\mathcal{R}}$, state s and $State^{\perp}$ has a finite number of nodes.

Proof. Immediate from the fact that, for decreasing conditional rewriting systems with finite number of rules, the rewrite relation and the set of normal forms are recursive [19, 37, 42]. □

Let us now formalize the analysis of unsatisfiability in [22] in the framework presented in this paper.

5.4. Non-Joinability Analysis

In the following we extend the analysis of unsatisfiability in [22] to handle conditional TRS as well. Let $\mathcal{R}$ be a TRS and $f(\tilde{t}) = g(\tilde{s})$ be one of the equations to be tested for unsatisfiability. The test in [22] can be thought of as the detection of the fact that the outermost function symbols f and g in the equation are not *joinable* using the reduction rules of an abstract term rewriting system $\mathcal{R}^{\top}$ that we describe in the following. The descriptions of terms in this analysis are extremely simple, just function symbols. Let $top(t)$ denote the outermost function symbol of the term t if $t \notin V$, t otherwise. By abuse, if C is a set of equations, $top(C)$ is the set of equations $\{top(t) = top(s) \mid (t = s) \in C\}$.

Definition 5.29. Let $\mathcal{T} = (\tau(\Sigma \cup V), \leq)$ and $\mathcal{T}^{\top} = (\Sigma, \leq)$. The *term description* is $\langle \mathcal{T}^{\top}, \gamma, \mathcal{T}\rangle$ where $\gamma : \mathcal{T}^{\top} \to \wp\,\mathcal{T}$ is defined by: $\gamma(f) = \{t \in \mathcal{T} \mid top(t) = f\}$

Definition 5.30. Let Eqn be the set of (unquantified) finite sets of equations over $(\tau(\Sigma \cup V), \leq)$ and $Eqn^{\top}$ be the set of (unquantified) finite sets of equations over Σ. The equation description is $\langle (Eqn^{\top}, \sqsubseteq), \gamma, (Eqn, \sqsubseteq)\rangle$, where $\gamma : Eqn^{\top} \to \wp\, Eqn$ is defined by: $\gamma(g') = \{g \in Eqn \mid top(g) \supseteq g'\}$.

In the above definition $\gamma(g') = true \Rightarrow g' \equiv true$. Also note that, whenever the alphabet Σ is finite, then so is $Eqn^{\top}$.

Definition 5.31. Let $Sub^{\top} = \{\epsilon\}$. The *substitution description* is $\langle (Sub^{\top}, \leq), \gamma, (Sub, \leq)\rangle$, where $\gamma : Sub^{\top} \to \{Sub\}$ is defined by: $\gamma(\epsilon) = Sub$.

Definition 5.32. Abstract term rewriting system
Let $(\Sigma, \mathcal{R})$ *be a TRS. We define the abstraction* $\mathcal{R}^\top$ *of* $\mathcal{R}$ *as follows:* $\mathcal{R}^\top = \{r_\mathcal{A} \mid r_\mathcal{A} \in abs(r) \ \wedge \ r \in \mathcal{R}\}$, *where*

$$
\begin{array}{rl}
r_\mathcal{A} \in abs(r) \Leftrightarrow & r = (f(\tilde{t}) \rightarrow \rho \ \Leftarrow \ C) \ \wedge \\
& top(\rho) \not\equiv f \ \wedge \\
& r^\top = (f \rightarrow top(\rho) \ \Leftarrow \ top(C)) \ \wedge \\
& r_\mathcal{A} = r^\top \sigma \ \wedge \\
& \sigma = \{x_1 \mapsto f_1, \ldots, x_n \mapsto f_n\} \ \wedge \\
& \{x_1, \ldots, x_n\} = Var(r^\top) \ \wedge \\
& f_1, \ldots, f_n \in \Sigma \ \wedge \\
& (top(\rho) \in V \Rightarrow top(\rho)\sigma \not\equiv f).
\end{array}
$$

Example 5.33. Consider the TRS $\mathcal{R} = \{g(f(X)) \rightarrow X \Leftarrow g(X) = X, f(X) = Z, g(0) = Z.\}$. Then the abstraction $\mathcal{R}^\top$ of $\mathcal{R}$ is the set of rules:

$$
\begin{array}{llllll}
\mathcal{R}^\top = \{ & g & \rightarrow & 0 & \Leftarrow & g = 0, \ f = 0. \\
& g & \rightarrow & 0 & \Leftarrow & g = 0, \ f = g, \ g = g. \\
& g & \rightarrow & 0 & \Leftarrow & g = 0, \ f = f, \ g = f. \\
& g & \rightarrow & f & \Leftarrow & g = f, \ f = 0, \ g = 0. \\
& g & \rightarrow & f & \Leftarrow & g = f, \ f = g, \ g = g. \\
& g & \rightarrow & f & \Leftarrow & g = f, \ f = f.\}
\end{array}
$$

In the following we define *joinability* in terms of an appropriate abstract reduction relation for abstract TRS $\mathcal{R}^\top$.

Definition 5.34. Let $\mathcal{R}^\top$ be an abstract *TRS*, and $State^\top$ be the abstract state domain induced from the equation description $Eqn^\top$ and the substitution description $Sub^\top$. *Joinability* reduction is a relation $\xrightarrow{\mathcal{A}}_{\mathcal{R}^\top} \ \subseteq State^\top \times State^\top$, defined as follows.

Let $t, t' \in State^\top$. $t \xrightarrow{\mathcal{A}}_{\mathcal{R}^\top} t'$ (we also say that there is a *joinability* reduction from t to t') iff:

1) $\exists(\lambda \rightarrow \rho \ \Leftarrow \ C) \ll \mathcal{R}^\top$ such that $t = \langle g, \epsilon \rangle$ and $t' = \langle g', \epsilon \rangle$ where $g' = (g \sim \{e\}) \cup \{e[\rho]_u\} \cup C$, $e \in g$, $u \in O(e)$ and $e|_u = \lambda$; or,
2) $t = \langle g, \epsilon \rangle$, g syntactically unifies and $t' = \langle true, \epsilon \rangle$; or,
3) $t = t'$

Example 5.35. Consider the level-canonical TRS $\mathcal{R} = \{h(g(Y)) \rightarrow h(Y), \ f(0) \rightarrow 0, \ f(c(X)) \rightarrow X \Leftarrow g(X) = X\}$ and the equation $f(Z) = h(W)$. The graph at the top of Figure 6 describes the analysis for $\mathcal{R}^\top = \{f \rightarrow 0, f \rightarrow 0 \Leftarrow g = 0, f \rightarrow c \Leftarrow g = c, f \rightarrow g \Leftarrow g = g, f \rightarrow h \Leftarrow g = h\}$ with initial state $\langle\{f = h\}, \epsilon\rangle$, where $f = h$ approximates $f(Z) = h(W)$. The analysis shows that the equation is unsatisfiable. Consider now the equation $f(0) = g(0)$. The graph at the bottom of Figure 6 describes the analysis for $\mathcal{R}^\top$ with initial state

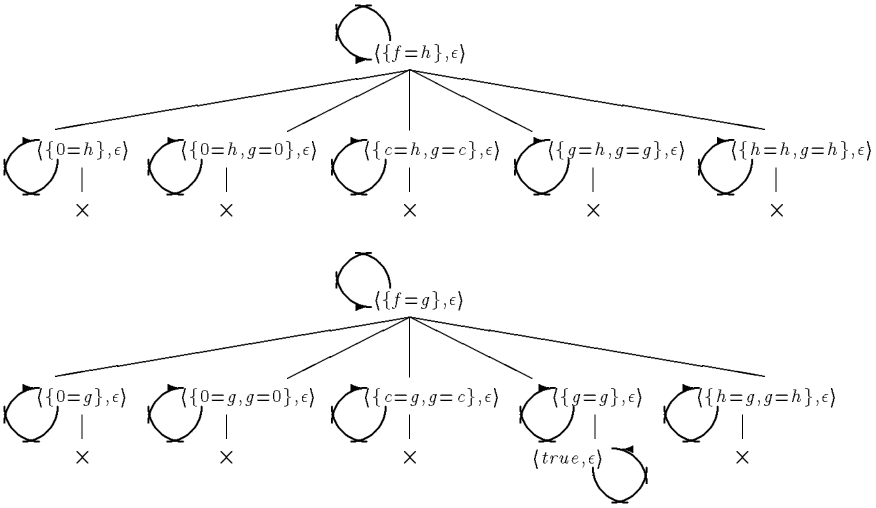

Figure 6. *Non-joinability* analyses

$\langle\{f = g\}, \epsilon\rangle$, where $f = g$ approximates $f(0) = g(0)$. The analysis does not detect the unsatisfiability.

It is important to note that the abstract TRS $\mathcal{R}^\top$ may not be terminating even if the concrete TRS is. In fact, it is easy to note that Rule (1) from Definition 5.34 can be applied infinitely many times when reducing a set of equations. However, since $Eqn^\top$ is finite, then the number of possibly different states in the abstract derivation is also finite. In order to cut loops when exploring the nodes of a *joinability* transition graph, it is sensible to use some simple loop-checking technique which takes into account (a part of) the history of the computation of the graph to compare states in the abstract derivations. If two states in a derivation are detected to be equivalent, the derivation can be safely pruned.

Proposition 5.36. (Termination of the analysis) Let $\mathcal{R}$ be a TRS. The abstract transition system for $\xrightarrow{\mathcal{A}}_{\mathcal{R}^\top}$, abstract state s and $State^\top$ has a finite number of nodes.

Proof. Immediate from the fact that $Eqn^\top$ is finite and hence $State^\top$ is also finite. □

The following example demonstrates the fact that the analysis of *non-joinability* is able to recognize unsatisfiability of equations in cases where the other methods do not and vice versa.

Example 5.37. Consider the TRS $\mathcal{R}$ from Example 5.35. The unsatisfiability of the equation $f(Z) = h(W)$ is not recognized by applying either the *abstract (basic) narrowing* analysis or the *eager normalization* analysis. Instead, the analysis of *non-joinability* does detect the unsatisfiability of the equation. On the other hand, the unsatisfiability of the equation $c(c(Z)) = c(g(W))$, which is not decided by the analysis of *non-joinability*, can be recognized by simply applying the *non-unifiability* analysis. The unsatisfiability of the equation $f(0) = g(0)$ is not detected by *non-joinability* but it is decided by *eager normalization*.

All these three methods are thus complementary since they do not subsume one another. Some enhanced test may be developed by properly combining them. This can prove useful to strengthen the construction of the dependency graph of ours to achieve more accurate approximations. There is therefore much room for gaining precision when analizing the unsatisfiability of a set of equations. The following example, which is mainly borrowed from [22], outlines this fact.

Example 5.38. Consider the level-canonical TRS $\mathcal{R} = \{f(0) \rightarrow d(0),\ \ a(d(X)) \rightarrow b(X,X),\ \ b(d(0),Y) \rightarrow b(0,f(Y)),\ \ b(f(X),Y) \rightarrow b(X,f(Y))\}$ and the equation $a(f(1)) = b(V,0)$. Narrowing produces infinitely many nonsubsuming equations when considering $b(V,E)$, which prevents the use of any loop-checking strategy we are acquainted with to prune the tree. The unsatisfiability of this equation cannot be determined either using any of the analyses developed before. Some enhanced analysis could detect the fact that the only way for a term headed by a to join a term headed by b is for the argument of the former to reach the form $d(X)$, whereas the resulting equation $f(1) = d(X)$ may easily be proven unsatisfiable using the approximated narrowing of ours.

6. CORRECTNESS OF THE ANALYSES

In the following we prove the correctness of the analyses which we have formalized so far. Our method allows to prove the theorems in a simple way, by making use of Theorem 4.6, proving a simple lemma which depends on the specific analysis.

In order to prove Lemma 6.3 we need to prove some useful technical properties first. Let us denote by $\propto_{AN}$ / $\propto_{NU}$ / $\propto_{EN}$ / $\propto_{NJ}$ the approximation relation defined in the previous section over the states of the Abstract Narrowing/Non-Unifiability/Eager-Normalization/Non-Joinability analysis, respectively.

Lemma 6.1. Let g, g' be (unquantified) finite sets of equations over $\tau(\Sigma \cup V)$ and $\tau(\Sigma \cup V \cup \{\bot\})$, respectively. Let $\theta, \sigma \in Sub$ and $\theta', \sigma' \in Sub^{\bot}$, where $Dom(\theta) \cap Dom(\sigma) = \emptyset$ and $Dom(\theta') \cap Dom(\sigma') = \emptyset$. If $\theta' \propto_{AN} \theta$, $\sigma' \propto_{AN} \sigma$, $g' \propto_{AN} g$, then
1) $\theta'\sigma' \propto_{AN} \theta\sigma$
2) $mgu_{\mathcal{A}}(g' \wedge \theta') \propto mgu(g \wedge \theta)$

PROOF.

1. We have to prove that $\theta\sigma \Rightarrow [\![\theta'\sigma']\!]$. Since $\theta' \propto_{AN} \theta$ and $\sigma' \propto_{AN} \sigma$, then $\theta \Rightarrow [\![\theta']\!]$ and $\sigma \Rightarrow [\![\sigma']\!]$. Hence $\theta\sigma \Leftrightarrow \theta \wedge \sigma$ (since $Dom(\theta) \cap Dom(\sigma) = \emptyset$) $\Rightarrow [\![\theta']\!] \wedge [\![\sigma']\!] \Leftrightarrow [\![\theta'\sigma']\!]$ (since $Dom(\theta') \cap Dom(\sigma') = \emptyset$).
2. By Lemma 5.12 it suffices to show that $g \wedge \theta \Rightarrow [\![g' \wedge \theta']\!]$. Since $g' \propto g$ and $\theta' \propto \theta$ then $g \Rightarrow [\![g']\!]$ and $\theta \Rightarrow [\![\theta']\!]$. Thus $g \wedge \theta \Rightarrow [\![g']\!] \wedge [\![\theta']\!] \Leftrightarrow [\![g' \wedge \theta']\!]$.

□

Lemma 6.2. Let $s = \langle g, \theta \rangle$, $s' = \langle g', \theta' \rangle$ and $s' \propto s$ (where $\propto = \propto_{AN}$ / $\propto_{NU}$ / $\propto_{EN}$ / $\propto_{NJ}$).

Assume that $s \leadsto_{\mathcal{R}} t$ (by a narrowing step) where $u \in \bar{O}(e)$ is the occurrence in $e \in g$ selected for this reduction. If $\nexists\ e' \in g'$ such that $e' \propto e$ and $u \in \bar{O}(e')$ then $s' \propto t$ (where $\propto = \propto_{AN}$ / $\propto_{NU}$ / $\propto_{EN}$ / $\propto_{NJ}$).

PROOF. Assume that $s \leadsto_{\mathcal{R}} t$ where $t = \langle (g \sim \{e\}) \cup \{e[\rho]_u\} \cup C, \theta\sigma\rangle = \langle g_t, \theta_t\rangle$, $e \in g$, $u \in \bar{O}(e)$, $(\lambda \rightarrow \rho \Leftarrow C) \ll \mathcal{R}$, $\sigma = mgu(\{(e|_u)\theta = \lambda\})$.

We have to prove that $g' \propto g_t$ and $\theta' \propto \theta_t$.

- **Abstract Basic Narrowing Analysis**:

($g' \propto g_t$) There are two cases. Either there does not exist $e' \in g'$ such that $e' \propto e$ or there exists one such e'. Let us consider the former case first.
Assume that $\nexists\ e' \in g'$ such that $e' \propto e$ and $u \in \bar{O}(e)$. Then $g' \propto (g \sim \{e\})$ and, since $(g \sim \{e\}) \subseteq (g \sim \{e\}) \cup \{e[\rho]_u\} \cup C$ then, by Definition 5.7, $g' \propto g_t$.
Let us consider the second case now. Thus, assume that $\exists e' \in g'$ such that $e' \propto e$. Then $u \notin \bar{O}(e')$ and $u \in \bar{O}(e)$. Since $u \notin \bar{O}(e')$, then (by Lemma 5.1) $(e'|_{u'}) \equiv \perp$, where $u' < u$. Hence $e[\rho]_u \in upw(\{e'\})$ and $g_t = ((g \sim \{e\}) \cup \{e[\rho]_u\} \cup C) \in upw(\{g'\})$, that is $g' \propto g_t$.

($\theta' \propto \theta_t$) Since $\theta' \propto \theta$ then $\theta \Rightarrow [\![\theta']\!]$. Moreover, $\theta \wedge \sigma = \theta\sigma$ since $Dom(\theta) \cap Dom(\sigma) = \emptyset$. Thus $\theta\sigma \Rightarrow [\![\theta']\!]$, i.e. $\theta' \propto \theta_t$.

The cases of *Eager Normalization* and *Non-unifiability* are completely similar, since the descriptions of terms, equations and substitutions are the same.

- **Non-Joinability Analysis**: Note that if $u \notin \bar{O}(e')$ then $u \neq \{\Lambda, 1, 2\}$.

($g' \propto g_t$) Since $g' \propto g \implies g \in \gamma(g')$ and $\gamma(g') = \{g \in Eqn \mid top(g) \supseteq g'\}$ then $g' \propto ((g \sim \{e\}) \cup \{e[\rho]_u\} \cup C) = g_t$, since $top(e) = top(e[\rho]_u)$ and $top(g_t) \supseteq top(g) \supseteq g'$.

($\theta' \propto \theta_t$) $\theta' = \epsilon$. Hence $\theta' \propto \theta\sigma = \theta_t$.

□

The following lemma essentially states that abstract (basic) narrowing reduction approximates (basic) narrowing reduction.

Lemma 6.3. Let $s' \propto s$ (where $\propto = \propto_{AN}$ / $\propto_{NU}$ / $\propto_{EN}$ / $\propto_{NJ}$). If there is a (basic) narrowing reduction from s to t, then there is an abstract narrowing *reduction from s' to some t' such that $t' \propto t$ (where $\propto = \propto_{AN}$ / $\propto_{NU}$ / $\propto_{EN}$ / $\propto_{NJ}$).*

PROOF. We only prove the case of $\propto_{AN}$, since all the other cases are trivial. Let us assume $s = \langle g, \theta\rangle$ and $s' = \langle g', \kappa\rangle$. Since $s' \propto s$, then $g' \propto g$ and $\kappa \propto \theta$. There are two cases to consider.

1. Assume that BCN unification rule has been applied. Then $t = \langle true, \theta\sigma\rangle$, where $\sigma = mgu(g\theta)$. Since $g' \propto g$ and $\kappa \propto \theta$ then $g \Rightarrow [\![g']\!]$ and $\theta \Rightarrow [\![\kappa]\!]$. Thus, $g \wedge \theta \Rightarrow [\![g' \wedge \kappa]\!]$. Hence, if there exists $mgu(g \wedge \theta)$ then there exists $mgu_{\mathcal{A}}(g' \wedge \kappa)$ and $[\![\kappa \circ mgu_{\mathcal{A}}(g'\kappa)]\!] \Leftrightarrow [\![mgu_{\mathcal{A}}(g' \wedge \kappa)]\!]$, where $mgu_{\mathcal{A}}(g' \wedge \kappa) \propto mgu(g \wedge \theta)$ (by Lemma 6.1(2)) $\Leftrightarrow \theta \circ mgu(g\theta) = \theta\sigma$.
2. Assume that BCN narrowing rule has been applied. Then $t = \langle (g \sim \{e\}) \cup \{e[\rho]_u\} \cup C, \theta\sigma\rangle$, where $e \in g$, $u \in \bar{O}(e)$, $(\lambda \rightarrow \rho \Leftarrow C) \ll \mathcal{R}$ and $\sigma = mgu(\{(e|_u)\theta = \lambda\})$. We consider two cases.
   (a) Assume that there exists $e' \in g'$, $e' \propto e$ and $u \in \bar{O}(e')$. Thus it is immediate that $(e'|_u) \propto (e|_u)$. By Lemma 6.1(2), there exists $mgu_{\mathcal{A}}(\{(e'|_u) =$

$\lambda \wedge \kappa\}) \propto mgu(\{(e|_u) = \lambda \wedge \theta\}) \Leftrightarrow \theta \circ mgu(\{(e|_u)\theta = \lambda\}) = \theta\sigma$. Let $\sigma_{\mathcal{A}} = mgu_{\mathcal{A}}(\{(e'|_u)\kappa = \lambda\})$. Then $\theta\sigma \Rightarrow [\![mgu_{\mathcal{A}}(\{(e'|_u) = \lambda \wedge \kappa\})]\!] \Leftrightarrow [\![mgu_{\mathcal{A}}(\{(e'|_u)\kappa = \lambda \wedge \kappa\})]\!] \Leftrightarrow [\![\kappa \wedge \sigma_{\mathcal{A}}]\!] \Leftrightarrow [\![\kappa\sigma_{\mathcal{A}}]\!]$ (since $Dom(\kappa) \cap Dom(\sigma_{\mathcal{A}}) = \emptyset$), i.e. $\kappa\sigma_{\mathcal{A}} \propto \theta\sigma$. By case (1) of Definition 5.18, $t' = \langle (g' \sim \{e'\}) \cup \{e'[sh(\rho)]_u\} \cup sh(C), \kappa\sigma_{\mathcal{A}}\rangle$. From Definition 5.16[3] it is immediate that $sh(\rho) \propto \rho$ and $sh(C) \propto C$. Hence $t' \propto t$.

(b) If there does not exist $e' \in g'$ such that $e' \propto e$ and $u \in \bar{O}(e')$ then the lemma follows immediately from Lemma 6.2 and the rule $t \overset{\mathcal{A}}{\rightsquigarrow}_{\mathcal{R}^{\perp}} t$ of abstract basic narrowing.

□

As a corollary of Lemma 6.3 and of Theorem 4.6 we obtain the correctness of all the analyses we have defined so far.

Corollary 6.4. (Correctness of the analyses) Let $\mathcal{E}$ be a level-canonical Horn equational theory and g an equation set. If the abstract transition system for $\overset{\mathcal{A}}{\hookrightarrow}_T = \overset{\mathcal{A}}{\rightsquigarrow}_{\mathcal{R}^{\perp}}$ $(\overset{\mathcal{A}}{\rightsquigarrow}_{\mathcal{R}^u} / \overset{\mathcal{A}}{\longrightarrow}_{\mathcal{R}} / \overset{\mathcal{A}}{\rightarrow}_{\mathcal{R}^{\top}})$ and initial state $\langle g', \epsilon\rangle$ (where $g' \propto g$) is finitely failed then g is unsatisfiable in $\mathcal{E}$.

7. INCREMENTAL EQUATIONAL ANALYZER

In this section we describe an incremental algorithm for analyzing the unsatisfiability of sets of equations. In this context, incrementality [3, 28] means that the problem resulting from adding a new set of equations (constraints) does not require the repetition of all the work already done. The kernel of this algorithm is a calculus which is based on abstract narrowing reduction as described in Section 5. The calculus explores with a depth-first strategy the nodes of the abstract transition system for the accumulated constraint which is incrementally built as long as new equations are added. Since the number of nodes to be visited is finite, a depth-first strategy is fair, i.e. every node of the abstract transition system will be eventually visited. We show the incremental calculus for the case of the Abstract Basic Narrowing analysis. It is easy to modify it for the other example analyses.

Definition 7.1. approximated $\mathcal{N}$arrowing $\mathcal{C}$alculus ($a\mathcal{NC}$)
Let $(\Sigma, \mathcal{R}^{\perp})$ be an abstract TRS. An $a\mathcal{NC}$-state is a list of pairs $(s, \mathcal{S})$, where $s = \langle g, \kappa\rangle \in State^{\perp}$ and $\mathcal{S}$ is a set. The elements of the set are equivalence classes of the triples $\langle e, u, r\rangle$ wrt renaming, where e is a Σ-equation, u is an occurrence of e and r is (a variant of) a reduction rule in $\mathcal{R}^{\perp}$. $\langle\rangle$ is a distinguished element of the sets. List constructors are denoted by [ ] and $\bullet$. The $a\mathcal{NC}$ transition relation is defined as follows:

[3] By Definition 4.2 the (standardised apart) clause renaming which is chosen for abstract reduction is irrelevant since we always get an element in the same equivalence class. We thus assume for the sake of simplicity that the clause renaming chosen for the abstract narrowing reduction is the same as in the (basic) narrowing reduction.

Unification Rule

$$(1) \quad \frac{\langle\rangle \notin \mathcal{S} \;\wedge\; \sigma = mgu_{\mathcal{A}}(g\kappa)}{(\langle g,\kappa\rangle,\mathcal{S}) \bullet L \rightarrow_{a\mathcal{NC}} (\langle true, \kappa\sigma\rangle, \emptyset) \bullet (\langle g,\kappa\rangle, \mathcal{S} \cup \{\langle\rangle\}) \bullet L}$$

Narrowing Rule

$$(2) \quad \frac{\begin{array}{l} e \in g \;\wedge\; u \in \bar{O}(e) \;\wedge\; e|_u \not\equiv \perp \;\wedge\; \langle e,u,r\rangle \notin \mathcal{S} \;\wedge \\ r = (\lambda \rightarrow \rho \;\Leftarrow\; C) \ll \mathcal{R}^{\perp} \;\wedge\; \sigma = mgu_{\mathcal{A}}(\{(e|_u)\kappa = \lambda\}) \end{array}}{\begin{array}{l} (\langle g,\kappa\rangle,\mathcal{S}) \bullet L \rightarrow_{a\mathcal{NC}} \\ (\langle (g \sim \{e\}) \cup \{e[\rho]_u\} \cup C, \kappa\sigma\rangle, \emptyset) \bullet (\langle g,\kappa\rangle, \mathcal{S} \cup \{\,\langle e,u,r\rangle\}) \bullet L \end{array}}$$

Removal Rule

$$(3) \quad \frac{\text{(1) and (2) do not apply}}{(s,\mathcal{S}) \bullet L \rightarrow_{a\mathcal{NC}} L}$$

Roughly speaking, the abstract state in the left part of a pair $(s, \mathcal{S})$ represents a node of the abstract transition system described in Section 5. The right part of the pair records the redexes of the equations in the abstract state which are reduced together with the corresponding applied rule. $\langle\rangle$ represents the fact that an abstract mgu of the equations in the state has been found. The list representing an execution state of the above $a$pproximated $\mathcal{N}$arrowing $\mathcal{C}$alculus is treated as a stack to emulate a depth-first strategy. The derivations can be represented as a finite tree. Leaves which have already been visited are simply removed. The calculus can be slightly improved if *failed* nodes are recognized and removed from the list. A *failed* node is any node $\langle g, \kappa\rangle$ such that one equation $e \in g$ is not (abstractly) unifiable and no abstract narrowing step can be applied to any of the occurrences in e. Our analysis can only detect unsatisfiability if no leaf of the tree has the form $\langle true, \kappa\rangle$.

Definition 7.2. Behaviour of the $a\mathcal{NC}$ calculus
Define the (set valued) function $\mathcal{N}_a$: $L' \in \mathcal{N}_a(L)$ if $L \rightarrow^*_{a\mathcal{NC}} (\langle true, \kappa\rangle, \emptyset) \bullet L'$

We note that the transition system in Definition 7.1 can be made deterministic by introducing a lexicographic ordering over the tuples $\langle e, u, r\rangle$ and always reducing the least tuple in the order. In this case, $\mathcal{N}_a(L)$ would be a singleton.

We can now define an *incremental Equational Analyzer (iEA)*:

Definition 7.3. An *iEA*-state is a pair $\langle c, L\rangle$, where c is a constraint and L is an $a\mathcal{NC}$-state. The empty *iEA*-state is $\langle \emptyset, [\,]\rangle$.

Definition 7.4. (*iEA transition relation* $\stackrel{\tilde{c}}{\hookrightarrow}_{iEA}$)

$$\frac{L' \in \mathcal{N}_a(merge(L, \tilde{c}))}{\langle c, L\rangle \;\stackrel{\tilde{c}}{\hookrightarrow}_{iEA}\; \langle c \cup \tilde{c}, L'\rangle}$$

where
$merge([\,], \tilde{c}) = [(\langle \tilde{c}, \epsilon\rangle, \emptyset)]$

$merge([(\langle g, \epsilon\rangle, \mathcal{S})], \tilde{c}) = [(\langle g \cup \tilde{c}, \epsilon\rangle, \mathcal{S} \sim \{\langle\rangle\})]$
$merge((\langle g, \kappa\rangle, \mathcal{S}) \bullet L, \tilde{c}) = (\langle g \cup \tilde{c}, \kappa\rangle, \mathcal{S} \sim \{\langle\rangle\}) \bullet merge(L, \tilde{c}) \quad$ if $L \neq [\,]$.

Roughly speaking, the function $merge(L, \tilde{c})$ in Definition 7.4 merges the list L which represents the execution state of the $a$pproximated $\mathcal{N}$arrowing $\mathcal{C}$alculus when an analysis failed to detect the unsatisfiability of the constraint c and the new constraint $\tilde{c}$ building the list of nodes which are to be narrowed in order to analyze the unsatisfiability of the accumulated constraint $c \cup \tilde{c}$.

Remark 7.5. The facts that every narrowable occurrence of c is, likewise, a narrowable occurrence of $c \cup \tilde{c}$ and that the nodes which are recognized to be *failed* can be dropped from the lists guarantees that the incremental visit of the search tree, as formalized in Definition 7.4, is complete.

The following theorem is a consequence of Corollary 6.4.

Theorem 7.6. (correctness of *iEA*) *Let c be a constraint and L be a list representing the execution state of the calculus $a\mathcal{NC}$ when the (incremental) analysis of unsatisfiability fails to detect the unsatisfiability of c. Let $\tilde{c}$ be a new constraint. If a transition $\langle c, L\rangle \overset{\tilde{c}}{\hookrightarrow}_{iEA} \langle c \cup \tilde{c}, L'\rangle$ cannot be proven then the constraint $c \cup \tilde{c}$ is unsatisfiable.*

PROOF. By contradiction: Assume that the constraint $c \cup \tilde{c}$ is satisfiable. From the definition of $\overset{\tilde{c}}{\hookrightarrow}_{iEA}$ and the remark above, it suffices to show that a transition $[(\langle c \cup \tilde{c}, \epsilon\rangle, \emptyset)] \rightarrow^*_{a\mathcal{NC}} (\langle true, \kappa\rangle, \emptyset) \bullet L''$ can be proven for the hypothesis to be contradicted. By Corollary 6.4 and from the assumption that $c \cup \tilde{c}$ is satisfiable there is no *finitely failed* abstract transition system for $c \cup \tilde{c}$. Therefore, any abstract transition system for this constraint has, at least, a leaf of the form $\langle true, \kappa\rangle$. Since the strategy formalized in Definition 7.1 guarantees that every node of an abstract transition system for $c \cup \tilde{c}$ would be eventually visited, the transition $[(\langle \tilde{c} \cup \tilde{c}, \epsilon\rangle, \emptyset)] \rightarrow^*_{a\mathcal{NC}} (\langle true, \kappa\rangle, \emptyset) \bullet L''$ can be proven. □

8. OPTIMIZATION AND RELATED WORK

We have formalized a number of different analyses to determine if a set of equations is unsatisfiable in a (level-)canonical Horn equational theory. These analyses provide safe approximations of the standard operational semantics of equational logic programs [30, 42]. We have considered basic narrowing for conditional term rewriting systems. It is possible to extend these results to more refined strategies for narrowing, such as selection narrowing [27] or LSE narrowing [12] and it would be worth treating the other variants of narrowing in the same systematic way. We note that the condition of level-canonicity is only required in order to guarantee the completeness of basic conditional narrowing. Due to the possible presence of extra-variables in the programs we cannot weaken level-canonicity to canonicity even if (full) conditional narrowing would be considered (see [42]), if we do not want to lose the completeness of the calculus.

The approach which we have taken in this paper is general. In particular, it applies when we consider any kind of analysis where we look for properties which are satisfied by all (or some) success paths. For specific analyses, it would be necessary to provide the appropriate abstract domains and approximations of the TRSs. In [8], the analysis for a specific goal (the abstract narrowing of a goal) is determined by exploiting the AND-compositionality of basic conditional narrowing and its abstract version. [8] also shows that the compositionality of the abstract semantics leads to a compositional analysis. A bottom-up abstract semantics adequate for modular data-flow analysis is defined in [7]. This abstract semantics captures the computational properties related to the use of logical variables.

The method in [11] to approximate sets of normal forms of ground instances of terms in constructor-based programs can also be formalized as an instance of the framework presented in this paper. Bert et al. [11] define a notion of term description and an abstract TRS which is computed as the least fixpoint of a continuous transformation on programs. Then, terms are abstracted and normalized with the rules of the abstract TRS. If the approximated sets of constructor normal forms do not have any element in common, then the terms are detected not to be $\mathcal{E}$-unifiable.

The analysis of abstract rewriting in [11] and our abstract basic narrowing analysis are incomparable in the sense that there are examples that can be solved using only one of the two methods. For instance, unsatisfiability of the goal $p(x) = x$ is recognized by the method of abstract basic narrowing in the constructor-based program $\mathcal{R} = \{p(0) \rightarrow c(0),\ p(s(x)) \rightarrow x,\ p(c(x)) \rightarrow s(p(x))\}$. However, the set which approximates the constructor normal forms of the ground instances of the term $p(x)$ on the lhs of the goal is a subset of the approximate set of constructor normal forms of the ground instances of the term x on the rhs of the goal, no matter what the level of approximation is. Thus the analysis based on abstract rewriting [11] is not able to detect the unsatisfiability of this equation.

The analysis of abstract rewriting could be improved if reformulated in the framework presented in this paper. In fact, rather than a rewriting relation on terms, the analysis could be redefined as a narrowing relation on equation sets without ignoring the abstract substitutions that are applied to make 'ground' the terms before performing a reduction step.

The analysis of unsatisfiability based on approximated narrowing seems to be particularly suitable for CLP($\mathcal{H}/\mathcal{E}$) [3] since constraints monotonically grow and the analysis can be made incrementally as long as a new set of equations is added. Example 5.19 shows that our method is able to recognize unsatisfiability in cases where the other methods would fail. The contrary is also true in general. Hence, standard methods to detect unsatisfiability should be preferably combined with our analysis. All the above mentioned methods always terminate and reduce the search tree without affecting the completeness of the calculus.

We notice that the concrete narrower can use the result computed by the abstract narrower in the way that we informally suggest. Let $\mathcal{E}$ be an equational theory and c a set of equations. Let $\{\sigma_1, \ldots, \sigma_k\}$ be the collection of all the abstract substitutions (answer set) in the leaves $\langle true, \sigma_j \rangle$, $j = 1, \ldots, k$ of the abstract transition graph for $\mathcal{E}$ with c. Then, for each $\mathcal{E}$-unifier θ of c, there exists i such that $\forall X \in Var(c)$, $X\sigma_i \preceq X\theta$. Thus if we collect the set of answers we can prune all paths which contain a substitution which does not compare with any element in the answer set. The formal definition of this method can be found in [6].

A prototype incremental analyzer based on the ideas described here has been implemented. The performance of the implementation of the algorithm, written in Prolog, has been evaluated on a number of programs. To test our implementation, in addition to a lazy resolution procedure based on the incremental algorithm iEA,

Goal	BCN	$Simpl$	$Simpl + aN$	iEA
g_1	∞	∞	5.08	4.18
g_2	0.78	0.54	12.50	2.48
g_3	0.80	0.58	40.42	2.99

Table 1. Results of the benchmarks of Example 2

we have chosen to implement an eager resolution procedure (relying on the basic conditional narrowing calculus BCN in Section 3) and two (non-incremental) lazy procedures relying on the set of transformations *Simpl* consisting of term rewriting, trivial equations removal, variable substitution, term decomposition and failure rules [30] (here we include an appropriate rule of failure based on the analysis of non-joinability, as formalized in Subsection 5.4). We also extend *Simpl* with an appropriate rule of failure based on abstract narrowing aN. It is difficult to state a general result for the efficiency improvement of the optimizations because this strongly depends on the selected example. Table 1 gives an idea of the achievable speedups of the following benchmarks, which correspond to the Example 5.19 in Section 5.1: $g_1 \equiv \Leftarrow f(g(X)) = c(0), h(f(Y)) = s(0)$, $g_2 \equiv \Leftarrow f(X) = c(g(h(Y))), c(f(Y)) = f(c(W))$ and $g_3 \equiv \Leftarrow f(X) = c(g(h(Y))), h(f(Z)) = W, f(T) = c(V)$.

Timings are CPU seconds on a SUN workstation with 16 Mbytes of RAM. Note that our BCN implementation is slow in comparison with current implementations of narrowing. However, our experiments with this prototype just aimed to check the incrementality and the ability to prove unsatisfiability. The first row shows that as soon as we add the abstract narrowing analysis aN we obtain a calculus which terminates (*Simpl+aN*) versus the standard *BCN* calculus which do not terminate even if the set of transformations *Simpl* is used. Moreover, the comparison between the column of the non incremental analysis (*Simpl + aN*) and that of the incremental calculus (*iEA*), shows that the speed-up due to incrementality is very good. Our conclusion is therefore that the techniques are practical and can be usefully combined for the analysis and optimization of equational logic programs.

Acknowledgment

We thank Giorgio Levi, María José Ramis and Germán Vidal for many helpful discussions.

REFERENCES

1. G. Aguzzi and M.C. Verri. On the termination of a unification procedure. In A. Bertoni, C. Böhm, and P. Miglioli, editors, *Proc. of the Third Italian Conference on Theoretical Computer Science*, pages 59–70, 1989.
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*, volume 631 of *Lecture Notes in Computer Science*, pages 443–457. Springer-Verlag, Berlin, 1992.
6. M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. Optimization of Equational Logic Programs Using Abstract Narrowing. Technical Report DSIC-II/1/93, UPV, 1993. Short version in J. Penjam and M. Bruynooghe, editors, *Proc. of PLILP'93*, volume 714 of *Lecture Notes in Computer Science*, pages 391-409, Springer-Verlag, Berlin, 1993.
7. M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. A Compositional Semantics for Conditional Term Rewriting Systems. In H.E. Bal, editor, *Proc. Sixth IEEE Int'l Conf. on Computer Languages ICCL'94*. IEEE Computer Society Press, 1994.
8. M. Alpuente, M. Falaschi, and G. Vidal. Compositional Analysis for Equational Horn Programs. In G. Levi and M. Rodríguez-Artalejo, editors, *Proc. Fourth Int'l Conf. on Algebraic and Logic Programming ALP'94, Madrid (Spain)*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1994.
9. K. R. Apt. Introduction to Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics. Elsevier, Amsterdam and The MIT Press, Cambridge, Mass., 1990.
10. J.A. Bergstra and J.W. Klop. Conditional Rewrite Rules: confluence and termination. *Journal of Computer and System Sciences*, 32:323–362, 1986.
11. D. Bert, R. Echahed, and B. M. Østvold. Abstract Rewriting. In P. Cousot, M. Falaschi, G. Filè, and A. Rauzy, editors, *Proc. Third Int'l Workshop on Static Analysis WSA'93*, volume 724 of *Lecture Notes in Computer Science*, pages 178–192. Springer-Verlag, Berlin, 1993.
12. A. Bockmayr, S. Krischer, and A. Werner. An Optimal Narrowing Strategy for General Canonical Programs. In M. Rusinowitz and J. L. Rémy, editors, *Proc. Third Int'l Workshop on Conditional Term Rewriting Systems CTRS'92*, volume 656 of *Lecture Notes in Computer Science*, pages 483–497. Springer-Verlag, Berlin, 1992.
13. P.G. Bosco, E. Giovannetti, G. Levi, and C. Palamidessi. A complete semantic characterization of K-leaf, a logic language with partial functions. In *Proceedings of the IEEE Symposium on Logic Programming*, pages 318–327, 1987.
14. H.J. Bürckert. Lazy E-unification - a method to delay alternative solutions. Technical report, Université de Nancy, France, 1987.
15. J. Chabin and P. Réty. Narrowing directed by a graph of terms. In R. V. Book, editor, *Proc. of RTA'91*, volume 488 of *Lecture Notes in Computer Science*, pages 112–123. Springer-Verlag, Berlin, 1991.
16. 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.
17. M. Codish, M. Falaschi, and K. Marriott. Suspension Analyses for Concurrent Logic Programs. *ACM Transactions on Programming Languages and Systems*, May 1994.
18. P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proc. Fourth ACM Symp. Principles of Programming Languages*, pages 238–252, 1977.
19. 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.

20. N. Dershowitz, M. Okada, and G. Sivakumar. Canonical Conditional Rewrite Systems. In *Proc. 9th Int'l Conf. on Automated Deduction CADE'88*, volume 310 of *Lecture Notes in Computer Science*, pages 538–549. Springer-Verlag, Berlin, 1988.
21. N. Dershowitz and A. Plaisted. Logic Programming cum Applicative Programming. In *Proc. First IEEE Int'l Symp. on Logic Programming*, pages 54–66. IEEE, 1984.
22. 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.
23. M. Dincbas and P. van Hentenryck. Extended Unification Algorithms for the Integration of Functional Programming into Logic Programming. *Journal of Logic Programming*, 4:197–227, 1987.
24. M. Fernández. Narrowing based procedures for equational disunification. *Applicable Algebra in Engineering, Communication and Computing*, 3:1–26, 1992.
25. 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.
26. J.H. Gallier and S. Raatz. Extending SLD-resolution to equational Horn clauses using E-unification. *Journal of Logic Programming*, 6:3–43, 1989.
27. 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.
28. P. Van Hentenryck. Incremental Constraint Satisfaction in logic programming. In D. H. D. Warren and P. Szeredi, editors, *Proc. Seventh Int'l Conf. on Logic Programming*, pages 189–202. The MIT Press, Cambridge, Mass., 1990.
29. P. Van Hentenryck and Y. Deville. Operational Semantics of Constraint Logic Programming over Finite Domains. In J. Maluszyński and M. Wirsing, editors, *Proc. of PLILP'91*, volume 528 of *Lecture Notes in Computer Science*, pages 395–406. Springer-Verlag, Berlin, 1991.
30. S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1989.
31. J.M. Hullot. Canonical Forms and Unification. In *5th Int'l Conf. on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Springer-Verlag, Berlin, 1980.
32. H. Hussman. Unification in conditional-equational theories. Technical report, Fakultät für Mathematik und Informatik, Universität Passau, 1986. Short version in *Proc. European Conf. on Computer Algebra EUROCAL'85*, volume 204 of *Lecture Notes in Computer Science*, pages 543-553, Springer-Verlag, Berlin, 1985.
33. J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proc. Fourteenth Annual ACM Symp. on Principles of Programming Languages*, pages 111–119. ACM, 1987.
34. J. Jaffar, 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.
35. A. Josephson and N. Dershowitz. An implementation of narrowing, the RITE way. In *Proc. Third IEEE Int'l Symp. on Logic Programming*, pages 187–197. IEEE, 1986.
36. S. Kaplan. Fair conditional term rewriting systems: unification, termination and confluence. In H.-J. Kreowski, editor, *Recent Trends in Data Type Specification*,

volume 116 of *Informatik-Fachberichte*, pages 136–155. Springer-Verlag, Berlin, 1986.

37. J.W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume I, pages 1–112. Oxford University Press, 1992.
38. 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.
39. 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.
40. M. J. Maher. On parameterized substitutions. Technical Report RC 16042, IBM - T.J. Watson Research Center, Yorktown Heights, NY, 1990.
41. K. Marriott and H. Søndergaard. Semantics-based Dataflow Analysis of Logic Programs. In G. Ritter, editor, *Information Processing 89*. North-Holland, 1989.
42. A. Middeldorp and E. Hamoen. Completeness Results for Basic Narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5:213–253, 1994.
43. J.J. Moreno and M. Rodriguez-Artalejo. BABEL: A Functional and Logic Programming Language based on a constructor discipline and narrowing. In I. Grabowski, P. Lescanne, and W. Wechler, editors, *Algebraic and Logic Programming*, volume 343 of *Lecture Notes in Computer Science*, pages 223–232. Springer-Verlag, Berlin, 1988.
44. W. Nutt, P. Réty, and G. Smolka. Basic narrowing revisited. *Journal of Symbolic Computation*, 7:295–317, 1989.
45. 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.
46. P. Réty. Improving Basic Narrowing Techniques. In *Proc. of RTA'87*, volume 256 of *Lecture Notes in Computer Science*, pages 228–241. Springer-Verlag, Berlin, 1987.
47. 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.
48. J.H. Siekmann. Unification Theory. *Journal of Symbolic Computation*, 7:207–274, 1989.