A Compositional Semantics for Conditional Term Rewriting Systems *

M. Alpuente[1] M. Falaschi[2] M. J. Ramis[1] G. Vidal[1]

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

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

Abstract

This paper considers compositions of conditional term rewriting systems as a basis for a modular approach to the design and analysis of equational logic programs. In this context, an equational logic program is viewed as consisting of a set of modules, each module defining a part of the program's functions. We define a compositional semantics for conditional term rewriting systems which we show to be rich enough to capture computational properties related to the use of logical variables. We also study how such a semantics may be safely approximated, and how the results of such approximations may be composed to yield a bottom-up abstract semantics adequate for modular data-flow analysis. A compositional analysis for equational unsatisfiability illustrates our approach.

1 Introduction

Conditional Term Rewriting Systems (CTRSs for short) are a natural computational paradigm for the integration of logic and equational programming [13,14,16,26]. In this context, programs are sets of conditional equations where the equation in the conclusion is implicitly oriented from left to right, and (some variant of) narrowing is used to solve equational goals, thus supporting logical variables and unification. For the completeness of implementations based on narrowing mechanisms, some form of confluence is needed [24]. The standard *initial algebra* semantics of a program $\mathcal{E}$, i.e. the quotient of the Herbrand Universe (or word algebra) by the congruence relation generated by $\mathcal{E}$, is known to be isomorphic to the least Herbrand E-model of the program, $\mathcal{M}(\mathcal{E})$, i.e. the set of all (ground) equations which hold in the underlying theory [16]. This semantics admits an equivalent fixpoint characterization and, for programs that can be oriented as a confluent term rewriting system $\mathcal{R}$, it can be further characterized as the set of all pairs of (ground) terms which can be reduced to a common $\mathcal{R}$-normal form [10,16,18].

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

It is widely acknowledged that one of the fundamental techniques in software engineering to deal with the complexity of large programs is to structure programs as modules which can then be composed together [14]. A modular structure of programs helps in designing and maintaining software. Moreover, proofs of correctness and data-flow analyses can be easier and more efficient when performed on small units of the programs. If we regard a program (module) as a set of clauses, then a natural notion of composition is given by their set union. A typical modular program is a program constructed according to the idea of module importing called *enrichment*, as formalized in algebraic specification theory [15] and later used in the algebraic foundation of modularized logic programming with equality [14]. Although it seems natural to think of programs as clause sets and of program composition as set union, it is less clear which semantics would be adequate for the program components. We look for a semantics which is compositional with respect to program union, i.e. for programs $\mathcal{E}_1$ and $\mathcal{E}_2$ the meaning of $\mathcal{E}_1 \cup \mathcal{E}_2$ can be derived by composing the meanings of $\mathcal{E}_1$ and $\mathcal{E}_2$. We call this property of the semantics OR-compositionality. OR-compositionality allows for incremental definition of structured programs and of their semantics, and provides a semantic basis for a modular approach to program analysis. Such an approach is significant for large program development since the alteration of a module does not require to recompute the semantics or to re-analyse the whole program. Thus, we also seek for a semantics which is able to model the properties of programs which are relevant for dataflow analyses, such as computed answer substitutions, call patterns, partial computed answer substitutions, etc.

We note that the least (ground) Herbrand E-model of a program is not rich enough to characterize properties like the set of computed answer substitutions for a query, as it is illustrated in the following example.

Example 1 *Consider the programs* $\mathcal{E}_1 = \{\ f(0) = 0 \ \Leftarrow\ \}$ *and* $\mathcal{E}_2 = \{\ f(0) = 0 \ \Leftarrow,$ $f(x) = 0 \ \Leftarrow \}$*. Both programs have the same least Herbrand E-model, namely* $\mathcal{M}(\mathcal{E}_1) = \mathcal{M}(\mathcal{E}_2) = \{0 = 0,\ 0 = f(0), \ldots,\ 0 = f^m(0), \ldots,\ f^n(0) = 0,\ f^n(0) = f(0), \ldots,$ $f^n(0) = f^m(0)\}$*. However, the two programs have a different behaviour in terms of computed answer substitutions. Consider, e.g., the goal* $\Leftarrow f(x) = 0$*, which computes the answer substitutions* $\{x/0\}$ *and the empty substitution* $\{\}$ *when executed in* $\mathcal{E}_2$*, while it only computes the substitution* $\{x/0\}$ *when executed in* $\mathcal{E}_1$*.*

We also note that the least Herbrand E-model is not adequate for compositionality. Indeed, the least Herbrand E-model of a program cannot be determined by the least Herbrand E-models of the clauses which compose the program, as illustrated by the following example.

Example 2 *Consider the programs* $\mathcal{E}_1 = \{a = b \Leftarrow b = c\}$ *and* $\mathcal{E}_2 = \{b = c \Leftarrow\}$*. Since* $\mathcal{M}(\mathcal{E}_1) = \{a = a,\ b = b,\ c = c\}$*,* $\mathcal{M}(\mathcal{E}_2) = \{b = b,\ c = c,\ b = c,\ c = b\}$*, and* $\mathcal{M}(\mathcal{E}_1 \cup \mathcal{E}_2) = \{a = b,\ b = a,\ a = c,\ c = a,\ a = a,\ b = b,\ c = c,\ b = c,\ c = b\}$*, hence* $\mathcal{M}(\mathcal{E}_1 \cup \mathcal{E}_2)$ *cannot be derived from the composition of* $\mathcal{M}(\mathcal{E}_1)$ *and* $\mathcal{M}(\mathcal{E}_2)$*.*

In this paper, we consider the problem of defining a semantics for CTRSs which correctly models the operational behaviour of equational logic programs in a compositional way. Our approach shares with the approach in [22,25] the idea of moving from a data-level semantics to a function-level semantics in order to capture compositionality. We extend the least Herbrand E-model semantics for CTRSs by considering interpretations which may contain non-ground equations. This allows us to model the set of computed answers for a given query. We introduce a semantics which is based on a (non-ground) immediate consequence operator $T^{ca}_{\mathcal{E}}$ which models the bottom-up one-step deduction (w.r.t. the equational theory $\mathcal{E}$) of non-ground equations from non-ground equations, and which we prove to be compositional w.r.t.

program union. In other words, we prove that $T_{\mathcal{E}}^{ca}$ contains the information necessary to describe the behaviour of a program $\mathcal{E}$ compositionally w.r.t. union in terms of computed answers.

1.1 Related work

[3] shows that computations and analysis of equational logic programs can be performed incrementally, by using the notion of AND-compositionality, that is, compositionality w.r.t. the conjunction of equations in a goal or in the condition of a clause. We do not consider AND-compositionality in this paper, and the results here are orthogonal to those in [3].

Starting with Dershowitz [9], several authors have studied modular aspects of term rewriting systems regarding the question of whether or not certain properties (such as confluence or termination) are preserved under composition of modules (e.g. [23,26]). All these works focus mainly on syntactic conditions to guarantee that the properties of confluence and termination are ensured in the composition, but they do not seek for a compositional semantics, which is the main concern of our work.

In the context of (pure) logic programming, [22,25] showed that, if the meaning of a logic program P is denoted by the standard immediate consequence operator T_P itself (rather than by its least fixpoint), then such a semantics is compositional for several interesting operators on programs including set union. However, this semantics does not characterize the computed answer observational equivalence on programs. [11,12] introduced models over Herbrand domains with variables and gave the first (non-compositional) semantics for positive logic programs modeling *computed answer substitutions (ca)*. Our approach is similar in that we also introduce variables in our domain of interpretation, but is different because of the technical problems related with semantic unification which are not present in the logic programming paradigm. We extend the principles underlying the *basic* strategy for narrowing [16,17,24], in which inferences are forbidden at terms introduced by substitutions in earlier inferences, to the case of equations in a fixpoint setting and we introduce also a transformation from equations to "flat" equations to deal with these complications.

1.2 Plan of the paper

The rest of the paper is organized as follows. Section 2 briefly presents some preliminary definitions and notations. Section 3.1 defines a non-compositional operational semantics $\mathcal{O}(\mathcal{E})$ modeling computed answers. Then, we characterize this semantics as the least fixpoint of the new immediate consequence operator $T_{\mathcal{E}}^{ca}$. In Section 3.2 we prove that $T_{\mathcal{E}}^{ca}$ is compositional w.r.t. program union. Section 4 introduces a bottom-up abstract fixpoint semantics that approximates success patterns (i.e. the set of the computed answer substitutions) and shows how it can be used for compositional analysis of modular programs. Section 5 concludes.

2 Preliminaries

Let us first summarize some known results about equations, conditional rewrite systems and equational unification. For full definitions refer to [10,19]. Throughout this paper, V will denote a countably infinite set of variables and Σ denotes a set of function symbols, or signature, each with a fixed associated arity. $\tau(\Sigma \cup V)$ and $\tau(\Sigma)$ denote the non-ground word (or term) algebra and the word algebra built on $\Sigma \cup V$ and Σ, respectively. $\tau(\Sigma)$ is usually called the Herbrand universe ($\mathcal{H}_\Sigma$) over Σ and it will be denoted by $\mathcal{H}$. $\mathcal{B}$ denotes the Herbrand base, namely the set of all ground equations which can be built with the elements of $\mathcal{H}$. A Σ-equation $s = t$ is a pair of terms $s, t \in \tau(\Sigma \cup V)$. A *program* is a Horn equational Σ-theory $\mathcal{E}$, which 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. We let $Goal$ denote the set of equational goals. A goal of the form $\Leftarrow x = y,\ x, y \in V$ is called a *trivial goal* and the equation $x = y$ is called a *trivial equation*. A *flat* equation is an equation of the form $f(x_1, \ldots, x_n) = x_{n+1}$ or $x_n = x_{n+1}$, where x_{n+1} and x_i are distinct variables, $i = 1, \ldots, n$. A *flat* equation set is a set of flat equations. Any set of equations S can be transformed into an equivalent one, $flat(S)$, which is flat [6,20]. For a given program $\mathcal{E}$, $C \ll \mathcal{E}$ denotes that C is a new variant of a clause in $\mathcal{E}$ such that C contains no variable previously met (standardised apart).

Terms are viewed as labelled trees in the usual way. Occurrences are represented by sequences, possibly empty, of natural numbers denoting an access path in a term. $\bar{O}(t)$ 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. By $Var(s)$ we denote the set of variables occurring in the syntactic object s, while $[s]$ denotes the set of ground instances of s. A *fresh* variable is a variable that appears nowhere else. The symbol $\tilde{}$ denotes a finite sequence of symbols. Identity of syntactic objects is denoted by $\equiv$.

We describe the lattice of syntactic 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 $true$, is implied by all elements of Eqn. We write $S \leq S'$ if S' logically implies S. Elements of Eqn are regarded as (quantified) conjunctions of equations and treated modulo logical equivalence. 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 S can be transformed into an equivalent one, $solve(S)$, 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 empty substitution is denoted by ϵ. A substitution $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ is a *unifier* of an equation set S iff $\widehat{\theta} = \{x_1 = t_1, \ldots, x_n = t_n\} \Rightarrow S$. We denote the set of unifiers of S by $unif(S)$ and $mgu(S)$ denotes the *most general unifier* of the unquantified equation set S. We write $mgu(\{s_1 = t_1, \ldots, s_n = t_n\}, \{s'_1 = t'_1, \ldots, s'_n = t'_n\})$ to denote the most general unifier of the set of equations $\{s_1 = s'_1, t_1 = t'_1, \ldots, s_n = s'_n, t_n = t'_n\}$. Observe that $mgu(\emptyset, \emptyset) = \epsilon$. The notions of application, composition and relative generality are defined in the usual way [4]. The notion of mgu extends to substitutions by $\theta_1 \Uparrow \theta_2 = mgu(\widehat{\theta_1} \cup \widehat{\theta_2})$. We write $\theta_{\lceil s}$ to denote the restriction of the substitution θ to the set of variables in the syntactic object s. The application of substitutions induces a preorder on terms. By $\approx$ we denote the associated equivalence relation (variance). We denote by E the set of equality axioms for a given program $\mathcal{E}$.

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 $\mathcal{E}$. We will denote by $\mathcal{H}/\mathcal{E}$ the finest partition $\tau(\Sigma)/=_{\mathcal{E}}$ induced by $=_{\mathcal{E}}$ over the set of ground terms $\tau(\Sigma)$. $\mathcal{H}/\mathcal{E}$ is usually called the *initial algebra* of $\mathcal{E}$ [10]. Satisfiability in $\mathcal{H}/\mathcal{E}$ is called *$\mathcal{E}$-unifiability*, that is, given a set of equations S, S is $\mathcal{E}$-unifiable iff there exists a substitution σ such that $\mathcal{E} \models S\sigma$ [10]. The substitution σ is called an $\mathcal{E}$-unifier of S.

A Herbrand interpretation I for a program $\mathcal{E}$ is a set of ground equations, with the understanding that $s = t$ is true w.r.t. I iff $s = t \in I$. A Herbrand interpretation satisfies a program clause iff, for each ground instance $\lambda = \rho \Leftarrow \widetilde{e}$ of the clause, we have that $\lambda = \rho \in I$ whenever $\widetilde{e} \subseteq I$. A Herbrand E-interpretation for $\mathcal{E}$ is a Herbrand interpretation for $\mathcal{E}$ obeying the equality axioms for $\mathcal{E}$. A Herbrand model for $\mathcal{E}$ is a Herbrand interpretation for $\mathcal{E}$ which satisfies each program clause in $\mathcal{E}$. A Herbrand E-model for $\mathcal{E}$ is a Herbrand model for $\mathcal{E}$ which satisfies the equality axioms for $\mathcal{E}$. The intersection of all Herbrand E-models for $\mathcal{E}$ is also a Herbrand E-model for $\mathcal{E}$ (the least Herbrand E-model), and it was proposed as the declarative semantics for positive programs [16]. This semantics is known to be isomorphic to the initial algebra $\mathcal{H}/\mathcal{E}$ of the program, and in the following will be denoted by $\mathcal{M}(\mathcal{E})$.

A Conditional Term Rewriting System 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 $Var(\rho) \cup Var(\widetilde{e}) \subseteq Var(\lambda)$. 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 conditional 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 the theories considered in this paper. The equational theory $\mathcal{E}$ is said to be canonical if the binary one-step rewriting relation $\rightarrow_{\mathcal{R}}$ defined by $\mathcal{R}$ is noetherian and confluent [19]. For canonical programs, $\mathcal{M}(\mathcal{E})$ is equivalent to the operational semantics given by the ground success set, i.e. the set of all ground equations $s = t$ such that s and t have a common $\mathcal{R}$-normal form, and to the fixpoint semantics given by the least fixpoint $T_{\mathcal{E}}\!\uparrow\!\omega$ of the following transformation $T_{\mathcal{E}}$ (immediate consequence operator), continuous on the complete lattice of Herbrand interpretations ordered by set inclusion [16].

$$T_{\mathcal{E}}(I) = \{\ t = t \in \mathcal{B}\ \} \ \cup\ \{\ e \in \mathcal{B} \mid \ (\lambda = \rho \Leftarrow \widetilde{e}) \in [\mathcal{E}],\ \{e[\rho]_u\} \cup \widetilde{e} \subseteq I,\ u \in Occ(e),\ e_{|u} = \lambda\ \}.$$

Informally, $T_{\mathcal{E}}(I)$ contains the set of all ground instances of the reflexivity axiom and the set of all ground equations that can be 'constructed' from elements of the Herbrand interpretation I by replacing one occurrence of the right-hand side of the head of a rule in $\mathcal{E}$ by the corresponding left-hand side.

The *narrowing* mechanism is a powerful tool for constructing complete $\mathcal{E}$-unification algorithms for useful classes of equational theories. In this context, completeness means that, for every solution to a given set of equations, a more general solution can be found by narrowing. Since unrestricted narrowing has quite a large search space, several strategies to improve the efficiency of narrowing have been devised. Basic conditional narrowing is a complete method for solving equations in the equational theory defined by a level-canonical CTRS [24]. All programs considered in this paper are assumed to be level-canonical, as well as their combinations. Level-canonicity implies canonicity. We formalize basic conditional narrowing as a transition system $(Goal \times Sub, \leadsto)$ whose transition relation $\leadsto\ \subseteq (Goal \times Sub) \times (Goal \times Sub)$ is defined as the smallest relation satisfying:

unification rule:

$$\frac{\sigma = mgu(G\theta)}{\langle \Leftarrow G, \theta\rangle \ \leadsto\ \langle \Leftarrow true, \theta\sigma\rangle}$$

narrowing rule:

$$\frac{e \in G \quad \wedge \quad u \in \bar{O}(e) \quad \wedge \quad (\lambda = \rho \Leftarrow \widetilde{e}) \ll \mathcal{E} \quad \wedge \quad \sigma = mgu(\{(e_{|u})\theta = \lambda\})}{\langle \Leftarrow G, \theta \rangle \quad \leadsto \quad \langle \Leftarrow (G \sim \{e\}) \cup \{e[\rho]_u\} \cup \widetilde{e}, \theta\sigma \rangle}$$

where the symbol $\sim$ stands for set difference. A *basic conditional narrowing derivation* is a sequence $\langle \Leftarrow G_0, \epsilon \rangle \leadsto \langle \Leftarrow G_1, \theta_1 \rangle \leadsto \ldots \leadsto \langle \Leftarrow G_n, \theta_n \rangle$. A successful derivation for $\Leftarrow G$ is a derivation $\langle \Leftarrow G, \epsilon \rangle \leadsto^* \langle \Leftarrow true, \theta \rangle$ and θ is called a *computed answer substitution* (for $\Leftarrow G$ in $\mathcal{E}$). Each *computed answer substitution* in a successful derivation for $\Leftarrow G$ is an $\mathcal{E}$-unifier of G. By abuse of notation, it is often called solution.

3 Compositional Semantics

In the next section we consider a (non-compositional) operational semantics for CTRSs modeling computed answer substitutions. When defining a semantics for a programming language it is essential to specify which properties of computations one is interested to 'observe'. In the case of equational logic programs, one may be interested to 'observe' the values computed by successful derivations (e.g. computed answer substitutions), by partial derivations or by finitely failed derivations. For each possible choice of an observable, a semantics which correctly models the observable would yield a different notion of program equivalence. So, for instance, two programs like those in Example 1 cannot be distinguished from the point of view of the ground success set, but have different computed answer substitutions. Throughout this paper we consider as observable the set of *computed answer substitutions*. In Section 3.2 we introduce a semantics which is compositional w.r.t. program union and for computed answer substitutions.

3.1 A Semantics Modeling Computed Answers

The operational semantics of a program is a mapping from the set of programs to a set of program denotations which, given a program $\mathcal{E}$, returns a set of 'results' of the computations in $\mathcal{E}$. For equational logic programs the operational semantics should be defined in terms of the set of all 'values' a functional expression can compute. In order to formulate a semantics modeling computed answers, the usual Herbrand base has to be extended to the set of all (possibly) non-ground equations modulo variance [11,12]. $\mathcal{H}_V$ denotes the *V-Herbrand universe* which allows variables in its elements, and is defined as $\tau(\Sigma \cup V)/_{\approx}$. For the sake of simplicity, the elements of $\mathcal{H}_V$ have the same representation as the elements of $\tau(\Sigma \cup V)$ and are also called terms. $\mathcal{B}_V$ denotes the *V-Herbrand base*, namely, the set of all equations $s = t$, where $s, t \in \mathcal{H}_V$. Note that the standard Herbrand base $\mathcal{B}$ is equal to $[\mathcal{B}_V]$. The power set of $\mathcal{B}_V$ is a complete lattice under set inclusion.

3.1.1 Operational Semantics

The operational semantics $\mathcal{O}(\mathcal{E})$ of an equational logic program $\mathcal{E}$ with computed answers as observable can be defined as follows.

Definition 3.1 *Let $\mathcal{E}$ be a program. Then,*
$\mathcal{O}(\mathcal{E}) = \{(f(\widetilde{x}) = y)\theta \mid f(\widetilde{x}) = y$ *is a flat equation,* $\langle \Leftarrow (f(\widetilde{x}) = y), \epsilon \rangle \leadsto^* \langle \Leftarrow true, \theta \rangle \}$.

Example 3 *Consider the programs $\mathcal{E}_1$ and $\mathcal{E}_2$ from Example 1. According to Definition 3.1, $\mathcal{O}(\mathcal{E}_1) = \{0 = 0,\ f(x) = f(x),\ f(0) = 0\}$, whereas $\mathcal{O}(\mathcal{E}_2) = \{0 = 0,\ f(x) = f(x),\ f(0) = 0,\ f(x) = 0\}$.*

The following theorem asserts that the computed answer substitutions of any (possibly conjunctive) goal $\Leftarrow G$ can be derived from $\mathcal{O}(\mathcal{E})$ (i.e. from the observable behaviour of single equations), by unification of the equations in the goal with the equations in the denotation. We note that this property is a kind of AND-compositionality which does not hold for ordinary (unrestricted) narrowing [3]. We assume that the equations in the denotation are renamed apart. Equations in the goal have to be flattened first, i.e. subterms have to be unnested so that the term structure is directly accessible to unification.

Definition 3.2 *Let $E = E_1 \cup E_2$ be a set of equations where all equations in E_2 are trivial equations and no equation in E_1 is trivial. We define the function $split : Eqn \rightarrow Eqn \times Eqn$ by $split(E) = (E_1, E_2)$.*

Theorem 3.3 (strong soundness and strong completeness)
Let $\mathcal{E}$ be a program and $\Leftarrow G \equiv\ \Leftarrow e_1, \ldots, e_n$ be a goal. Let $split(flat(G)) = (G_1, G_2)$. Then θ is a computed answer substitution for $\Leftarrow G$ in $\mathcal{E}$ iff there exist $G' \equiv e'_1, \ldots, e'_m \ll \mathcal{O}(\mathcal{E})$ such that $\theta' = mgu(G_1, G')$ and $\theta_{\restriction G} = (\theta' \Uparrow mgu(G_2))_{\restriction G}$.

Theorem 3.3 shows that $\mathcal{O}(\mathcal{E})$ is the fully abstract semantics w.r.t. computed answer substitutions.

Example 4 *Let us consider the program $\mathcal{E} = \{g(0) = 0 \Leftarrow,\ f(0) = 0 \Leftarrow,\ f(g(x)) = f(x) \Leftarrow\}$. According to Definition 3.1, $\mathcal{O}(\mathcal{E}) = \{0 = 0,\ g(x) = g(x),\ f(x) = f(x),\ g(0) = 0,\ f(0) = 0,\ f(g(x)) = f(x), \ldots,\ f(g^n(x)) = f(x),\ f(g(0)) = 0, \ldots,\ f(g^n(0)) = 0\}$. The goal $\Leftarrow G \equiv\ \Leftarrow y = f(z)$ computes the answers $\{\{y/f(z)\},\ \{y/0, z/0\},\ \{y/f(x), z/g(x)\},\ \{y/0, z/g(0)\}, \ldots,\ \{y/f(x), z/g^n(x)\}, \{y/0, z/g^n(0)\}\}$ in $\mathcal{E}$, which exactly coincides with the set of substitutions computed by unifying the flat equation $f(z) = y$ with the 'facts' in $\mathcal{O}(\mathcal{E})$.*

According to Theorem 3.3, $\mathcal{O}(\mathcal{E})$ can be used to simulate the execution for any goal G, that is, $\mathcal{O}(\mathcal{E})$ can be viewed as a (possibly infinite) set of 'unit' clauses, and the computed answer substitutions for $\Leftarrow G$ in $\mathcal{E}$ can be determined by 'executing' $flat(G)$ in the program $\mathcal{O}(\mathcal{E})$ by standard unification, as if the equality symbol were an ordinary predicate. In the following subsection we associate a continuous operator with an equational logic program, describe the corresponding fixpoint semantics, and show the relation with the operational semantics.

3.1.2 Fixpoint Semantics

We now introduce a new immediate consequence operator $T^{ca}_{\mathcal{E}}$ on a novel kind of 'interpretations' whose least fixpoint is shown to be equivalent to the computed answer substitutions semantics $\mathcal{O}(\mathcal{E})$. The main idea behind $\mathcal{T}$-interpretations is to keep apart equations and substitutions in order to avoid applying the $T^{ca}_{\mathcal{E}}$ operator on subterms which were introduced by instantiation, in a similar way as in the *basic* narrowing strategy, where the substitution part of prior narrowings is not subsequently narrowed. The definition of $\mathcal{T}$-interpretations is introduced mainly for technical reasons and the elements of a $\mathcal{T}$-interpretation will be often called 'equations'.

Definition 3.4 *A $\mathcal{T}$-interpretation is a set of tuples $\langle e, \theta\rangle$, where $e \in \mathcal{B}_V$ and $\theta \in Sub$. For any $\mathcal{T}$-interpretation $\mathcal{I} \in 2^{\mathcal{B}_V \times Sub}$ we let $\dot{\mathcal{I}}$ denote the V-interpretation $I = \{e\theta \mid \langle e, \theta\rangle \in \mathcal{I}\}$. For any V-interpretation $I \in 2^{\mathcal{B}_V}$, we let $\underset{.}{I}$ denote the $\mathcal{T}$-interpretation $\mathcal{I} = \{\langle e, \epsilon\rangle \mid e \in I\}$.*

Theorem 3.5 *Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be $\mathcal{T}$-interpretations. Let $\dot{\subseteq}$ be defined as follows: $\mathcal{I}_1 \dot{\subseteq} \mathcal{I}_2$ iff $\dot{\mathcal{I}_1} \subseteq \dot{\mathcal{I}_2}$. The set $2^{\mathcal{B}_V \times Sub}$ of $\mathcal{T}$-interpretations is a complete lattice w.r.t. $\dot{\subseteq}$.*

For any program $\mathcal{E}$, we denote by $\Phi_{\mathcal{E}}$ the set of identical equations $f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$, for each function symbol f occurring in $\mathcal{E}$, of any rank $n \geq 0$. As we will see, these *functional reflexivity axioms* play an important role in defining the fixpoint semantics of $\mathcal{E}$.

Definition 3.6 *Let $\mathcal{E}$ be a program and $\mathcal{I}$ be a $\mathcal{T}$-interpretation. Then,*

$$
\begin{array}{llll}
T^{ca}_{\mathcal{E}}(\mathcal{I}) = & \underset{.}{\Phi_{\mathcal{E}}} \;\cup & \{\; s \in \mathcal{B}_V \times Sub \mid & (\lambda = \rho \Leftarrow \tilde{e}) \ll \mathcal{E}, \\
 & & & \{\langle l = r, \vartheta\rangle\} \cup \tilde{s}' \subseteq \mathcal{I}, \\
 & & & mgu(flat(\tilde{e}), \dot{\tilde{s}'}) = \sigma, \\
 & & & mgu(\{\lambda = (r_{|u})\vartheta\}\sigma) = \theta, \\
 & & & u \in \bar{O}(r), \\
 & & & s = \langle l = r[\rho]_u, \vartheta\sigma\theta\rangle \;\}.
\end{array}
$$

The following proposition allows us to define a fixpoint semantics for equational logic programs.

Proposition 3.7 *The $T^{ca}_{\mathcal{E}}$ operator is continuous on the complete lattice of $\mathcal{T}$-interpretations.*

Definition 3.8 *Let $T(\mathcal{E}) = lfp(T^{ca}_{\mathcal{E}}) = T^{ca}_{\mathcal{E}} \uparrow \omega$. The least fixpoint semantics of a program $\mathcal{E}$ is defined as $\mathcal{F}(\mathcal{E}) = \dot{T}(\mathcal{E})$.*

The equivalence between the operational and the least fixpoint semantics is established by the following theorem.

Theorem 3.9 *Let $\mathcal{E}$ be a program. Then $\mathcal{F}(\mathcal{E}) = \mathcal{O}(\mathcal{E})$.*

The following theorem relates the non-ground semantics $\mathcal{F}(\mathcal{E})$ to the standard least Herbrand E-model semantics $\mathcal{M}(\mathcal{E})$.

Theorem 3.10 *Let $\mathcal{E}$ be a program and $r^{\mp}$ be the transitive-symmetric closure of relation r under replacement (the f-*substitutivity *property). Then, $\mathcal{M}(\mathcal{E}) = [\mathcal{O}(\mathcal{E})]^{\mp}$.*

According to Theorem 3.10, the new semantics $\mathcal{O}(\mathcal{E})$ can be thought of as a non-ground representation of the standard least Herbrand E-model semantics $\mathcal{M}(\mathcal{E})$ just containing the necessary information which allows one to determine the computed answers, as it was established by Theorem 3.3.

3.2 A Compositional Semantics

A semantics $\mathcal{S}$ is compositional, or homomorphic, w.r.t. a program composition operator $\star$ if the meaning (semantics) $\mathcal{S}(\mathcal{C}_1 \star \mathcal{C}_2)$ of a compound construct can be determined by composing the meanings of the constituents $\mathcal{S}(\mathcal{C}_1)$ and $\mathcal{S}(\mathcal{C}_2)$, i.e. for a suitable homomorphism $f_\star$, $\mathcal{S}(\mathcal{C}_1 \star \mathcal{C}_2) = f_\star(\mathcal{S}(\mathcal{C}_1), \mathcal{S}(\mathcal{C}_2))$. The semantics that we have considered so far is not compositional w.r.t. union of programs as shown by the following example.

Example 5 *Let us consider the programs* $\mathcal{E}_1 = \{f(x) = 0 \Leftarrow f(0) = 0\}$ *and* $\mathcal{E}_2 = \{f(0) = 0 \Leftarrow\}$. *According to Definition 3.1,* $\mathcal{O}(\mathcal{E}_1) = \{0 = 0,\ f(x) = f(x)\}$, *and* $\mathcal{O}(\mathcal{E}_2) = \{0 = 0,\ f(x) = f(x),\ f(0) = 0\}$. *Since* $\mathcal{O}(\mathcal{E}_1 \cup \mathcal{E}_2) = \{0 = 0,\ f(x) = f(x),\ f(0) = 0,\ f(x) = 0\}$, *the semantics of the union of the programs cannot be obtained from the semantics of the programs.*

Example 5 shows that the denotation of programs in terms of sets of (possibly) non-ground equations is not adequate to model OR-compositionality. According to the idea in [22,25], we suggest to denote each program by the corresponding immediate operator $T^{ca}_{\mathcal{E}}$. Note that the semantics of $\mathcal{E}$ is no longer viewed just as a set of equations, but as a function which takes a set of 'equations' and deduces new 'equations' from it.

Definition 3.11 (compositional semantics)
The compositional semantics for a program $\mathcal{E}$ *is defined as* $\mathcal{S}(\mathcal{E}) = T^{ca}_{\mathcal{E}}$.

The meaning of a program $\mathcal{E}$ is therefore denoted by the mapping which, given a $\mathcal{T}$-interpretation $\mathcal{I}$, yields the set of immediate consequences of $\mathcal{I}$ in $\mathcal{E}$, that is $\forall \mathcal{I} \in 2^{\mathcal{B}_V \times Sub}$. $\mathcal{S}(\mathcal{E})(\mathcal{I}) = T^{ca}_{\mathcal{E}}(\mathcal{I})$. The following result shows that $\mathcal{S}(\mathcal{E})$ models computed answer substitutions in a compositional way.

Theorem 3.12 (compositionality)
Let $\mathcal{E}_1, \mathcal{E}_2$ *be programs. Let* $T^{ca}_{\mathcal{E}_1}$ *and* $T^{ca}_{\mathcal{E}_2}$ *be their associated denotations. Then,* $\forall \mathcal{I} \in 2^{\mathcal{B}_V \times Sub}$. $T^{ca}_{\mathcal{E}_1 \cup \mathcal{E}_2}(\mathcal{I}) = T^{ca}_{\mathcal{E}_1}(\mathcal{I}) \cup T^{ca}_{\mathcal{E}_2}(\mathcal{I})$.

The semantics $\mathcal{S}(\mathcal{E})$ can be considered as a semantic basis for modular analysis of equational logic programs since, by using suitable abstractions of $T^{ca}_{\mathcal{E}}$, we can analyze program constituents w.r.t. several interesting program properties and then compose the results to obtain the analysis of the whole program [5], as we formalize in the following section.

4 Compositional Abstract Semantics

The theory of abstract interpretation [8] provides a formal framework to develop advanced data-flow analysis tools. In this section, starting from the fixpoint semantics in Section 3, we develop an abstract semantics which approximates the observables and is adequate for modular data-flow analysis, such as the analysis of unsatisfiability of equation sets. We assume the framework of abstract interpretation for analysis of equational unsatisfiability as defined in [1,2]. We first recall the abstract domains and the associated abstract operators. Then we describe a novel abstract immediate consequence operator $T^{\sharp ca}_{\mathcal{E}}$ able to approximate the operator $T^{ca}_{\mathcal{E}}$, and the abstract fixpoint semantics $\mathcal{F}^{\sharp}(\mathcal{E})$. In the following we denote the abstract analog of a concrete object O by $O^{\sharp}$.

4.1 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 $E = Eqn$ or $E = Sub$, the description is called an *equation description* or a *substitution description*, respectively. The correspondence between the abstract and concrete domain is established through a 'concretization' function $\gamma : D \rightarrow 2^E$. 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 [1].

Abstract equations and abstract substitutions correspond to abstract program denotations and abstract observable properties, respectively. The domains for equations and substitutions are as follows.

Definition 4.1 (abstract Herbrand universe)
By $\mathcal{H}_V = (\tau(\Sigma \cup V), \leq)$, 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 $\mathcal{H}_V^\sharp = (\tau(\Sigma \cup V \cup \{\perp\}), \preceq)$ be the domain of terms over the signature augmented by $\perp$, where the partial order $\preceq$ is defined as follows:
(a) $\forall t \in \mathcal{H}_V^\sharp . \perp \preceq t$ and $t \preceq t$ and
(b) $\forall s_1, \ldots, s_n, s_1', \ldots, s_n' \in \mathcal{H}_V^\sharp, \forall f/n \in \Sigma.\ s_1' \preceq s_1 \wedge \ldots \wedge s_n' \preceq s_n \Rightarrow f(s_1', \ldots, s_n') \preceq f(s_1, \ldots, s_n)$
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 $e' \preceq e$. Note that $S' \preceq true \Rightarrow S' \equiv true$.
2) $S' \sqsubseteq S$ iff $(S' \preceq S)$ and $(S \preceq S'$ implies $S' \subseteq S)$.

Roughly speaking, the special symbol $\perp$ introduced in the abstract domains represents any concrete term. The behaviour of the symbol $\perp$ from a programming viewpoint resembles that of an "anonymous" variable in Prolog. From the viewpoint of logic, $\perp$ stands for an existentially quantified variable [1,21]. 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 4.2 *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 is given by logical implication: let $\theta, \kappa \in Sub^\sharp$, $\kappa \preceq \theta$ iff $[\![\widehat{\theta}]\!] \Rightarrow [\![\widehat{\kappa}]\!]$.*

The descriptions for terms, substitutions and equations are as follows.

Definition 4.3 *Let $\mathcal{H}_V = (\tau(\Sigma \cup V), \leq)$ and $\mathcal{H}_V^\sharp = (\tau(\Sigma \cup V \cup \{\perp\}), \preceq)$. The term description is $\langle \mathcal{H}_V^\sharp, \gamma, \mathcal{H}_V \rangle$ where $\gamma : \mathcal{H}_V^\sharp \rightarrow 2^{\mathcal{H}_V}$ is defined by: $\gamma(t') = \{t \in \mathcal{H}_V | t' \preceq t\}$.*

Let Eqn be the set of finite sets of equations over $\tau(\Sigma \cup V)$ and $Eqn^\sharp$ be the set of finite sets of equations over $\tau(\Sigma \cup V \cup \{\perp\})$. The equation description is $\langle (Eqn^\sharp, \sqsubseteq), \gamma, (Eqn, \leq) \rangle$, where $\gamma : Eqn^\sharp \rightarrow 2^{Eqn}$ is defined by: $\gamma(g') = \{g \in Eqn | g' \sqsubseteq g$ and g is unquantified $\}$.

Let Sub be the set of substitutions over $\tau(\Sigma \cup V)$ and $Sub^\sharp$ be the set of substitutions over $\tau(\Sigma \cup V \cup \{\perp\})$. The substitution description is $\langle (Sub^\sharp, \preceq), \gamma, (Sub, \leq) \rangle$, where $\gamma : Sub^\sharp \rightarrow 2^{Sub}$ is defined by: $\gamma(\kappa) = \{\theta \in Sub | \kappa \preceq \theta\}$.

In order to perform computations over the abstract domains, we have to define the notion of *abstract unification*. We define the abstract most general unifier for an equation set $S' \in Eqn^\sharp$ as follows. Let $\exists y_1 \ldots y_n . S = solve([\![S']\!])$ and $\kappa = \{y_1/\perp, \ldots, y_n/\perp\}$. Then $mgu^\sharp(S') = S\kappa$. The fact that $\forall \theta \in unif([\![S]\!]).\ mgu^\sharp(S) \preceq \theta$ justifies our use of 'most general'. The safety of the abstract unification algorithm has been proven in [1].

Our analysis is based on a form of simplified (abstract) program which always terminates and in which the query can be executed efficiently. Our notion of abstract program is parametric with respect to a loop-check. Two different instances can be found in [1,2].

Definition 4.4 *A loop-check is a graph $\mathcal{G}_{\mathcal{E}}$ associated with a program $\mathcal{E}$, i.e. a relation consisting of a set of pairs of terms, such that: (1) the transitive closure $\mathcal{G}_{\mathcal{E}}^{+}$ is decidable and (2) Let $\mathring{t} = t'$ be a function which assigns to a term t some node t' in $\mathcal{G}_{\mathcal{E}}$. If there is an infinite sequence: $\langle \Leftarrow G_0, \theta_0\rangle \leadsto \langle \Leftarrow G_1, \theta_1\rangle \leadsto \ldots$ then $\exists i \geq 0.\ \langle \mathring{t}_i, \mathring{t}_i\rangle \in \mathcal{G}_{\mathcal{E}}^{+}$, where $t_i = e_{|u}\theta_i$, $e \in G_i$ and $u \in \bar{O}(e)$. (we refer to $\langle \mathring{t}_i, \mathring{t}_i\rangle$ as a 'cycle' of $\mathcal{G}_{\mathcal{E}}$.)*

A program is abstracted by simplifying the right-hand side and the body of each clause. This definition is given inductively on the structure of terms and equations. The main idea is that terms whose corresponding nodes in $\mathcal{G}_{\mathcal{E}}$ have a cycle are drastically simplified by replacing them by $\perp$.

Definition 4.5 (abstract program)
Let $\mathcal{E}$ be a program. Let $\mathcal{G}_{\mathcal{E}}$ be a loop-check for $\mathcal{E}$. We define the abstraction of $\mathcal{E}$ as follows: $\mathcal{E}^{\sharp} = \{\lambda = sh(\rho, \mathcal{G}_{\mathcal{E}}) \Leftarrow sh(\widetilde{e}, \mathcal{G}_{\mathcal{E}}) \mid (\lambda = \rho \Leftarrow \widetilde{e}) \in \mathcal{E}\}$, where the shell $sh(x, \mathcal{G})$ of an expression x according to a loop-check $\mathcal{G}$ is defined inductively

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

We can now formalize the abstract semantics.

4.2 Bottom-up Abstract Semantics

We define an abstract fixpoint semantics in terms of the least fixpoint of a continuous transformation $T_{\mathcal{E}}^{\sharp_{ca}}$ based on abstract unification and the operation of abstraction of a program. The idea is to provide a finitely computable approximation of the concrete denotation of the program $\mathcal{E}$. In the following, we define the abstract transformation $T_{\mathcal{E}}^{\sharp_{ca}}$.

Definition 4.6 (abstract Herbrand base, abstract Herbrand interpretation)
The abstract Herbrand base of equations $\mathcal{B}_V^{\sharp}$ is defined as the set of equations over the abstract Herbrand universe $\mathcal{H}_V^{\sharp}$. An abstract Herbrand interpretation is any element of $2^{\mathcal{B}_V^{\sharp}}$. An abstract $\mathcal{T}$-interpretation is an element of $2^{\mathcal{B}_V^{\sharp} \times Sub^{\sharp}}$. A partial order $\dot{\subseteq}^{\sharp}$ on abstract $\mathcal{T}$-interpretations can be defined in a similar way to the order $\dot{\subseteq}$ on $\mathcal{T}$-interpretations.

We can easily show that the set of abstract $\mathcal{T}$-interpretations is a complete lattice w.r.t. $\dot{\subseteq}^{\sharp}$.

Definition 4.7 *Let $\mathcal{E}$ be a program and $\mathcal{G}_{\mathcal{E}}$ be a loop-check for $\mathcal{E}$. Let $\mathcal{I}$ be an abstract $\mathcal{T}$-interpretation. Then,*

$$\begin{array}{llll} T_{\mathcal{E}}^{\sharp_{ca}}(\mathcal{I}) = & \Phi_{\mathcal{E}} \quad \cup \quad \{ \ s \in \mathcal{B}_V^{\sharp} \times Sub^{\sharp} \mid & (\lambda = \rho \Leftarrow \widetilde{e}) \ll \mathcal{E}^{\sharp}, \\ & & \{\langle l = r, \vartheta\rangle\} \cup \widetilde{s'} \subseteq \mathcal{I}, \\ & & mgu^{\sharp}(flat(\widetilde{e}), \widetilde{s'}) = \sigma, \\ & & mgu^{\sharp}(\{\lambda = (r_{|u})\vartheta\}\sigma) = \theta, \\ & & u \in \bar{O}(r), \\ & & s = \langle l = r[\rho]_u, \vartheta\sigma\theta\rangle \ \}. \end{array}$$

Proposition 4.8 *The $T_{\mathcal{E}}^{\sharp ca}$ operator is continuous on the complete lattice of abstract $\mathcal{T}$-interpretations.*

Definition 4.9 (abstract least fixpoint semantics)
Let $T^{\sharp}(\mathcal{E}) = lfp(T_{\mathcal{E}}^{\sharp ca})$. The abstract least fixpoint semantics of a program $\mathcal{E}$ is $\mathcal{F}^{\sharp}(\mathcal{E}) = \dot{T}^{\sharp}(\mathcal{E})$.

The following theorem states that $\mathcal{F}^{\sharp}(\mathcal{E})$ is always finitely computable.

Theorem 4.10 *There exists a finite positive number k such that $T^{\sharp}(\mathcal{E}) = T_{\mathcal{E}}^{\sharp ca} \uparrow k$.*

From a semantics viewpoint, given a program $\mathcal{E}$, the fixpoint semantics $\mathcal{F}(\mathcal{E})$ is approximated by the corresponding abstract fixpoint semantics $\mathcal{F}^{\sharp}(\mathcal{E})$. That is, we can compute an abstract approximation of the concrete semantics in a finite number of steps. The correctness of the abstract fixpoint semantics with respect to the concrete semantics is proved by the following:

Theorem 4.11 *There exists a finite positive number k such that $T_{\mathcal{E}}^{\sharp ca} \uparrow k \propto T_{\mathcal{E}}^{ca} \uparrow \omega$.*

Corollary 4.12 *$\mathcal{F}^{\sharp}(\mathcal{E}) \propto \mathcal{F}(\mathcal{E})$.*

The semantics $\mathcal{F}^{\sharp}(\mathcal{E})$ collects goal-independent information about success patterns of a given program. The relation between the abstract fixpoint and the concrete operational semantics (computed answer substitutions) is given by the following theorem. Roughly speaking, given a goal $\Leftarrow G$, we obtain a description of the set of the computed answers of $\Leftarrow G$ by abstract unification of the equations in $flat(G)$ with equations in the approximated semantics $\mathcal{F}^{\sharp}(\mathcal{E})$.

Theorem 4.13 (strong completeness)
Let $\mathcal{E}$ be a program and $\Leftarrow G \equiv\ \Leftarrow e_1, \ldots, e_n$ be a goal. If θ is a computed answer substitution for $\Leftarrow G$ in $\mathcal{E}$, then there exists $G' \equiv e'_1, \ldots, e'_m \ll \mathcal{F}^{\sharp}(\mathcal{E})$ such that $\theta' = mgu^{\sharp}(flat(G), G')$ and $\theta' \preceq \theta$.

Example 6 *The analysis of the program $\mathcal{E} = \{g(0) = c(0) \Leftarrow,\ h(0) = 0 \Leftarrow,\ h(c(0)) = c(0) \Leftarrow,\ h(g(x)) = c(h(x)) \Leftarrow g(x) = c(x)\}$ using the loop-check $\mathcal{G}_{\mathcal{E}} = \{\langle h(x), h(x)\rangle\}$ returns the set $\mathcal{F}^{\sharp}(\mathcal{E}) = \{0 = 0,\ g(x) = g(x),\ h(x) = h(x),\ c(x) = c(x),\ g(0) = c(0),\ h(0) = 0,\ h(c(0)) = c(0),\ h(g(0)) = c(\perp)\}$, which approximates the program success set. Given a goal $\Leftarrow G \equiv\ \Leftarrow h(g(x)) = y$, basic conditional narrowing computes the substitutions $\{\{y/h(g(x))\},\ \{x/0,\ y/h(c(0))\},\ \{x/0,\ y/c(0)\},\ \{x/0,\ y/c(h(0))\}\}$ and then goes on indefinitely. The abstract substitutions returned by the abstract unification of the equations in the flattened goal $\Leftarrow h(z) = y,\ g(x) = z$ into the abstract denotation $\mathcal{F}^{\sharp}(\mathcal{E})$ are $\{\{y/h(g(x))\},\ \{x/0,\ y/h(c(0))\},\ \{x/0,\ y/c(0)\}, \{x/0,\ y/c(\perp)\}\}$, which approximate the computed answers of $\Leftarrow G$.*

The following corollary represents the main result in this section.

Corollary 4.14 *If $mgu^{\sharp}(flat(G), G') = fail$ for all $G' \equiv e'_1, \ldots, e'_n \ll \mathcal{F}^{\sharp}(\mathcal{E})$, then G is unsatisfiable in $\mathcal{E}$.*

Example 7 *The goal $\Leftarrow h(g(x)) = x$ is unsatisfiable in the program $\mathcal{E}$ of Example 6.*

We are then left with the problem of defining the compositional abstract semantics.

4.3 Compositional Abstract Semantics

Following the approach to program compositionality in Section 3.2, programs are denoted by their immediate consequence operators. Hence, it seems reasonable to expect that the abstract transformation maps could also be composed similarly, i.e. we may expect that $T^{\sharp_{ca}}_{\mathcal{E}_1 \cup \mathcal{E}_2} = \lambda \mathcal{I}. T^{\sharp_{ca}}_{\mathcal{E}_1}(\mathcal{I}) \cup T^{\sharp_{ca}}_{\mathcal{E}_2}(\mathcal{I})$. The following example shows that such a simple property does not hold for the abstract immediate consequence operator $T^{\sharp_{ca}}_{\mathcal{E}}$ defined in Section 4.2.

Example 8 *Consider the programs* $\mathcal{E}_1 = \{a = b \Leftarrow,\ c = a \Leftarrow\}$, $\mathcal{E}_2 = \{b = a \Leftarrow\}$. *Since there is no cycle in the definition of* $\mathcal{E}_1$ *or* $\mathcal{E}_2$, *then we assume* $\mathcal{E}_1^\sharp = \mathcal{E}_1$ *and* $\mathcal{E}_2^\sharp = \mathcal{E}_2$, *while* $(\mathcal{E}_1 \cup \mathcal{E}_2)^\sharp = \{a = \bot \Leftarrow,\ b = \bot \Leftarrow,\ c = \bot \Leftarrow\}$. *Let us consider* $I = \{c = c\}$. *Then,* $\dot{T}^{\sharp_{ca}}_{\mathcal{E}_1}(\dot{I}) = \{a = a,\ b = b,\ c = c,\ c = a\}$, $\dot{T}^{\sharp_{ca}}_{\mathcal{E}_2}(\dot{I}) = \{a = a,\ b = b\}$, *whereas* $\dot{T}^{\sharp_{ca}}_{\mathcal{E}_1 \cup \mathcal{E}_2}(\dot{I}) = \{a = a,\ b = b,\ c = c,\ c = \bot\}$.

Roughly speaking, what is wrong with Example 8 is the fact that the abstract program $(\mathcal{E}_1 \cup \mathcal{E}_2)^\sharp$ may not be obtained by the simple union of the abstract clauses of $\mathcal{E}_1^\sharp$ and $\mathcal{E}_2^\sharp$. In fact, when we compose $\mathcal{E}_1^\sharp$ and $\mathcal{E}_2^\sharp$, information about possibly new cycles is not kept in this formalization. Hence, a further abstraction process is needed in order to capture cycles which possibly arise when combining modules, as formalized in the following theorem.

Theorem 4.15 *Let* $\mathcal{E}_1, \mathcal{E}_2$ *be programs. Then,* $\forall \mathcal{I} \in 2^{\mathcal{B}^\sharp_V \times Sub^\sharp}$. $T^{\sharp_{ca}}_{\mathcal{E}_1 \cup \mathcal{E}_2}(\mathcal{I}) = T^{\sharp_{ca}}_{\mathcal{E}_1}(\mathcal{I}) \cup^\sharp T^{\sharp_{ca}}_{\mathcal{E}_2}(\mathcal{I})$ *where the* abstract union $\cup^\sharp$ *of abstract Herbrand interpretations gives us the ability of considering the information about cycles present in the loop-check* $\mathcal{G}_{\mathcal{E}_1 \cup \mathcal{E}_2}$ *induced by the combined program* $\mathcal{E}_1 \cup \mathcal{E}_2$, *in the following way:*

$$T^{\sharp_{ca}}_{\mathcal{E}_1}(\mathcal{I}) \cup^\sharp T^{\sharp_{ca}}_{\mathcal{E}_2}(\mathcal{I}) = sh^\sharp(T^{\sharp_{ca}}_{\mathcal{E}_1}(\mathcal{I}) \cup T^{\sharp_{ca}}_{\mathcal{E}_2}(\mathcal{I}), \mathcal{G}_{\mathcal{E}_1 \cup \mathcal{E}_2}),$$

where, if $\Upsilon(\mathcal{I}) = \bigcup_{\langle f(x) = f(x), \epsilon \rangle \in \mathcal{I}} \{\langle f(x) = f(x), \epsilon \rangle\}$, *then*

$$sh^\sharp(\mathcal{I}) = \Upsilon(\mathcal{I}) \cup \bigcup_{\langle l = r, \theta \rangle \in\ \mathcal{I} \sim \Upsilon(\mathcal{I})} \{\langle l = sh(r, \mathcal{G}), \theta \rangle\}.$$

We note that $\mathcal{G}_{\mathcal{E}_1 \cup \mathcal{E}_2}$ can be derived in a compositional way from $\mathcal{G}_{\mathcal{E}_1}$ and $\mathcal{G}_{\mathcal{E}_2}$. However, because of space limitations we leave this out.

The abstract $T^{\sharp_{ca}}_{\mathcal{E}}$ operator can be thought of as a tool specialized in the analysis of equational unsatisfiability. The algebraic compositionality of $T^{\sharp_{ca}}_{\mathcal{E}}$ allows for the algebraic composition of analysis for different program modules, as will be shown in our last example.

Example 9 *Consider the programs*
$\mathcal{E}_1 = \{x + 0 = x\ \Leftarrow,\ x + s(y) = s(x + y)\ \Leftarrow\}$,
$\mathcal{E}_1^\sharp = \{x + 0 = x\ \Leftarrow,\ x + s(y) = s(\bot)\ \Leftarrow\}$ *and*
$\mathcal{E}_2 = \{addw([\,]) = 0\ \Leftarrow,\ addw([x|y]) = weight(x) + addw(y)\}$,
$\mathcal{E}_2^\sharp = \{addw([\,]) = 0\ \Leftarrow,\ addw([x|y]) = \bot\ \Leftarrow\}$.

Then we have,
$T^{\sharp_{ca}}_{\mathcal{E}_1 \cup \mathcal{E}_2}(\emptyset) = T^{\sharp_{ca}}_{\mathcal{E}_1}(\emptyset) \cup^\sharp T^{\sharp_{ca}}_{\mathcal{E}_2}(\emptyset) = \{\langle 0 = 0, \epsilon\rangle,\ \langle s(x) = s(x), \epsilon\rangle,\ \langle x + y = x + y, \epsilon\rangle,\ \langle [\,] = [\,], \epsilon\rangle,\ \langle [x|y] = [x|y], \epsilon\rangle,\ \langle addw(x) = addw(x), \epsilon\rangle,\ \langle weight(x) = weight(x), \epsilon\rangle\}$.
$T^{\sharp_{ca}}_{\mathcal{E}_1 \cup \mathcal{E}_2}(T^{\sharp_{ca}}_{\mathcal{E}_1 \cup \mathcal{E}_2}(\emptyset)) = T^{\sharp_{ca}}_{\mathcal{E}_1}(T^{\sharp_{ca}}_{\mathcal{E}_1 \cup \mathcal{E}_2}(\emptyset)) \cup^\sharp T^{\sharp_{ca}}_{\mathcal{E}_2}(T^{\sharp_{ca}}_{\mathcal{E}_1 \cup \mathcal{E}_2}(\emptyset)) = \{\langle 0 = 0, \epsilon\rangle,\ \langle s(x) = s(x), \epsilon\rangle,\ \langle x + y = x + y, \epsilon\rangle, \langle [\ ] = [\ ], \epsilon\rangle,\ \langle [x|y] = [x|y], \epsilon\rangle,\ \langle addw(x) = addw(x), \epsilon\rangle, \langle weight(x) = weight(x), \epsilon\rangle\}\ \cup\ \{\langle x + y = x, \{y/0\}\rangle,\ \langle x + y = s(\bot), \{y/s(y')\}\rangle,\ \langle addw(x) = 0, \{x/[\,]\}\rangle,\ \langle addw(x) = \bot, \{y/[x'|y'']\}\rangle\}$. *(fixpoint)*

5 Conclusions

We have presented an approach to the semantics of conditional term rewriting systems able to capture the natural ability of equational logic programs to compute answers. We have also shown that it is possible to formulate an equivalent characterization of this semantics as the least fixpoint of a suitable continuous transformation, which we have proven to be compositional w.r.t. program union. We have finally presented an abstraction of this fixpoint semantics which yields an approximated finite (goal-independent) description of the success patterns of the program. A compositional analysis of equational unsatisfiability has been developed to illustrate our approach.

The concrete and the abstract semantics can support modular development and analysis of large programs. To the best of our knowledge this is the first compositional analysis for equational Horn programs.

References

[1] M. Alpuente, M. Falaschi, and F. Manzo. Analyses of Unsatisfiability for Equational Logic Programming. Technical Report DSIC-II/29/92, UPV, 1992. To appear in the *Journal of Logic Programming*. Short version in *Proc. of PLILP'92, Leuven (Belgium)*, volume 631 of *Lecture Notes in Computer Science*, pages 443-457. Springer-Verlag, 1992.

[2] M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. Narrowing Approximations as an optimization for Equational Logic Programs. In M. Bruynooghe and J. Penjam, editors, *Proc. of PLILP'93, Tallinn (Estonia)*, volume 714 of *Lecture Notes in Computer Science*, pages 391–403. Springer-Verlag, Berlin, 1993.

[3] 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.

[4] 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.

[5] R. Barbuti, R. Giacobazzi, and G. Levi. A General Framework for Semantics-based Bottom-up Abstract Interpretation of Logic Programs. *ACM Transactions on Programming Languages and Systems*, 15(1):133–181, 1993.

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

[7] 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.

[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. Termination of Linear Rewriting Systems. In *Proc. 8th Int'l Colloquium on Automata, Languages and Programming ICALP'81*, volume 115 of *Lecture Notes in Computer Science*, pages 448–458. Springer-Verlag, Berlin, 1981.

[10] 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.

[11] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative Modeling of the Operational Behavior of Logic Languages. *Theoretical Computer Science*, 69(3):289–318, 1989.

[12] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs. *Information and Computation*, 103(1):86–113, 1993.

[13] L. Fribourg. A Narrowing Procedure for Theories with Constructors. In *7th Int'l Conf. on Automated Deduction CADE'84*, pages 259–278, 1984.

[14] J. Goguen and J. Meseguer. Models and Equality for Logical Programming. In G. Levi H. Ehrig, R. Kowalski and U. Montanari, editors, *Proc. TAPSOFT'87*, volume 250 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, Berlin, 1987. Volume 2.

[15] J.J. Goguen, J. Thatcher, and E. Wagner. *An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types*, volume 4, pages 80–149. Prentice-Hall, 1978.

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

[17] J.M. Hullot. Canonical Forms and Unification. In *5th Int'l Conf. on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Springer-Verlag, Berlin, 1980.

[18] S. Kaplan. Conditional Rewrite Rules. *Theoretical Computer Science*, 33:175–193, 1984.

[19] J.W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume I. Oxford University Press, 1991.

[20] E. Knill, P.T. Cox, and T. Pietrzykowski. Equality and abductive residua for Horn Clauses. *Theoretical Computer Science*, 120:1–44, 1993.

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

[22] P. Mancarella and D. Pedreschi. An Algebra of Logic Programs. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth Int'l Conf. on Logic Programming*, pages 1006–1023. The MIT Press, Cambridge, Mass., 1988.

[23] A. Middeldorp. Modular Properties of Conditional Term Rewriting Constructor Systems. Technical Report CS-R9105, Centre for Mathematics and Computer Science, Amsterdam, 1991.

[24] A. Middeldorp and E. Hamoen. Completeness Results for Basic Narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5:213–253, 1994.

[25] R. A. O'Keefe. Towards an Algebra for Constructing Logic Programs. In *Proc. IEEE Symp. on Logic Programming*, pages 152–160. IEEE, 1985.

[26] M. Rusinowitz and J.L. Remy, editors. *Proc 3rd Int'l Workshop on Conditional Term Rewriting Systems CTRS-92*, volume 656 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1992.