Compositional Analysis for Equational Horn Programs*

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

[1] Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia,
Camino de Vera s/n, Apdo. 22012, 46020 Valencia, Spain.
e.mail:{alpuente,gvidal}@dsic.upv.es
[2] Dipartimento di Elettronica e Informatica,
Università di Padova,
Via Gradenigo 6/A, 35131 Padova, Italy.
e.mail: falaschi@di.unipi.it

Abstract. We introduce a compositional characterization of the operational semantics of equational Horn programs. Then we show that this semantics and the standard operational semantics based on (basic) narrowing coincide. We define an abstract narrower mimicking this semantics, and show how it can be used as a basis for efficient AND-compositional program analysis. As an application of our framework, we show a compositional analysis to detect the unsatisfiability of an equation set with respect to a given equational theory. We also show that our method allows us to perform computations and analysis incrementally in a Constraint Equational setting and that the test of satisfiability in this setting can be done in parallel.

Keywords: Semantic analysis, compositionality, equational logic programming, term rewriting systems.

1 Introduction

Compositionality is a desirable property which has been recognised as fundamental in the semantics of programming languages [11]. Compositionality has to do with a (syntactic) composition operator $\diamond$, and holds when the meaning (semantics) $\mathcal{S}(C_1 \diamond C_2)$ of a compound construct is defined by composing the meanings of the constituents $\mathcal{S}(C_1)$ and $\mathcal{S}(C_2)$, i.e. for a suitable function $f_\diamond$, $\mathcal{S}(C_1 \diamond C_2) = f_\diamond(\mathcal{S}(C_1), \mathcal{S}(C_2))$. In the case of logic programs [11], one could be concerned with AND-composition (of atoms in a goal or in a clause body) or with OR-composition, i.e. composition of (sets of) clauses. In the context of equational logic programming [15,19], consideration has been given to the problem of defining a compositional semantics for the direct union of complete theories which correctly models the computational properties related to the use of logical variables [4]. In this paper, we address the problem of solving equations

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

in equationally defined theories but, unlike [4], we want to define compositionally the meaning of the union of $\mathcal{E}$-unification problems. Let a theory $\mathcal{E}$ be fixed and consider two finite equation sets Γ_1, Γ_2. We want to define the meaning of $\Gamma_1 \cup \Gamma_2$ (with respect to $\mathcal{E}$) in terms of the meanings of Γ_1 and Γ_2. Throughout the paper, $\mathcal{E}$ is assumed to be axiomatized as an equational Horn theory, which is called the 'program' [12,15,19]. We do not consider compositionality with respect to the union of programs in this paper. For this topic we refer to [4].

The semantics of equational Horn programs is usually defined as some variant of *narrowing*, a method for generating complete sets of $\mathcal{E}$-unifiers with respect to a canonical set of clauses. Simple restrictions on narrowing, like narrowing only at basic positions, are still complete for theories which satisfy some additional properties [25]. The use of narrowing as the operational mechanism for computing leads to a powerful extension of ordinary logic programs [15,28] and the computation model has many opportunities for parallelism. For example, [10] describes a kind of OR-parallelism in which, for each position in the term and each rule in the program, alternative narrowings are explored concurrently according to some heuristic function. Our work concerns an AND-parallel computation model of equational Horn programs, where all subexpressions can be narrowed independently and the computed substitutions obtained so far can then be composed. This mechanism of computation was also mentioned in [28]. We extend the notion of *parallel composition* of substitutions introduced in [17,27] to the case of equational logic programs. Hence we show how complete sets of $\mathcal{E}$-unifiers for a given goal $\Leftarrow \Gamma_1, \Gamma_2$ can be generated by composing the complete sets of $\mathcal{E}$-unifiers computed by narrowing the separate subgoals $\Leftarrow \Gamma_1$ and $\Leftarrow \Gamma_2$. This allows us to model the combination of substitutions computed by AND-parallel 'narrowing processes', i.e. by agents which narrow subexpressions in parallel. We show that for unrestricted narrowing a "semantic" composition operator would be necessary. However, for basic narrowing we achieve a stronger result and show that the substitutions can be composed syntactically.

We have recently defined an equational logic language [1] as an instance of Constraint Logic Programming [18], where the equations to be solved with respect to an equational theory are considered as constraints. For a computational step in this framework, it is essential to be able to (semi-)decide if a set of equations is satisfiable. The computation of complete sets of $\mathcal{E}$-unifiers is less striking in this context, while it is essential a mechanism to evaluate the satisfiability of the constraints incrementally. In [2] we have defined a lazy procedure which does not prove the satisfiability of the equational constraint c but just checks that c is not unsatisfiable by means of an approximated narrower. We show that a compositional narrowing can be taken as a basis for a compositional analysis for the problem of unsatisfiability. Then we show that compositionality leads, as a by-product, to an incremental implementation for the analysis. Therefore, while OR-compositionality, i.e. compositionality w.r.t. union of programs, has proven significant for programming with modules [4,11], we show that AND-compositionality can lead to incremental computations.

This paper is organized as follows. After introducing some preliminary notions in Section 2, Section 3 defines an operator which describes the operational semantics of equational Horn programs in a compositional way. In Section 4, we introduce *compositional conditional narrowing*, an AND-parallel computation model for equational Horn programs. We characterize the *success* set of a goal, i.e. the set of the computed answer substitutions corresponding to all successful narrowing derivations. That is, we prove that the meaning of a composite goal can be obtained by composing the meanings of its conjuncts, when considering the success set as observable. Then we state the completeness of our semantics. In Section 5 we recall an abstract algorithm for the static analysis of unsatisfiability of equation sets [2] and modify it to perform the analysis compositionally. Section 6 formulates an incremental analyzer for equational constraints and presents some encouraging results from the implementation of the analyzer. Section 7 concludes. More details and missing proofs can be found in [5].

2 Preliminaries

We briefly recall some known results about equations, conditional rewrite systems and equational unification. For full definitions refer to [9,20]. Throughout this paper, V will denote a countably infinite set of variables and Σ denotes a set of function symbols, each with a fixed associated arity. $\tau(\Sigma \cup V)$ and $\tau(\Sigma)$ denote the sets of terms and ground terms built on $\Sigma \cup V$ and Σ, respectively. A Σ-equation $s = t$ is a pair of terms $s, t \in \tau(\Sigma \cup V)$. Terms are viewed as labelled trees in the usual way. 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 $uw = v$. $\bar{O}(t)$ 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. These notions extend to equations in a natural way. Identity of syntactic objects is denoted by $\equiv$. $Var(s)$ is the set of distinct variables occurring in the syntactic object s. A *fresh* variable is a variable that appears nowhere else. The symbol $\tilde{}$ denotes a finite sequence of symbols.

We describe the lattice of equation sets following [7]. We let Eqn denote the set of possibly existentially quantified finite sets of equations over terms. We let *fail* denote the unsatisfiable equation set, which (logically) implies all other equation sets. Likewise, the empty equation set, denoted 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. Any set of equations E can be transformed into an equivalent one $solve(E)$ which is solved. We restrict our interest to the set of idempotent substitutions over $\tau(\Sigma \cup V)$, which is denoted by Sub. There is a natural isomorphism between substitutions and unquantified equation sets. The equational representation of a substitution $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ is the set of equations $\widehat{\theta} = \{x_1 = t_1, \ldots, x_n = t_n\}$. The identity function on V is called the empty substitution and denoted ϵ. Given a substitution θ and a set of variables $W \subseteq V$, we denote by $\theta_{\lceil W}$ the substitution obtained from θ by restricting its domain, $Dom(\theta)$, to W.

We consider the usual preorder on substitutions $\leq$: $\theta \leq \sigma$ iff $\exists \gamma . \sigma \equiv \theta\gamma$. Note that $\theta \leq \sigma$ iff $\widehat{\sigma} \Rightarrow \widehat{\theta}$ [27]. A substitution $\{x_1/t_1, \ldots, x_n/t_n\}$ is a *unifier* of an equation set E iff $\{x_1 = t_1, \ldots, x_n = t_n\} \Rightarrow E$. 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. In abuse of notation, we let *fail* denote failure when computing the $mgu(E)$. While every unquantified equation set has a *most general unifier* [23], this is not generally true for equation sets with existentially quantified variables [7].

An equational Horn 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 equations. An equational goal is an equational Horn clause with no head. We let *Goal* denote the set of equational goals. The set of *states* is defined by $State = Goal \times Sub$.

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

An equational Horn theory $\mathcal{E}$ which satisfies the above assumptions can be viewed as a term rewriting system $\mathcal{R}$ where the rules are the heads (implicitely oriented from left to right) and the conditions are the respective bodies. We assume that these assumptions hold for all theories we consider in this paper. The equational theory $\mathcal{E}$ is said to be canonical (complete) if the binary one-step conditional rewriting relation $\rightarrow_{\mathcal{R}}$ defined by $\mathcal{R}$ is noetherian and confluent. 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). Given a conditional TRS $\mathcal{R}$, an equational goal clause $\Leftarrow g$ conditionally narrows into a goal clause $\Leftarrow g'$ (in symbols $\Leftarrow g \overset{\theta}{\hookrightarrow} \Leftarrow g'$) if there exists an equation $e \in g$, $u \in \bar{O}(e)$, a standardised apart variant $(\lambda \rightarrow \rho \Leftarrow \widetilde{e}) \ll \mathcal{R}$ and a substitution θ such that $\theta = mgu(\{e_{|u} = \lambda\})$ and $g' = ((g \sim \{e\}) \cup \{e[\rho]_u\} \cup \widetilde{e})\theta$. A *narrowing derivation* is defined by $\Leftarrow g \overset{\theta}{\hookrightarrow}{}^{*} \Leftarrow g'$ iff $\exists \theta_1, \ldots, \theta_n.\ \Leftarrow g \overset{\theta_1}{\hookrightarrow} \ldots \overset{\theta_n}{\hookrightarrow} \Leftarrow g'$ and $\theta = \theta_1 \ldots \theta_n$. We say that the derivation has length n. In order to treat syntactical unification as a narrowing step, we add to the TRS $\mathcal{R}$ the rule $x = x \rightarrow true \Leftarrow$, $x \in V$. Then $t = s \overset{\sigma}{\hookrightarrow} true$ holds iff $\sigma = mgu(\{t = s\})$. A successful derivation for $\Leftarrow g$ in $\mathcal{R} \cup \{x = x \rightarrow true \Leftarrow\}$ is a narrowing derivation $\Leftarrow g \overset{\theta}{\hookrightarrow}{}^{*} \Leftarrow true$ and $\theta_{\lceil Var(g)}$ is called a computed answer substitution for $\Leftarrow g$ in $\mathcal{R}$. If $n = 0$ then $\theta = \epsilon$.

Each equational Horn theory $\mathcal{E}$ generates a smallest congruence relation $=_{\mathcal{E}}$ called *$\mathcal{E}$-equality* on the set of terms $\tau(\Sigma \cup V)$ (the least equational theory which contains all logic 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 $\mathcal{E}$. $\mathcal{E}$-equality is extended to substitutions by $\sigma =_{\mathcal{E}} \theta$ iff $\forall x \in V.\ x\sigma =_{\mathcal{E}} x\theta$.

A finite set of equations $\Gamma = \{s_1 = t_1, \ldots, s_n = t_n\}$ together with an equational theory $\mathcal{E}$ is called an *$\mathcal{E}$-unification* problem. A substitution σ is an *$\mathcal{E}$-unifier* or a *solution* of the equation set Γ iff $\mathcal{E} \models (\widehat{\sigma} \Rightarrow \Gamma)$ [24]. The set $\mathcal{U}_{\mathcal{E}}(\Gamma)$ of all *$\mathcal{E}$-unifiers* of Γ is recursively enumerable [12,15,29].

For *$\mathcal{E}$-unification* problems, 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 Γ is complete iff every $\mathcal{E}$-unifier σ of Γ 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 $\mu\mathcal{U}_{\mathcal{E}}(\Gamma)$ of *$\mathcal{E}$-unifiers* of Γ do not always exist. An *$\mathcal{E}$-unification* procedure is complete if it generates a complete set of *$\mathcal{E}$-unifiers* for all input equation system. Conditional narrowing has been shown to be a complete $\mathcal{E}$-unification algorithm for canonical theories satisfying different restrictions [15,25].

3 Equational Parallel Composition

In the following, we recall the notion of *parallel composition* of substitutions, denoted by $\Uparrow$. Roughly speaking, parallel composition is the operation of unification generalized to substitutions.

Parallel composition corresponds to one of the basic operations performed by the AND-parallel execution model of logic programs [17,27]. Namely, when two subgoals (of the same goal) are run in parallel, the answer substitutions (computed independently) have to be combined to get the final result. This 'combination' can be done as follows [17,27]. Given two idempotent substitutions θ_1 and θ_2, we let $\theta_1 \Uparrow \theta_2 = mgu(\widehat{\theta_1} \cup \widehat{\theta_2})$. Parallel composition is idempotent, commutative, associative and has a null element *fail* and an identical element ϵ. $\Uparrow$ is lifted to sets of substitutions by

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

Parallel composition was proposed in [27] as a basis for a compositional characterization of the semantics of Horn Clause Logic. We are able to generalize the notion of parallel composition to the case when unification in equational theories is considered. In the following definition we formalize the notion of *equational parallel composition*, denoted by $\Uparrow_{\mathcal{E}}$.

Definition 1. Let $\theta_1, \theta_2 \in Sub$. We define the operator $\Uparrow_{\mathcal{E}} : Sub \times Sub \rightarrow \wp(Sub)$ as follows:

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

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

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

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

Definition 2. Given $\Theta_1, \Theta_2 \in \wp(Sub)$, let:

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

Given an $\mathcal{E}$-unification problem $\Gamma = \Gamma_1 \cup \Gamma_2$, a compositional characterization of the set of all $\mathcal{E}$-unifiers of Γ is given in the following proposition.

Proposition 3. *Let* $\Gamma = \Gamma_1 \cup \Gamma_2$, $\Theta_1 = \mathcal{U}_{\mathcal{E}}(\Gamma_1)$ *and* $\Theta_2 = \mathcal{U}_{\mathcal{E}}(\Gamma_2)$. *Then* $\mathcal{U}_{\mathcal{E}}(\Gamma) = \Theta_1 \Uparrow_{\mathcal{E}} \Theta_2$.

We note that it is much more complex to evaluate $\Uparrow_{\mathcal{E}}$ than the much simpler operation $\Uparrow$. However, equational parallel composition can be redefined in terms of 'most general' unifiers in the case of *finitary* theories, for which the solutions to an $\mathcal{E}$-unification problem Γ can always be represented by a complete and minimal finite set $\mu\mathcal{U}_{\mathcal{E}}(\Gamma)$ of (maximally general) $\mathcal{E}$-unifiers, which is unique up to equivalence [29]. Equational theories which are of finitary *unification type* play an important role in logic programming with equality [19]. In general, the unification type of an equational theory is undecidable. On the other hand, for a finitary theory the minimality requirement is often too strong, since an algorithm which generates a superset of $\mu\mathcal{U}_{\mathcal{E}}(\Gamma)$ may be far more efficient than a minimal one and hence sometimes preferable. In the following section, we will show that we can still work with ordinary parallel composition $\Uparrow$ when we consider the class of (level-)canonical equational theories, for which the problem of $\mathcal{E}$-unification reduces to ordinary (syntactic) unification plus narrowing [16].

4 Compositional Conditional Narrowing

In this section we recall the operational semantics of our language, following [4]. We specify the *observables*, that is, the property we are interested to "observe" in a computation (e.g. the set of successes, the finite failure set, the infinite failure set, etc.) and that has to be reflected in the semantics. In this paper we are interested in the success set. We describe the *success* set of a goal, i.e. the set of all computed answer substitutions corresponding to all successful narrowing derivations, by a formal operational semantics $\mathcal{O} : Goal \mapsto \wp\, Sub$, based on a (labelled) transition relation, which associates a set of substitutions with a goal.

Basic conditional narrowing has been proposed as the operational model of equational logic programs [15,25]. Basic conditional narrowing gives a complete set of solutions for level-canonical sets of conditional rewrite rules [25]. Compared to ordinary narrowing, the basic strategy leads to a smaller search space by eliminating some search paths that lead to reducible solutions. In the rest of this paper, we assume the level-canonical program $\mathcal{R}$ to be fixed. We formulate basic conditional narrowing as a transition system $(State, \leadsto)$ whose transition relation $\leadsto \subseteq State \times State$ is defined as the smallest relation satisfying

$$\langle \Leftarrow g, \theta\rangle \leadsto \langle \Leftarrow g', \theta\sigma\rangle \text{ iff } e \in g \ \wedge \ u \in \bar{O}(e) \ \wedge (\lambda \rightarrow \rho \Leftarrow \widetilde{e}) \ll \mathcal{R} \ \wedge$$
$$\sigma = mgu(\{(e_{|u})\theta = \lambda\}) \ \wedge \ g' = (g \sim \{e\}) \cup \{e[\rho]_u\} \cup \widetilde{e}.$$

Note that, in the above inference rule, the computed substitution σ is not applied to the equations in the derived state, as opposed to ordinary (unrestricted) narrowing (cf. Section 2). This ensures that no narrowing step will reduce any expression brought by a substitution computed in a previous step.

A *basic conditional narrowing derivation* is a sequence of states $s_1 \leadsto s_2 \leadsto \ldots$, where $s_i \equiv \langle \Leftarrow g_i, \theta_i\rangle$ is the ith state in the sequence.

Based on this transition system, we define the operational semantics of an equational goal $\Leftarrow g$ in the TRS $\mathcal{R} \cup \{x = x \rightarrow true \Leftarrow\}$ by the (non ground) set (*success set*)

$$\mathcal{O}_{\mathcal{R}}(\Leftarrow g) = \{\theta \mid \langle \Leftarrow g, \epsilon\rangle \leadsto^* \langle \Leftarrow true, \theta\rangle\}^3.$$

Now we are ready to give a compositional characterization of the operational semantics of equational Horn programs in a style similar to that of [27] for logic programs. We define a new narrowing procedure by means of a transition relation from equational goals to equational goals, labeled on substitutions.

Definition 4. (Compositional Conditional Narrowing)
We define *compositional conditional narrowing* as a labelled transition system $(Goal, Sub, \longmapsto)$ whose transition relation $\longmapsto \subseteq Goal \times Sub \times Goal$ is the smallest relation which satisfies

$$(1)\ \frac{u \in \bar{O}(e) \ \wedge \ (\lambda \rightarrow \rho \Leftarrow \widetilde{e}) \ll \mathcal{R} \ \wedge \ \sigma = mgu(\{e_{|u} = \lambda\})}{\Leftarrow \{e\} \stackrel{\sigma}{\longmapsto} \Leftarrow \{e[\rho]_u\} \cup \widetilde{e}}$$

$$(2)\ \frac{\Leftarrow g_1 \stackrel{\theta_1}{\longmapsto} \Leftarrow g_1' \ \wedge \ \Leftarrow g_2 \stackrel{\theta_2}{\longmapsto} \Leftarrow g_2'}{\Leftarrow g_1, g_2 \stackrel{\theta_1 \Uparrow \theta_2}{\longmapsto} \Leftarrow g_1', g_2'}$$

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

[3] We often write $\mathcal{O}(\Leftarrow g)$ instead of $\mathcal{O}_{\mathcal{R}}(\Leftarrow g)$ when $\mathcal{R}$ is understood.

Note that, by not applying the substitutions to the goal at each derivation step, compositional conditional narrowing may have to overcompute when compared to (basic) conditional narrowing thus possibly causing execution to slow down (see Example 2 below).

The computation model formalized in Definition 4 could be taken as a basis for an AND-parallel computation model of equational Horn programs. We note that the model has not been devised to achieve maximal parallelism in the sense that not all redexes in a given goal are allowed to perform one narrowing step independently. Namely, redexes which occur in a same equation are not reduced in parallel, while they could. To overcome this lack, it suffices to introduce the following *flattening* rule, which preserves the reachable solutions

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

provided that both $e_{|u}$ and $e[x]_u$ contain at least one function symbol [26]. Note that we need not determine the level of granularity [22] (as it neither affects correctness nor completeness); this we consider to be an implementation issue that could enable (more) effective parallelizations.

Our approach differs from other AND-parallel execution models, such as e.g. [22], where subexpressions are only narrowed in parallel if they are *independent*, i.e. if they do not share (unbound) variables. A 'need-driven' syncronization model is imposed which compels processes to wait for the value of a parallel subexpression if such a value is needed. In [21], a *dependent* AND-parallel execution model is exploited, but the imposed syncronization mechanisms produce too much overhead.

A new operational semantics of an equational goal $\Leftarrow g$ in the TRS $\mathcal{R} \cup \{x = x \rightarrow true \Leftarrow\}$ can now be defined by

Definition 5. $\mathcal{O}'(\Leftarrow g) = \{\theta \mid \Leftarrow g \overset{\theta_1}{\longmapsto} \ldots \overset{\theta_n}{\longmapsto} \Leftarrow true \text{ and } \theta = \theta_1 \Uparrow \ldots \Uparrow \theta_n, \theta \not\equiv fail\}$.

In Definition 5 we use parallel composition because, in the transition system in Definition 4, the computed substitution is not applied to either the derived goal or to the redex $e_{|u}$ selected to be narrowed, as opposed to the standard semantics. Therefore, the next computation step will not take this substitution into account and the next substitution that is computed has to be combined with the previous one.

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

Theorem 6. $\mathcal{O}'(\Leftarrow g_1, g_2) = \mathcal{O}'(\Leftarrow g_1) \Uparrow \mathcal{O}'(\Leftarrow g_2)$.

The following result states that the compositional conditional semantics and the standard basic conditional semantics coincide. The correspondence is restricted to successes, namely to the substitutions computed by all successfully terminating derivations. The compositional semantics and the standard basic semantics have a different failure behaviour: the latter delivers finite failure for more goals than the former, as illustrated in the following example.

Example 2. Let $\mathcal{R} = \{f(0) \to 0 \Leftarrow,\ f(c(X)) \to c(f(X)) \Leftarrow\}$ and consider the goal $\Leftarrow g \equiv \Leftarrow f(c(0)) = 0$. Then there is only one (failed) basic narrowing derivation for $\Leftarrow g$ in $\mathcal{R}$:

$$\langle \Leftarrow f(c(0)) = 0, \epsilon\rangle \leadsto \langle \Leftarrow c(f(X)) = 0, \{X/0\}\rangle \leadsto \langle \Leftarrow c(0) = 0, \{X/0\}\rangle,$$

whereas there exists the nonterminating compositional narrowing derivation:

$$\Leftarrow f(c(0)) = 0 \overset{\{X/0\}}{\longmapsto} \Leftarrow c(f(X)) = 0 \overset{\{X/c(Y)\}}{\longmapsto} \Leftarrow c(c(f(Y))) = 0 \longmapsto \ldots$$

Corollary 7. $\langle \Leftarrow g, \epsilon\rangle \leadsto^* \langle \Leftarrow true, \theta\rangle$ *iff* $\Leftarrow g \overset{\theta_1}{\longmapsto} \ldots \overset{\theta_n}{\longmapsto} \Leftarrow true$ *and* $\theta = \theta_1 \Uparrow \ldots \Uparrow \theta_n$, $\theta \not\equiv fail$.

We note that the equivalence established by Theorem 6 and Corollary 7 does not hold for ordinary (unrestricted) narrowing. Roughly speaking, what is wrong with ordinary narrowing is the fact that it transfers terms from the substitution part into the goal, thus introducing narrowing steps (at *non-basic* positions) that might not be proven when the subgoals are solved independently. As a consequence, ordinary narrowing is not compositional using the parallel composition operator $\Uparrow$. The following example illustrates this point.

Example 3. Let $\mathcal{R}$ be the following program

$$\begin{array}{rl} \mathcal{R} = \{ & z(s(0)) \to 0 \Leftarrow \\ & z(one(X)) \to 0 \Leftarrow \\ & one(0) \to s(0) \Leftarrow \\ & one(s(X)) \to one(X) \Leftarrow\ \}, \end{array}$$

and consider the following (ordinary) narrowing derivation:

$$\begin{array}{l} \Leftarrow X = s(0),\ z(X) = 0 \overset{\{X/one(Y)\}}{\hookrightarrow} \Leftarrow one(Y) = s(0),\ 0 = 0 \overset{\{Y/0\}}{\hookrightarrow} \\ \Leftarrow s(0) = s(0),\ 0 = 0 \overset{\epsilon}{\hookrightarrow} \Leftarrow true,\ 0 = 0 \overset{\epsilon}{\hookrightarrow} \Leftarrow true \end{array}$$

with computed answer substitution $\theta = \{X/one(0)\}$. According to Definition 4, there is no compositional narrowing derivation for $\mathcal{R}$ with initial goal $\Leftarrow X = s(0),\ z(X) = 0$ with computed answer substitution $\{X/one(0)\}$ as the goal $\Leftarrow X = s(0)$ only computes the substitution $\{X/s(0)\}$. Note that basic conditional narrowing does not compute the answer substitution $\{X/one(0)\}$ either.

As a consequence of Corollary 7, every solution found by basic conditional narrowing is found by compositional basic conditional narrowing as well. Hence, we have the following corollary for level-canonical systems.

Corollary 8. (completeness)
The set $\{\vartheta_{\restriction Var(g)} \mid \vartheta \in \mathcal{O}'(\Leftarrow g)\}$ *is a complete set of* $\mathcal{E}$*-unifiers of* g.

We note that, by forcing the join of the parallel solutions every time that all equations in the goal have performed a single step independently, the compositional execution model formalized by Definition 4 might not couch all the exploitable AND-parallelism. Corollary 7 suggests that many different computation schemes are possible. For instance, we can solve in parallel all (sub-)goals, joining the AND-parallel (sub-)goals when the (sub-)goals are completely solved, instead of forcing the syncronization of the AND-parallel branches at every single reduction step. In the following section, we show how this execution scheme can be efficiently exploited for program analysis.

5 Abstract Basic Conditional Narrowing

Abstract interpretation is a theory of semantic approximation which is used to provide statically sound answers to some questions about the run-time behaviour of programs [8]. The 'concrete' data and semantic operators are approximated and replaced by corresponding abstract data and operators. The 'answers' obtained by using the abstract data and operators have to be proven sound by exploiting the correspondence with the concrete data and operators. In this section, we recall the framework of abstract interpretation for analysis of equational unsatisfiability we defined in [2]. Then we extend this framework by defining a compositional abstract semantics which safely approximates the observables.

Our analysis of unsatisfiability is an abstraction of the transition system semantics for basic conditional narrowing that we have introduced in Definition 4. We first recall the abstract domains and the associated abstract operators together with some previous results concerning them. We note that a different analysis of unsatisfiability is introduced in [6] for constructor-based programs where the computation is done by *abstract rewriting*. This method is not comparable to ours, since there are examples which can be analyzed by only one of the two methods and our method is able to capture some computational properties related to the use of logical variables that abstract rewriting does not.

5.1 Abstract Domains and Operators

A *description* is the association of an *abstract domain* $(D, \leq)$ (a poset) with a *concrete domain* $(E, \leq)$ (a poset). When $E = Eqn$, $E = Sub$ or $E = State$, the description is called an *equation description*, a *substitution description* or a *state description*, respectively. The correspondence between the abstract and concrete domain is established through a 'concretization' function $\gamma : D \rightarrow \wp E$. We say that *d approximates e*, written $d \propto e$, iff $e \in \gamma(d)$. The approximation relation can be lifted to relations and cross products as usual [2].

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 [2]. State descriptions consist of a set of equations with substitution descriptions. The descriptions for equations, substitutions and term rewriting systems are defined as follows.

Definition 9. 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 $\perp \notin \Sigma$. Let $\mathcal{T}_{\mathcal{A}} = (\tau(\Sigma \cup V \cup \{\perp\}), \preceq)$ be the domain of terms over the signature augmented by $\perp$, where the partial order $\preceq$ is defined as follows:

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

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

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 $S' \subseteq S)$.

Roughly speaking, we introduce the special symbol $\perp$ in the abstract domains to represent any concrete term. Logically, $\perp$ stands for an existentially quantified variable [2,24]. Define $[\![S]\!] = S'$, where the n-tuple of occurrences of $\perp$ in S is replaced by an n-tuple of existentially quantified fresh variables in S'.

Definition 10. An abstract substitution is a set of the form $\{x_1/t_1, \ldots, x_n/t_n\}$ 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 $t_i \in \tau(\Sigma \cup V \cup \{\perp\})$. The ordering on abstract substitutions can be given as logical implication: let $\theta, \kappa \in Sub_{\mathcal{A}}$, $\kappa \preceq \theta$ iff $[\![\widehat{\theta}]\!] \Rightarrow [\![\widehat{\kappa}]\!]$.

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

Definition 11. Let $\mathcal{T} = (\tau(\Sigma \cup V), \leq)$ and $\mathcal{T}_{\mathcal{A}} = (\tau(\Sigma \cup V \cup \{\perp\}), \preceq)$. The *term description* is $\langle \mathcal{T}_{\mathcal{A}}, \gamma, \mathcal{T} \rangle$ where $\gamma : \mathcal{T}_{\mathcal{A}} \to \wp\, \mathcal{T}$ is defined by: $\gamma(t') = \{t \in \mathcal{T} \mid t' \preceq t\}$.

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

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

Define the abstract state domain $State_{\mathcal{A}}$ induced by $Eqn_{\mathcal{A}}$ and $Sub_{\mathcal{A}}$ to be $State_{\mathcal{A}} = \{\langle \Leftarrow g, \kappa \rangle \mid g \in Eqn_{\mathcal{A}}, \kappa \in Sub_{\mathcal{A}}\}$.

In the following, we formalize the idea that abstract narrowing reduction approximates narrowing reduction by replacing concrete states, unification and term rewriting systems with abstract states, abstract unification and abstract term rewriting systems. We define the abstract most general unifier for an equation set $E' \in Eqn_{\mathcal{A}}$ as follows. First replace all occurrences of $\perp$ in E' by existentially quantified fresh variables. Then take a solved form of the resulting quantified equation set and finally replace the existentially quantified variables again by $\perp$. Formally: let $\exists y_1 \ldots y_n.E = solve(\llbracket E' \rrbracket)$ and $\kappa = \{y_1/\perp, \ldots, y_n/\perp\}$. Then $mgu_{\mathcal{A}}(E') = E\kappa$.

We now extend the notion of parallel composition from substitutions to abstract substitutions by replacing unification by abstract unification.

Definition 12. Let $\kappa_1, \kappa_2 \in Sub_{\mathcal{A}}$. We define the abstract parallel composition $\kappa_1 \Uparrow_{\mathcal{A}} \kappa_2$ by:

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

Our notion of abstract term rewriting system is parametric with respect to a loop-check, i.e. a finite graph of functional dependencies built from the equational theory, which helps to recognize the narrowing derivations which definitely terminate. The purpose of a loop-check is to reduce the search space to end up with a finite search space. Two different instances can be found in [2,3].

Definition 13. A loop-check is a graph $\mathcal{G}_{\mathcal{R}}$ associated with a term rewriting system $\mathcal{R}$, i.e. a relation consisting of a set of pairs of terms, such that: (1) the transitive closure $\mathcal{G}_{\mathcal{R}}^{+}$ is decidable and (2) Let $\mathring{t} = t'$ be a function which assigns to a term t some node t' in $\mathcal{G}_{\mathcal{R}}$. If there is an infinite sequence:
$\langle \Leftarrow g_0, \theta_0 \rangle \rightsquigarrow \langle \Leftarrow g_1, \theta_1 \rangle \rightsquigarrow \ldots$
then

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

(we refer to $\langle \mathring{t}_i, \mathring{t}_i \rangle$ as a 'cycle' of $\mathcal{G}_{\mathcal{R}}$.)

A loop-check can be thought of as a sort of 'oracle' whose usefulness in proving the termination of basic narrowing derivations is stated in the following proposition.

Proposition 14. [2] *Let $\mathcal{R}$ be a term rewriting system and $\mathcal{G}_{\mathcal{R}}$ be a loop-check for $\mathcal{R}$. If there is no cycle in $\mathcal{G}_{\mathcal{R}}$, then every basic conditional narrowing derivation for $\mathcal{R}$ terminates.*

To illustrate our definition, we consider a simple example here.

Example 4. Let $\mathcal{R} = \{X + 0 \rightarrow X \Leftarrow,\ X + s(Y) \rightarrow s(X + Y) \Leftarrow\}$ and define $\mathring{t} = t'$ the function which, given a graph, assigns to a term t some node t' in the graph such that t' unifies with t (variables are implicitly renamed to be disjoint). Then the graph $\mathcal{G} = \{\langle X + s(Y), X + s(Y)\rangle\}$ is a loop-check for $\mathcal{R}$.

Most papers on loop-checking consider the application of loop-checks at run-time. Static loop-checks have not received that much attention yet. In the following we show how a loop-check can be used to obtain a form of (compiled) abstract program which always terminates and in which the semantics of a given goal can be approximated safely. A TRS is abstracted by simplifying the right-hand side and the body of each rule. This definition is given inductively on the structure of terms and equations. The main idea is that terms which are mapped to a cycle of the loop-check are drastically simplified by replacing them by $\perp$. This enforces termination.

Definition 15. (abstract term rewriting system)
Let $\mathcal{R}$ be a TRS. Let $\mathcal{G}_{\mathcal{R}}$ be a loop-check for $\mathcal{R}$. We define the abstraction of $\mathcal{R}$ using $\mathcal{G}_{\mathcal{R}}$ as follows:

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

$$sh(x) = \begin{cases} x & \text{if } x \in V \\ f(sh(t_1), \ldots, sh(t_k)) & \text{if } x = f(t_1, \ldots, t_k) \text{ and } \langle \mathring{x}, \mathring{x} \rangle \notin \mathcal{G}_{\mathcal{R}}^{+} \\ sh(l) = sh(r) & \text{if } x = (l = r) \\ sh(e_1), \ldots, sh(e_n) & \text{if } x = e_1, \ldots, e_n \\ \perp & \text{otherwise} \end{cases}$$

Example 5. (Continued from Example 4) The abstraction of $\mathcal{R}$ using the loop-check $\mathcal{G}$ is: $\mathcal{R}_{\mathcal{A}} = \{X + 0 \to X \Leftarrow,\ X + s(Y) \to s(\perp) \Leftarrow\}$.

5.2 Compositional Abstract Narrowing

We now introduce compositional abstract (basic) narrowing.

Definition 16. Let $\mathcal{R}_{\mathcal{A}}$ be an abstract *TRS*. We define *compositional abstract (basic) narrowing* as a transition system $(State_{\mathcal{A}}, \leadsto_{\mathcal{A}})$ whose transition relation $\leadsto_{\mathcal{A}} \subseteq State_{\mathcal{A}} \times State_{\mathcal{A}}$ is defined as the smallest relation satisfying:

$$(1)\ \frac{\begin{array}{l} e \in g \ \wedge\ u \in \bar{O}(e) \ \wedge\ (\lambda \to \rho \Leftarrow \widetilde{e}) \ll \mathcal{R}_{\mathcal{A}} \ \wedge \\ \sigma = mgu_{\mathcal{A}}(\{(e_{|u})\kappa = \lambda\}) \end{array}}{\langle \Leftarrow g, \kappa \rangle \ \leadsto_{\mathcal{A}}\ \langle \Leftarrow (g \sim \{e\}) \cup \{e[\rho]_u\} \cup \widetilde{e}, \kappa\sigma \rangle}$$

$$(2)\ \frac{\langle \Leftarrow g_1, \kappa \rangle \leadsto_{\mathcal{A}}^{*} \langle \Leftarrow true, \kappa_1 \rangle \ \wedge\ \langle \Leftarrow g_2, \kappa \rangle \leadsto_{\mathcal{A}}^{*} \langle \Leftarrow true, \kappa_2 \rangle}{\langle \Leftarrow (g_1, g_2), \kappa \rangle \leadsto_{\mathcal{A}} \langle \Leftarrow true, \kappa_1 \Uparrow_{\mathcal{A}} \kappa_2 \rangle}$$

Example 6. (Continued from Example 5) The sequences

$\langle \Leftarrow X + 0 = 0, \epsilon \rangle \leadsto_{\mathcal{A}} \langle \Leftarrow X = 0, \epsilon \rangle \leadsto_{\mathcal{A}} \langle \Leftarrow true, \{X/0\} \rangle$,
$\langle \Leftarrow X + s(Y) = s(0), \epsilon \rangle \leadsto_{\mathcal{A}} \langle \Leftarrow s(\perp) = s(0), \epsilon \rangle \leadsto_{\mathcal{A}} \langle \Leftarrow true, \epsilon \rangle$, and
$\langle \Leftarrow (X + 0 = 0,\ X + s(Y) = s(0)),\ \epsilon \rangle \leadsto_{\mathcal{A}} \langle \Leftarrow true,\ \{X/0\} \rangle$

are three (successful) compositional abstract narrowing derivations for $\mathcal{R}_{\mathcal{A}} \cup \{x = x \to true \Leftarrow\}$. Note that there is no successful compositional abstract narrowing derivation for $\mathcal{R}_{\mathcal{A}}$ with initial goal $\langle \Leftarrow s(0) + Y = 0, \epsilon \rangle$.

The following definition formalizes the *compositional abstract basic narrowing semantics* for the success set.

Definition 17. (abstract semantics)

$$\Delta_{\mathcal{R}_{\mathcal{A}}}(\Leftarrow g) = \{\kappa \in Sub_{\mathcal{A}} \mid \langle \Leftarrow g, \epsilon \rangle \leadsto^*_{\mathcal{A}} \langle \Leftarrow true, \kappa \rangle\}.$$

The main purpose of introducing compositional abstract basic narrowing here is to suggest a mechanism for the static analysis of the run-time behaviour of programs. We now establish a preliminary result that clarifies our interest in compositional abstract basic narrowing reduction. It basically states that in the abstract computations no solutions are lost, that is, each concrete computed answer is still 'represented' by a more general answer in the abstract semantics.

Theorem 18. *Let $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$ and $g' \propto g$. Then, for every solution $\theta \in \mathcal{O}_{\mathcal{R}}(\Leftarrow g)$ there exists $\kappa \in \Delta_{\mathcal{R}_{\mathcal{A}}}(\Leftarrow g')$ such that $\kappa \propto \theta$.*

Our analysis of unsatisfiability is formalized in the following theorem.

Corollary 19. *If $\Delta_{\mathcal{R}_{\mathcal{A}}}(\Leftarrow g) = \emptyset$, then g is unsatisfiable in $\mathcal{R}$.*

The following theorem constitutes the main result in this section and basically states that compositional abstract narrowing is compositional w.r.t. the AND operator.

Theorem 20. $\Delta_{\mathcal{R}_{\mathcal{A}}}(\Leftarrow g_1, g_2) = \Delta_{\mathcal{R}_{\mathcal{A}}}(\Leftarrow g_1) \Uparrow_{\mathcal{A}} \Delta_{\mathcal{R}_{\mathcal{A}}}(\Leftarrow g_2)$.

As a consequence of Theorem 20, the analysis for a specific goal (the abstract meaning of a goal) can be determined by exploiting the AND-compositionality of the basic narrowing semantics and its abstract version. In Section 6 we will formalize the idea that the compositionality of the abstract semantics w.r.t the union of $\mathcal{E}$-unification problems, as established in Theorem 20, provides for incrementality when dealing with constraint satisfaction problems in the framework of constraint logic programming, where sets of constraints are incrementally added to a solver.

6 Incremental Equational Analyzer

In the context of constraint logic programming [14,18], incremental search consists of proving the solvability of a sequence of constraint problems by transforming the existing solution to each previously solved problem into a solution to the next problem [13].

When dealing with equational constraints [1], the tests of solvability can be extremely redundant. Termination is not even guaranteed. In [2] we propose a lazy resolution procedure [14] which incorporates an analysis of unsatisfiability which allows us to avoid some useless computations. To achieve efficiency, the analyses also need to be incremental, that is, when adding a new equation set $\widetilde{c}$ to an already tested set c of constraints, the analysis should not start checking the accumulated constraint $c \cup \widetilde{c}$ from scratch. In this section, we formulate an incremental algorithm for analyzing the unsatisfiability of equation sets within a constraint setting [1]. The kernel of the algorithm is the calculus of compositional abstract (basic) narrowing reduction as formulated in Section 5.

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

Definition 21. (incremental constraint satisfaction problem)
Let $c_0, c_1, \widetilde{c}_1, \ldots, c_n, \widetilde{c}_n$ be constraints, where $c_i = c_{i-1} \cup \widetilde{c}_i$. The incremental constraint satisfaction problem consists of (efficiently) checking the (un)satisfiability of c_i by using some information from the computations of $c_0, \ldots, c_{i-1}$, $i = 1, \ldots, n$.

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

We define an *incremental Equational Analyzer (iEA)* as follows.

Definition 22. An *iEA*-state is a pair $\langle c, \Theta \rangle$, where c is a constraint and Θ is a set of substitutions. The empty *iEA*-state is $\langle \emptyset, \emptyset \rangle$.

Definition 23. (*iEA transition relation* $\xrightarrow{\widetilde{c}}_{iEA}$)

$$\frac{\Theta' = \Theta \Uparrow_{\mathcal{A}} \Delta_{\mathcal{R}_{\mathcal{A}}}(\Leftarrow \widetilde{c})}{\langle c, \Theta \rangle \xrightarrow{\widetilde{c}}_{iEA} \langle c \cup \widetilde{c}, \Theta' \rangle}$$

We note that, if the accumulated abstract success set $\Theta' = \emptyset$ then $c \cup \widetilde{c}$ is unsatisfiable by Corollary 19. Our strategy proves the unsatisfiability of $c \cup \widetilde{c}$, or it builds the (non-empty) abstract success set $\Delta_{\mathcal{R}_{\mathcal{A}}}(\Leftarrow c \cup \widetilde{c})$, as stated by:

Theorem 24. *Let c be a constraint and $\Theta = \Delta_{\mathcal{R}_{\mathcal{A}}}(\Leftarrow c) \neq \emptyset$. Then,*

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

We note that the computed abstract answer set $\Delta_{\mathcal{R}_{\mathcal{A}}}(\Leftarrow c \cup \widetilde{c})$ can be used to guide the final execution of a 'full' narrower which can find the concrete solutions and possibly recognize the unsatisfiability not detected by this lazy procedure, as described in [3].

The analysis above has been implemented in Prolog and tested on several programs with good results. To demonstrate this point empirically, consider the performance of the simple program `parity`:

```
X + 0      → X          ⇐
X + s(Y)   → s(X + Y)   ⇐
parity(X)  → even       ⇐ X = Y + Y
```

constraint	CAn	ICAn	AbNar	APCom
`parity(X) = even`	0.42	0.42	0.30	0.00
`parity(Y) = even`	3.88	2.80	0.32	1.20
`X + Y =` s^2`(0)`	**18.24**	**3.44**	0.44	2.92
`parity(X) = even`	0.40	0.40	0.28	0.00
`Y = s(0), Z = X + Y`	3.78	1.52	0.46	0.66
`X + Z =` s^3`(0)`	**26.36**	**4.44**	**fail**	
`X + Y =` s^4`(0)`	0.72	0.72	0.54	0.00
`parity(X) = even`	3.66	2.10	0.32	1.46
`parity(Y) = even`	**19.48**	**3.64**	0.30	2.62
`parity(X) = parity(Y)`	3.62	3.66	2.30	0.00
`X + Y = s(Z)`	23.18	12.00	0.46	9.52
`Z = 0`	**20.26**	**1.14**	**fail**	
`Z = 0, parity(X) = even`	0.52	0.54	0.42	0.00
`Y + Z =` s^2`(0)`	3.38	1.62	0.62	0.86
`Y = X + Z`	**10.62**	**5.24**	0.44	4.76

CAn Constraint Analyzer
ICAn Incremental Constraint Analyzer
AbNar Abstract Narrowing
APCom Abstract Parallel Composition

Table 1. *Incremental* vs. *Non-Incremental* constraint analyzer times (secs, using BIM-Prolog, SUN 3/80)

In Table 1 we report on some experiments we have performed for the case of a sequential implementation. We have not tried with the parallel interpreter yet. The time in the second column (ICAn) is the result of the sum of the time in the third and fourth column (AbNar and APCom) plus some extratime for some simplification rules which are only relevant for the implementation. We compare the time performances of the incremental vs. the non-incremental analyzers. If the incrementality was exploited our interpreter was able to achieve up to 95% gain in efficiency.

7 Conclusion and further research

The contribution of this paper is twofold. We have presented a formal compositional semantics for the success set of equational logic programs which is suitable for AND-parallel implementations. We have then shown that this semantics leads to compositional analyses and have given an example of an enhanced analysis of unsatisfiability which is suitable for theories where equations are considered as constraints.

The approach which we have taken is of general interest. In particular it applies to any kind of analysis where we look for properties which are satisfied by all (or some) success paths. For specific analyses, it will be necessary to provide the appropriate abstract domains and approximation of the term rewriting system. A groundness analysis which follows the approach proposed here is defined in [3].

References

1. 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*.
2. M. Alpuente, M. Falaschi, and F. Manzo. Analyses of Unsatisfiability for Equational Logic Programming. Technical Report DSIC-II/29/92, UPV, 1992. Short version 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. To appear in the *Journal of Logic Programming.*
3. M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. Narrowing Approximations as an Optimization for Equational Logic Programs. 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.
4. 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*, pages 171–182. IEEE Computer Society Press, 1994.
5. M. Alpuente, M. Falaschi, and G. Vidal. Semantics-Based Compositional Analysis for Equational Horn Programs. Technical Report DSIC-II/5/93, UPV, 1993.
6. R. Bert, R. Echahed, and B.M. Østvold. Abstract Rewriting. In *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.
7. M. Codish, M. Falaschi, and K. Marriott. Suspension Analyses for Concurrent Logic Programs. *ACM Transactions on Programming Languages and Systems*, 1994.
8. 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.
9. 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.
10. N. Dershowitz and N. Lindenstrauss. An Abstract Concurrent Machine for Rewriting. In H. Kirchner and W. Wechler, editors, *Proc. Second Int'l Conf. on Algebraic and Logic Programming*, volume 463 of *Lecture Notes in Computer Science*, pages 318–331. Springer-Verlag, Berlin, 1990.
11. M. Gabbrielli and G. Levi. On the Semantics of Logic Programs. In J. Leach Albert, B. Monien, and M. Rodriguez Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium*, volume 510 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, Berlin, 1991.
12. J.H. Gallier and S. Raatz. Extending SLD-resolution to equational Horn clauses using E-unification. *Journal of Logic Programming*, 6:3–43, 1989.

13. P. Van Hentenryck and T. Le Provost. Incremental Search in Constraint Logic Programming. *New Generation Computing*, 9:257–275, 1991.
14. M. Höhfeld and G. Smolka. Definite relations over constraint languages. Technical report, IBM Deutschland GmbH, Stuttgart, 1988.
15. S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1989.
16. 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.
17. J.-M. Jacquet. *Conclog: A Methodological Approach to Concurrent Logic Programming*. PhD thesis, University of Namur, Belgium, 1989.
18. 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.
19. J. Jaffar, J.-L. Lassez, and M.J. Maher. A theory of complete logic programs with equality. *Journal of Logic Programming*, 3:211–223, 1984.
20. 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.
21. H. Kuchen and W. Hans. An AND-Parallel Implementation of the Functional Logic Language Babel. In *Aachener Informatik-Bericht*, volume 12, pages 119–139, RWTH Aachen, 1991.
22. H. Kuchen, J.J. Moreno-Navarro, and M. Hermenegildo. Independent AND-Parallel Implementation of Narrowing. In M. Bruynooghe and M. Wirsing, editors, *Proc. of PLILP'92, Leuven (Belgium)*, volume 631 of *Lecture Notes in Computer Science*, pages 24–38. Springer-Verlag, Berlin, 1992.
23. 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.
24. M. J. Maher. On parameterized substitutions. Technical Report RC 16042, IBM - T.J. Watson Research Center, Yorktown Heights, NY, 1990.
25. A. Middeldorp and E. Hamoen. Completeness results for basic narrowing. *AAECC*, 5:213–253, 1994.
26. W. Nutt, P. Réty, and G. Smolka. Basic narrowing revisited. *Journal of Symbolic Computation*, 7:295–317, 1989.
27. 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.
28. U.S. Reddy. Narrowing as the Operational Semantics of Functional Languages. In *Proc. Second IEEE Int'l Symp. on Logic Programming*, pages 138–151. IEEE, 1985.
29. J.H. Siekmann. Unification Theory. *Journal of Symbolic Computation*, 7:207–274, 1989.

This article was processed using the LTEX macro package with LLNCS style