A Generalized Semantics for Constraint Logic Programs *

Roberto Giacobazzi[1], Saumya K. Debray[2], Giorgio Levi[1]

1) Dipartimento di Informatica
Università di Pisa
Corso Italia 40, 56125 Pisa
{giaco,levi}@di.unipi.it

2) Department of Computer Science
The University of Arizona
Tucson, AZ 85721
debray@cs.arizona.edu

In FGCS'92 proceedings

Abstract

We present a simple and powerful generalized algebraic semantics for constraint logic programs that is parameterized with respect to the underlying constraint system. "Generalized semantics" abstract away from standard semantics objects, by focusing on the general properties of any (possibly non-standard) semantics definition. In constraint logic programming, this corresponds to a suitable definition of the constraint system supporting the semantics definition. An algebraic structure is introduced to formalize the constraint system notion, thus making applicable classical mathematical results and both a top-down and bottom-up semantics are considered. Non-standard semantics for CLP can then be formally specified by means of the same techniques used to define standard semantics. Different non-standard semantics for constraint logic languages can be specified in this framework: e.g. abstract interpretation, machine level traces and any computation based on an instance of the constraint system.

1 Introduction

Constraint logic programming (CLP) is a generalization of the pure logic programming paradigm, having similar model-theoretic, fixpoint and operational semantics [Jaffar and Lassez 87]. Since the basic operational step in program execution is a test for solvability of constraints in a given algebraic structure, CLP has in addition an algebraic semantics. CLP is then a general paradigm which may be instantiated on various semantic domains, thus achieving a good expressive power. One relevant feature is the distinction between testing for solvability and computing a solution of a given constraint formula. In the logic programming case, this corresponds to the unification process, which tests for solvability by computing a solution (a set of equations in solved form or *most general unifier*). In CLP, the computation of a solution of a constraint is left to a constraint solver, which does not affect the semantic definition of the language. This allows different computational domains, e.g. real arithmetic, to be considered without requiring complicated encodings of data objects as first order terms. Since the fundamental linguistic aspects of CLP can be separated from the details specific to particular constraint systems, it seems natural to parameterize the semantics of CLP languages with respect to the underlying constraint system [Saraswat *et al.* 91]. We refer to such a semantics as *generalized semantics*. It turns out that generalized semantics provide a powerful tool for dealing with a variety of applications relating to the semantics of CLP programs. For example, by considering a domain of "abstract constraints" instead of the "concrete constraints" that are actually manipulated during program execution, we obtain for free a formal treatment of abstract interpretation of CLP programs: this provides a foundation for dataflow analysis and program manipulation of CLP programs. In this paper we address the problem of defining a generalized semantics for constraint logic programs. This can also be the base to specify non-standard semantics for other logic-based languages (e.g. in [Barbuti *et al.* 92] Prolog control features are expressed in terms of a constraint logic language). The algebraic approach we take to constraint interpretation makes it easy to identify a suitable set of operators, which can be instantiated in different ways to obtain the definition of different non-standard semantics. An interesting aspect of such a development is that non-standard interpretations such as abstract interpretations can be

*The work of R. Giacobazzi and G. Levi was supported by "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo" of C.N.R. under grant n. 9100880.PF69 and by the Esprit Basic Research Action 3012 - Compulog. The work of S. Debray was supported in part by the National Science Foundation under grant number CCR-8901283.

developed entirely within an algebraic framework: for example, the notion of "abstraction" can be characterized simply via additional axioms that specify which terms are to be considered "equal" under the abstract interpretation, and relationships between different abstract interpretations can be characterized in terms of homomorphisms between the corresponding algebras.

In this paper, two kinds of generalized semantics top-down and bottom-up, are considered. Since computations are always performed in the algebra of constraints, the two approaches represent just two ways to perform possibly non-standard computations. The reader is assumed to be acquainted with the basic notions of *lattice theory* and *sorted algebras*. Full proofs, not included due to space limitations, are present in the full version of this paper.

2 Constraint Algebras

As defined in [Jaffar and Lassez 87], the semantics of constraints is given in terms of an algebraic structure which interprets constraint formulas, while the semantics of a constraint logic program is given in terms of the well known fixpoint, model-theoretic and operational characterizations. In this section we introduce an incremental algebraic specification for constraint systems.

2.1 Term Systems

In the following we introduce the notion of *term system* as an algebra of terms provided with a binary operator which realizes substitutions [Cirulis 88]. We are interested in term systems where every term depends only on a finite number of variables[1]. They represent the first basic definition in the semantics construction.

Definition 2.1 A *term system* $\mathcal{T}$ is an algebraic structure $(\mathcal{T}, S, V)$ where we refer to the elements of $\mathcal{T}$ as $\mathcal{T}$-*terms* (*terms* in short); V is a countable set of $\mathcal{T}$-*variables* (*variables*, for short) in $\mathcal{T}$; S_V is a countable set of binary operations on $\mathcal{T}$, indexed by V; and the following conditions hold, for all $x, y \in V$ and $t, t', t'' \in \mathcal{T}$:

T_1. $s_x(t, x) = t$, *(identity)*

T_2. $s_x(t, y) = y$, if $x \neq y$, *(annihilation)*

T_3. $s_x(t, s_x(y, t')) = s_x(y, t')$ if $x \neq y$, *(renaming)*

T_4. $s_x(t', s_y(t'', t)) = s_y(s_x(t', t''), t)$ if $x \neq y$ and y *ind* t' *(independent composition)*

where a $\mathcal{T}$-term t is *independent* on the $\mathcal{T}$-variable x, denoted as "x *ind* t," if $s_x(t', t) = t$ for every $t' \in \mathcal{T}$. We say that a variable v *occurs in* a term t if $\neg(x \text{ } ind \text{ } t)$. ∎

[1] A more general definition that considers sets of arbitrary cardinalities is given in [Cirulis 88]: for our purposes, it suffices to consider denumerable sets.

Intuitively, $s_x(t, t')$ denotes the operation "substitute t for every occurrence of the variable x in t'." For notational convenience, we denote $s_x(t, t')$ as $[t/_x]t'$. This notation can be extended to substitutions on multiple variables, by writing $s_{x_1}(t_1, s_{x_2}(t_2, \cdots s_{x_k}(t_k, t) \cdots))$ as $[t_1/_{x_1} \ \dots \ t_k/_{x_k}]t$.

Example 2.1 Let Σ be a denumerable collection of function symbols. We denote by $\tau(\Sigma, V)$ the set of possibly non-ground terms defined on Σ. The standard term system $\mathcal{T}_{(\Sigma,V)} = (\tau(\Sigma, V), \mathsf{Sub}, V)$ is a term system provided that substitutions in Sub perform idempotent substitutions. In this case v *ind* t iff the variable v does not occur in t. □

Let Π be a finite collection of predicate symbols. A $(\mathcal{T}, \Pi)$-atom has the form $p(t_1, ..., t_n)$ where $p \in \Pi$ and $t_i \in \mathcal{T}$, $\forall i = 1, ..., n$. We denote by $W_1 \backslash W_2$ the set W_1 where the elements in W_2 have been removed. The powerset of a set S is denoted by 2^S, and any tuple of syntactic objects (terms, atoms, *etc.*) $o_1, ..., o_n$ is denoted by $\langle o_1, ..., o_n \rangle$.

2.2 An Algebraic Framework

We give now a formal algebraic specification for the language of constraints on a given term system.

Definition 2.2 A *Closed Semiring* [Aho *et al.* 74] is an algebraic structure $(\mathcal{C}, \otimes, \oplus, 1, 0)$, such that: (1) $(\mathcal{C}, \oplus, 0)$ is a (join-)idempotent and commutative monoid; (2) $(\mathcal{C}, \otimes, 1)$ is a (meet-)monoid; (3) 0 is an *annihilator* for $\otimes$; (4) if $a_1, ..., a_n, ..$ is a countable sequence of elements in $\mathcal{C}$, $a_1 \oplus a_2 \oplus ... \oplus a_n \oplus ..$ exists and is unique; (5) associativity, commutativity and idempotence of $\oplus$ apply to infinite as well as finite joins; and (6) $\otimes$ *distributes* over finite and counably infinite joins. ∎

Example 2.2
[[Aho *et al.* 74]] Let $\mathcal{A}_{\Re} = (\Re^+, +, min, 0, +\infty)$ where $\Re^+$ is the set of non-negative reals including $+\infty$, and $\mathcal{A}_\Sigma = (\Sigma^*, \cdot, \cup, \{\epsilon\}, \emptyset)$ where Σ^* is the family of sets of finite-length strings of symbols from the finite alphabet Σ (including the empty string ϵ) and $\cdot$ denotes concatenation. Both $\mathcal{A}_{\Re}$ and $\mathcal{A}_\Sigma$ are closed semirings. Notice that in $\mathcal{A}_\Sigma$ $\cdot$ is not commutative. □

Any semantics definition supports the notion of *observable behaviour* for a given program. Modelling answer constraints in constraint logic programming corresponds to consider answer constraints as the observable property for any CLP program. Thus, the notion of solution for a given answer constraint has to be restricted (projected) to the variables of the corresponding query (output variables). Closed semirings are too weak to capture the notion of variable projection. We handle this notion by means of a family of "hiding" operators on the underlying algebra, as in [Saraswat *et al.* 91]. *Cylindric algebras* [Henkin *et al.* 85] provide a suitable framework to enhance our algebraic structures. A cylindric algebra is formed by enhancing a Boolean algebra by means of a family of unary operations called *cylindrifications.* The intuition here is that given a constraint c, the cylindrification operation $\exists_S(c)$ yields the constraint obtained by "projecting out" information about the variables in S from c. They are necessary here because when we solve a goal in a constraint logic program, we are interested only in constraints on the variables that appear in that goal: thus, any additional constraints that may have been imposed on other variables during the course of the computations should be projected away in the representation of the final answer constraint. This is accomplished using cylindrification. Technically, cylindric algebras allow us to make projections on finite sets of variables. However, since our semantic formulation is in terms of infinite unfolding, as discussed later in the paper, it may also be necessary to allow projections on infinite sets. The machinery of cylindric algebras is not quite adequate for this, but the problem can be handled using *polyadic algebras* [Henkin *et al.* 85], which allow possibly countably many cylindrifications.

Diagonal elements [Henkin *et al.* 85] are considered as a way to provide parameter passing [Saraswat *et al.* 91]. In constraint logic programming the equality symbol "=" is assumed in any constraint system to provide term unification. However, cylindric algebras were introduced to provide an algebraic formalization of first-order-logic, actually oriented to first-order-languages without operation symbols; thus ignoring all terms but variables. This framework is not adequate to provide an algebraic semantic framework for constraint logic programs. We extend diagonal elements to deal with generic terms, following the approach in [Cirulis 88]. Diagonal elements represent equations on a given term system $\mathcal{T}$. This approach results in introducing "term-unification" (i.e. equations on terms) as distinguished elements in the algebra.

Definition 2.3 A *cylindric closed semiring* is an algebraic structure $(\mathcal{C}, \otimes, \oplus, 1, 0, \exists_\Delta, d_{t,t'})_{\Delta \subseteq V; t,t' \in \mathcal{T}}$ where $\mathcal{C}$ is a set called the *universe* of semiring, V is a countable set of variables, $\mathcal{T}$ is a term system; $0, 1, d_{t,t'}$ are distinct elements of $\mathcal{C}$, for each $t, t' \in \mathcal{T}$; $\{\exists_\Delta\}_{\Delta \subseteq V}$ are unary operations on $\mathcal{C}$; $\otimes, \oplus$ are binary operations on $\mathcal{C}$; such that the following postulates are satisfied for any $c, c' \in \mathcal{C}$; $\Delta, \Psi \subseteq V$ and $t, t' \in \mathcal{T}$:

S_1. the structure $(\mathcal{C}, \otimes, \oplus, 1, 0)$ is a closed semiring,

C_1. $\exists_\Delta 0 = 0$,

C_2. $c \oplus \exists_\Delta c = \exists_\Delta c$,

C_3. $\exists_\Delta(c \otimes \exists_\Delta c') = \exists_\Delta c \otimes \exists_\Delta c'$,

C_4. $\exists_\Delta \exists_\Psi c = \exists_{(\Delta \cup \Psi)} c$,

C_5. $\exists_\Delta$ distributes over finite and countably infinite joins,

D_1. $d_{t,t} = 1$,

D_2. $d_{t,t'} = \exists_{\{x\}}(d_{t,x} \otimes d_{x,t'})$ where x *ind* t, t',

D_3. $d_{z,[t/x]t'} = \exists_{\{x\}}(d_{z,t'} \otimes d_{x,t})$, where $z \neq x$ and x *ind* t, z *ind* t'.

∎

Notice that Axiom D_3 relates the notion of substitution in the term system $\mathcal{T}$ with diagonal elements of $\mathcal{C}$ (which intuitively correspond to the notion of equality constraints) in the expected way.

The notions of "independence" and "occurrence" of variables extends in the obvious way from terms in $\mathcal{T}$ to constraints in $\mathcal{C}$. Let $\{x_1, ..., x_n\} \subseteq V$, in the following we will denote $\exists_{var(c)/\{x_1,...,x_n\}} c$, i.e. hiding from all the variables in c except $\{x_1, \ldots, x_n\}$, as $\exists(c)_{\{x_1,\ldots,x_n\}}$. We also denote as $d_{\langle t_1,\ldots,t_n\rangle,\langle t'_1,\ldots,t'_n\rangle}$ the element $d_{t_1,t'_1} \otimes ... \otimes d_{t_n,t'_n}$, where $t_1, ..., t_n, t'_1, ..., t'_n \in \mathcal{T}$. Any closed semiring can be extended to a cylindric closed semiring by letting $d_{t,t'} = 1$ for each $t, t' \in \mathcal{T}$ and $\exists_\Delta c = c$ for each $c \in \mathcal{C}$ and $\Delta \subseteq \mu$. Following [Henkin *et al.* 85] we refer to them as *discrete cylindric closed semirings.*

In the general theory of cylindric algebras, the commutative and transitive properties of diagonal elements (i.e. $d_{t,t'} = d_{t',t}$ and $(d_{t,t'} \otimes d_{t',t''}) \oplus d_{t,t''} = d_{t,t''}$) are derived by the axioms. Because of the weakness of cylindric closed semirings, these properties are not derivable from the axioms. However they are not required in proving the semantic results given below. They can be added to provide the theory of equality.

Given a closed semiring, we can induce a partial ordering relation $\sqsubseteq_\oplus$ on $\mathcal{C}$, such that $c_1 \sqsubseteq_\oplus c_2$ iff $c_1 \oplus c_2 = c_2$. As a consequence, $(\mathcal{C}, \sqsubseteq_\oplus)$ is a complete lattice.

2.3 Constraint Systems

In this section we formalize the notion of *constraint system*, based on the above algebraic framework.

Definition 2.4 A *constraint interpretation structure* is any cylindric closed semiring. Given a constraint interpretation structure $\mathcal{A}$ with universe $\mathcal{C}$, an $\mathcal{A}$-*constraint* (*constraint* for short) is any element in $\mathcal{C}$.
∎

Idempotence, associativity and commutativity are the least set of properties [Barbuti *et al.* 91,Debray and Ramakrishnan 91] which allow $\oplus$ to model the set union operation. $\otimes$ corresponds to the constraint conjunction and plays the important role of collecting the information during the computation. Distributivity allows to represent constraints as possibly infinite joins of finite meets (also called *simple constraints*). Closure on (possibly infinite) countable elements in $\mathcal{C}$ allows to denote infinite joins of constraints.

Example 2.3 Let us assume that $\Pi = \Pi_C \cup \Pi_P$ and $\Pi_C \cap \Pi_P = \emptyset$. We refer to $\Lambda_{(\Sigma,\Pi_C)}$ as the *free algebra of formulas* in the sorted vocabulary (S, Σ, Π_C): where S (sort) is a set of symbols, Σ a specified set of operations with a corresponding signature on S and Π_C a set of predicate symbols with a signature on S; enhanced with the disjunction symbol $\vee$, the conjunction symbol $\wedge$, the existential quantifier $\exists$, the identity symbol $=$, the truth and falsehood symbols T and F and closed under countably infinite disjunctions of formulas in $\Lambda_{(\Sigma,\Pi_C)}$. Equations and possibly existentially quantified $(\mathcal{T}_{(\Sigma,V)}, \Pi_C)$-atoms are called *atomic constraints*.

Let us consider the *solution compact* many sorted algebraic structure $\mathcal{R}_{(\Sigma,\Pi_C)}$ [Jaffar and Lassez 87], defined over the many sorted alphabet (S, Σ, Π_C), consisting of: a collection $D\mathcal{R}$ of non-empty sets denoted $\{D\mathcal{R}_s\}_s$, where $s \in S$; an assignment of a function $D\mathcal{R}_{s_1} \times ... \times D\mathcal{R}_{s_n} \rightarrow D\mathcal{R}_s$ to each n-ary function symbol $f \in \Sigma$, where $(s_1, ..., s_n, s)$ is the signature of f; an assignment of a function $D\mathcal{R}_{s_1} \times ... \times D\mathcal{R}_{s_n} \rightarrow \{true, false\}$ to each n-ary predicate symbol $p \in \Pi_C$, where $(s_1, ..., s_n)$ is the signature of p.

Let us consider a constraint c in $\Lambda_{(\Sigma,\Pi_C)}$. $\mathcal{R} \models c$, iff there exists a mapping ϑ (the solution of the constraint) from each distinct free variable x in c into $D\Re_s$, (free variables in a constraint c are denoted $FV(c)$) where s is the sort associated with x, and $c\vartheta$ is $\mathcal{R}$-equivalent to T ($\mathcal{R} \models c\vartheta$).

Let $c_1 = \bigvee_{i \in I_1} c'_i$ and $c_2 = \bigvee_{i \in I_2} c''_i$ denote possibly infinite disjunctions of conjunctions of atomic constraints c'_i and c''_i, where i ranges over I_1 and I_2 being sets of possibly infinite indexes. The equivalence relation $\approx_{\mathcal{R}}$ on $\Lambda_{(\Sigma,\Pi_C)}$ is defined as follows

$$c_1 \approx_{\mathcal{R}} c_2 \text{ iff } \bigcup_{i \in I_1} \{\vartheta \mid \mathcal{R} \models c'_i\vartheta\} = \bigcup_{i \in I_2} \{\vartheta \mid \mathcal{R} \models c''_i\vartheta\}.$$

$\approx_{\mathcal{R}}$ is a congruence relation on the one sorted algebra $(\Lambda_{(\Sigma,\Pi_C)}, \wedge, \vee, T, F, \exists_X, t = t')_{X \subseteq V; t,t' \in \mathcal{T}_{(\Sigma,V)}}$. The standard constraint interpretation structure is then given by the quotient algebra, denoted as $\mathcal{A}_{st} = (\Lambda_{(\Sigma,\Pi_C)}, \wedge, \vee, T, F, \exists_X, t = t')_{X \subseteq V; t,t' \in \mathcal{T}_{(\Sigma,V)}} / \approx_{\mathcal{R}}$. It is trivially a meet-idempotent and commutative cylindric closed semiring. □

Example 2.4 [$CLP(\mathcal{H})$] Let us consider the following signature associated with the usual Herbrand universe definition, $\Sigma = \{a, b, ..., f, g, ...\}$. Atomic constraints are one sorted equations on the term system $\mathcal{T}_{(\Sigma,V)}$. The corresponding Herbrand interpretation structure $\mathcal{A}_{\mathcal{H}}$, is the quotient algebra $(\mathcal{C}_{\mathcal{H}}, \wedge, \vee, T, F, \exists_X, t = t')_{X \subseteq V; t,t' \in \mathcal{T}_{(\Sigma,V)}} / \approx_{EQ}$, modulo $\approx_{EQ}$, where $\mathcal{C}_{\mathcal{H}} = \{ \; t = t' \mid t, t' \in \mathcal{T}_{(\Sigma,V)} \; \}$ and $\approx_{EQ}$ is the equivalence relation induced by the algebraic structure interpreting diagonal elements as unification [Jaffar and Lassez 87]. It is straightforward to prove that this corresponds to the pure logic programming case. □

To relate constraint interpretation structures, we follow the approach to "static semantics correctness" in [Barbuti and Martelli 83]. Correctness of non-standard semantics specifications can be handled in an algebraic way through the notion of morphism. However, the algebraic notion of morphism can be made less restrictive by assuming that the carriers of the involved algebras be partially ordered sets. We introduce a weaker notion of morphism, capturing the approximation possibly induced by abstract interpretations or any approximate semantics defined in the framework.

Definition 2.5 Let A_Σ and B_Σ be (many sorted) algebraic structures over the sorted alphabet (S, Σ). Let us assume that for each $s \in S$, $(DB_s, \preceq_{DB_s})$ is a partially ordered set. A *weak morphism* $\sigma : A \rightarrow B$ is a family of functions $\sigma_s : DA_s \rightarrow DB_s$, for $s \in S$, such that: $\sigma_s(f_A) \preceq_{DB_s} f_B$, for each constant symbol $f :\rightarrow s$ in Σ and $\sigma_s(f_A(a_1, ..., a_n)) \preceq_{DB_s} f_B(\sigma_{s_1}(a_1), ..., \sigma_{s_n}(a_n))$, for each operation symbol $f : s_1 ... s_n \rightarrow s$ in Σ. ∎

Definition 2.6 Let $\mathcal{A}$ be a constraint interpretation structure. A *constraint interpretation morphism* is a weak morphism ε from $(\Lambda_{(\Sigma,\Pi_C)}, \wedge, \vee, T, F, \exists_X, t = t')_{X \subseteq V; t,t' \in \mathcal{T}_{(\Sigma,V)}}$ in $\mathcal{A}$. ∎

Example 2.5 The *standard constraint interpretation morphism* ε_{st} is a morphism which associates with any formula in $\Lambda_{(\Sigma,\Pi_C)}$, the corresponding equivalence class modulo $\approx_{\mathcal{R}}$. □

In general, a constraint system is an interpretation (in a closed semiring) for constraint formulas.

Definition 2.7 A *constraint system* is a pair $\Gamma = \langle \mathcal{A}, \varepsilon \rangle$ where $\mathcal{A}$ is a constraint interpretation structure and ε is a corresponding constraint interpretation morphism. ∎

Similar algebraic structures for the definition of constraint systems have been introduced in

[Saraswat *et al.* 91] to specify the semantics of the more complex class of *concurrent constraint* languages characterized by the *ask/tell* paradigm.

Constraint systems are specified as *systems of partial information* in the style of Scott's information systems [Scott 82], (*simple constraint systems*), which are tuples $(\mathcal{C}, \Delta, \vdash)$, where $\mathcal{C}$ is an non-empty set of "primitive" constraints and $\vdash \subseteq 2^{\mathcal{C}} \times \mathcal{C}$ is an *entailment relation* such that $\forall u, v \in 2^{\mathcal{C}}$: (1) $u \vdash \Delta$, (2) $u \vdash X$ whenever $X \in u$ and (3) if $v \vdash Y$ for all $Y \in u$ and $u \vdash X$, then $v \vdash X$. The relation $\vdash$ can be extended on $2^{\mathcal{C}} \times 2^{\mathcal{C}}$ as follows: $\forall u, v \in 2^{\mathcal{C}}$, $u \vdash v$ iff $u \vdash X$ for every $X \in v$.

Composition of constraints is defined in terms of set-union, which is a well known commutative and idempotent operator. Hiding and parameter passing are handled by cylindrification (only finite-variable cylindrifications are allowed) and diagonal elements. The difference is then in the underlying algebraic structure: while information systems provide an elegant framework to develop the (standard) semantics for concurrent constraint languages, we are interested in more appropriate algebraic structures to generalize standard semantics results on CLP. In our case, the constraint system is parametric with respect to a given term system. This introduces a more structured approach (two steps) to non-standard constraint system definition (e.g. abstract interpretation). As for the basic algebraic structure, the choice of *closed semirings* results more natural in the context of the present paper. We are interested in possibly non-commutative/idempotent compositions (meets) of constraints (see $\mathcal{A}_\Sigma$ in Example 2.2). Moreover (see *Prop* in Section 4.1 below) standard logical and arithmetic operators (e.g. $\vee$, $\wedge$, T and F) can be specified more naturally as an instance of a closed semiring instead of as an instance of an information system. Nevertheless, it is easy to associate an information system with any $\otimes$-commutative and idempotent closed semiring. Let $(\mathcal{C}, \otimes, \oplus, 1, 0)$ be a $\otimes$-commutative and idempotent closed semiring. The corresponding information system $(\mathcal{C}, \Delta, \vdash)$ is defined as follows $\Delta = 0$ and $\forall u, v \in 2^{\mathcal{C}}$: $u \vdash v$ iff $v \sqsubseteq_\oplus u$.

The key difference is in the semantics definition. In [Saraswat *et al.* 91] the semantics of constraint languages is specified as closures on the constraint system, thus amalgamating the semantics construction and data-objects. We follow the standard approach (see section below) in generalizing the standard operational and fixpoint semantics characterizations, already known in logic programming. A more structured approach to the generalization process can be obtained by separating the domain of constraints with the various techniques to construct models (e.g. fixpoints of continuous transformations) for constraint logic programs. The independence of the semantics constructions from the underying constraint systems focuses the generalization process on the constraint system definition, thus simplifying the specification of non-standard semantics.

Generalized constraint logic programs are defined in the usual way. Let $\mathcal{A}$ be a constraint interpretation structure on the term system $\mathcal{T}$. An $\mathcal{A}$-*clause* is a formula of the form $H \;:-\; c\Box B_1, ..., B_n$ with $n \geq 0$ where H (the *head*) and $B_1, ..., B_n$ (the *body*) are $(\mathcal{T}, \Pi_P)$-atoms, c is an $\mathcal{A}$-constraint and $:-$ and "," denote logic implication and conjunction respectively. An $\mathcal{A}$-*goal* is a formula $c\Box B_1, ..., B_n$, where c is constraint and each B_i is $(\mathcal{T}, \Pi_P)$-atom. A *(generalized) constraint logic program*, also called $\mathcal{A}$-program is a finite set of $\mathcal{A}$-clauses.

3 Generalized Semantics

The mechanism introduced in [Falaschi *et al.* 89] to model computed answer substitutions is generalized in CLP, by allowing constrained atoms into the base of interpretations [Gabbrielli and Levi 91]. Each constrained atom $p(\bar{x}) \;:-\; c$, in fact, represents the set of instances $p(\bar{x})\vartheta$, where ϑ is a solution of the constraint c.

Definition 3.1 Let $\mathcal{A}$ be an interpretation structure. A *constrained atom* has the form $p(\bar{x}) \;:-\; c$ where c is an $\mathcal{A}$-constraint, $p(\bar{x})$ is a $(\mathcal{T}, \Pi_P)$-atom and $FV(c) = \bar{x}$. ▮

Definition 3.2 Let $\mathcal{A}$ be an interpretation structure and $\Im$ be the corresponding set of constrained atoms. We define a partial order $\preceq$ on $\Im$ such that $p(\bar{x}_1) \;:-\; c_1 \preceq p(\bar{x}_2) \;:-\; c_2$ iff there exists $\bar{x}'$ such that $\exists_{\{\bar{x}_1\}}(d_{\bar{x}',\bar{x}_1} \otimes c_1) \sqsubseteq_\oplus c_2$. ▮

The equivalence relation induced by the partial order $\preceq$ is denoted by $\sim$. The $\mathcal{A}$-*base of interpretations* $\mathcal{B}$, is $\Im/_\sim$.

Definition 3.3 $\Xi \subseteq 2^{\mathcal{B}}$ is the collection of sets of constrained atoms I such that $I \in \Xi$ iff $I \uplus \emptyset = \emptyset \uplus I = I$, where $\uplus$ is defined as:

$$\lambda I_1, I_2. \left\{ p(\bar{x}) \;:-\; \textstyle\sum_j \tilde{c}_j \;\middle|\; \begin{array}{l} p(\bar{x}_j) \;:-\; c_j \in I_1 \cup I_2, \\ \tilde{c}_j = \exists_{\bar{x}_j}(d_{\bar{x},\bar{x}_j} \otimes c_j) \\ \text{and } \bar{x} \; ind \; c_j \end{array} \right\}.$$

▮

An $\mathcal{A}$-*interpretation* is any element of Ξ. $\uplus$ is strongly related to $\oplus$. As usual we define $I_1 \sqsubseteq_\uplus I_2$ iff $I_1 \uplus I_2 = I_2$ such that $(\Xi, \sqsubseteq_\uplus)$ is a complete lattice. Each interpretation always consists of a finite set of constrained atoms, containing at most one constrained atom for each program predicate symbol: $p(\bar{x}) \;:-\; \sum_{j \in W} c_j \in I$, where for each $j \in W$, c_j

represents the set of admissible (i.e. computable in the program) solutions for the predicate symbol p, on the variables $\bar{x}$. As a consequence infinite joins of constraints are allowed in constrained atoms. This is well defined by the closure of $\mathcal{C}$. In the following we will often omit $\mathcal{A}$ in specifying programs, goals, *etc.*

3.1 Operational Semantics

Let $\Gamma = \langle \mathcal{A}, \varepsilon \rangle$ be a constraint system and P be an $\mathcal{A}$-program. Define $\rightsquigarrow_P \subseteq \mathcal{A}\text{-}Goals \times \mathcal{A}\text{-}Goals$ (an $\mathcal{A}$-*derivation step*) to be the smallest relation such that $G^{\mathcal{A}} \rightsquigarrow_P G^{'\mathcal{A}}$ iff $G^{\mathcal{A}} = c_0 \Box p_1(\bar{t}_1), ..., p_n(\bar{t}_n)$; there exist n (renamed apart) versions of clauses in P: $p_i(\bar{x}_i) \ :- \ c_i \Box \bar{G}_i, \ i = 1..n$; $G^{'\mathcal{A}} = c_0 \otimes \tilde{c}_1 \otimes ... \otimes \tilde{c}_n \Box \bar{G}_1, ..., \bar{G}_n$, where for each $i = 1..n$, $\tilde{c}_i = d_{\bar{x}_i, \bar{t}_i} \otimes c_i$.

An $\mathcal{A}$-derivation from an $\mathcal{A}$-goal $G^{\mathcal{A}}$ is a finite or infinite sequence of (different) $\mathcal{A}$-goals such that every $\mathcal{A}$-goal is obtained from the previous one by means of a single $\mathcal{A}$-derivation step. A successful derivation is a finite sequence whose last element has an empty body. The operational semantics is then defined in terms of the successful computations specified by the transitive closure of the transition relation on $\mathcal{A}$-goals:

$$\mathcal{O}^{\Gamma}(P) = \left\{ \ p(\bar{x}) \ :- \ \sum \exists (c)_{\bar{x}} \ \middle| \ 1 \Box p(\bar{x}) \rightsquigarrow_P^* c \Box \ \right\}.$$

Goal dependent semantics is defined in terms of a function $\mathcal{G}$ that yields the computed answer constraint for any $\mathcal{A}$-goal, such that

$$\mathcal{G}(G^{\mathcal{A}}) = \exists (c)_{var(G^{\mathcal{A}})} \text{ iff } G^{\mathcal{A}} \rightsquigarrow_P^* c \Box \ .$$

Theorem 3.1 *Let $G^{\mathcal{A}} = c_0 \Box p_1(\bar{t}_1), ..., p_n(\bar{t}_n)$ be a goal. $\mathcal{G}(G^{\mathcal{A}}) = c$ iff there exist $p_i(\bar{x}_i) \ :- \ c_i \in \mathcal{O}^{\Gamma}(P)$, for $i = 1, ..., n$ and $c = \exists (c_0 \otimes d_{\bar{x}_1, \bar{t}_1} \otimes c_i ... \otimes d_{\bar{x}_n, \bar{t}_n} \otimes c_n)_{var(G^{\mathcal{A}})}$.*

3.2 Fixpoint Semantics

In this section we define a fixpoint semantics which is proved to be equivalent to the operational semantics.

Definition 3.4 Let P be an $\mathcal{A}$-program, the mapping $T_P^{\mathcal{A}} : \Xi \to \Xi$, is defined as follows $T_P^{\mathcal{A}}(I) = \biguplus_{C \in P} T_C^{\mathcal{A}}(I)$ where if $C: \ p(t) \ :- \ c \Box p_1(\bar{t}_1), ..., p_n(\bar{t}_n)$ then

$$T_C^{\mathcal{A}}(I) = \left\{ \ p(\bar{x}) : - \exists (\tilde{c})_{\bar{x}} \ \middle| \ \begin{array}{l} \text{for each } i = 1...n : \\ p_i(\bar{x}_i) : - c_i \in I \\ c_i' = d_{\bar{x}_i, \bar{t}_i} \otimes c_i \\ \tilde{c} = d_{\bar{x}, \bar{t}} \otimes c \otimes c_1' .. \otimes .. c_n' \\ \bar{x} \ ind \ c, c_1', ..., c_n' \end{array} \right\}$$

∎

$T_P^{\mathcal{A}}$ is a continuous function on the complete lattice $(\Xi, \sqsubseteq_{\uplus})$. Let $lfp(f)$ denote the least fixpoint of a function f and $\mathcal{F}^{\Gamma}(P) = lfp(T_P^{\mathcal{A}}) = T_P^{\mathcal{A}} \uparrow \omega$. The following result states the equivalence between the operational and the fixpoint semantics, for any constraint system Γ.

Theorem 3.2 *Let P be a program and Γ a constraint system. Then $\mathcal{F}^{\Gamma}(P) = \mathcal{O}^{\Gamma}(P)$.*

4 Abstract Interpretation of CLP

The definition of an abstract constraint system is performed in two steps: *term abstraction* and *constraint abstraction*. In the first step new syntactic objects are introduced to represent sets of concrete terms. In the second one, constraints on the abstracted term system are abstracted. Since the complete lattice of interpretations is induced by the closed semiring structure, any abstract interpretation will correspond to a suitable definition of a constraint system associated with a particular application.

Definition 4.1 Given a constraint system $\Gamma = \langle \mathcal{A}, \varepsilon \rangle$, a constraint system $\Gamma' = \langle \mathcal{A}', \varepsilon' \rangle$, is correct with respect to Γ iff there exists a weak algebraic morphism α_c ($\alpha_c : \mathcal{A} \to \mathcal{A}'$) which is a monotonic mapping of $(\mathcal{C}, \sqsubseteq_{\oplus})$ into $(\mathcal{C}', \sqsubseteq_{\oplus'})$. ∎

Notice that, since α_c is monotonic, it behaves as an algebraic morphism with respect to the $\oplus$ operator. Termination has been guaranteed by requiring that all chains be finite.

Definition 4.2 A constraint interpretation structure $\mathcal{A}$ is *Noetherian* iff $(\mathcal{C}, \sqsubseteq_{\oplus})$ does not contain any infinite chain. A constraint system $\langle \mathcal{A}, \varepsilon \rangle$ is Noetherian iff $\mathcal{A}$ is Noetherian. ∎

Given a Noetherian constraint system Γ, it is easy to prove that $(\Xi, \sqsubseteq_{\uplus})$ is Noetherian. An *abstract constraint system* is a Noetherian constraint system $\Gamma^{\sharp}$ which is correct with respect to the standard one Γ_{st}. It is straightforward to show that in any abstract constraint system $\langle \mathcal{A}^{\sharp}, \varepsilon^{\sharp} \rangle$, $\varepsilon^{\sharp} = \alpha_c \circ \varepsilon_{st}$. Moreover, by weakness and monotonicity, the composition of two monotonic weak morphisms is still a monotonic weak morphism. Let $\Gamma^{\sharp}$ be a correct abstract constraint system. The mapping $\alpha : \Xi \to \Xi^{\sharp}$ such that $\alpha(I) = \left\{ \ p(x) \ :- \ \alpha_c(c) \ \middle| \ p(x) \ :- \ c \in I \ \right\}$ is continuous. Abstract interpretations for constraint logic programs correspond to the definition of an abstract constraint system together with a program evaluation strategy. The first defines what an abstract computation is, while the second one deals with a specific evaluation strategy to collect abstract information. Top-down abstract interpretations correspond to the abstraction of the operational semantics. Bottom-up evaluations instead allow to compute a finite abstract approximation of the fixpoint semantics associated with a given constraint logic program. Goal-independence is an attractive feature of bottom-up evaluations. Global program analysis, especially useful in type inference, can then be specified as a bottom-up evaluation in a suitable constraint system.

Proposition 4.1 *Given a program P and an abstract constraint system $\Gamma^\sharp = \langle \mathcal{A}^\sharp, \varepsilon^\sharp \rangle$, there exists a finite positive k such that $\mathcal{F}^{\Gamma^\sharp}(P) = T_P^{\mathcal{A}^\sharp} \uparrow k$.*

The correctness of the analysis is reduced to the correctness of the constraint system.

Theorem 4.2 *Let P and $\Gamma^\sharp$ be a program and an abstract constraint system respectively. Then $\alpha(\mathcal{O}^\Gamma(P)) \sqsubseteq_\uplus^\sharp \mathcal{O}^{\Gamma^\sharp}(P)$ and $\alpha(\mathcal{F}^\Gamma(P)) \sqsubseteq_\uplus^\sharp \mathcal{F}^{\Gamma^\sharp}(P)$.*

Example 4.1 The closed semiring $\mathcal{A}_\Re$ developed in Example 2.2 can be used to define a simple complexity analysis tool for constraint logic programs on reals, $CLP(\Re)$ [Jaffar and Lassez 87]. Let $|.|_\tau : \tau(\Sigma, V) \to \mathcal{N}$ be a mapping associating a "weight" with any term, where $\mathcal{N}$ is the set of natural numbers. Let us consider a morphism ε such that for each constraint $c : t_1 < t_2$, $\varepsilon(c) = |t_1|_\tau + |t_2|_\tau$.

The interpretation structure $(\mathcal{N}, +, min, 0, +\infty)$, where cylindrifications are defined as in the discrete case and diagonal elements are natural numbers $d_{t_1,t_2} = |t_1|_\tau + |t_2|_\tau$, is trivially Noetherian. A *lower-bound* complexity analysis can be performed returning a lower bound estimation of the costs in arithmetic computations, as in the following example for a simple integration routine:

$$
\begin{array}{l}
\mathsf{int}(A,B,x) \ :- \ 0 < B - A \leq \epsilon, \\
\qquad x = (B-A) * f(A + (B-A)/2) \Box \ \mathsf{E}(\epsilon). \\
\mathsf{int}(A,B,x) \ :- \ B - A > \epsilon, \\
\qquad M = A + (B-A)/2, \ x = x_1 + x_2 \\
\qquad \Box \ \mathsf{int}(A, M, x_1), \mathsf{int}(M, B, x_2), \mathsf{E}(\epsilon). \\
\mathsf{E}(x) \ :- \ x = 1/n \Box \mathsf{N}(n). \\
\mathsf{N}(n') \ :- \ n' = n + 1 \Box \mathsf{N}(n). \\
\mathsf{N}(n') \ :- \ n' = 1 \Box.
\end{array}
$$

Let, for instance, c_+, c_* and c_f be the costs of addition, multiplication/division and f respectively. Variables and constants have a zero cost. Thus, denoting Γ_τ such constraint system:

$$
\mathcal{F}^{\Gamma_\tau}(P) = \left\{ \begin{array}{ll} int(A,B,x) \ :- & 4c_+ + 3c_* + c_f, \\ E(n') \ :- & c_*, \\ N(n') \ :- & 0 \end{array} \right\}.
$$

□

A space of approximate constraints can be specified by defining an auto-weak morphism ρ which is an *upper closure operator* (i.e. an idempotent, monotonic and extensive operator) on $(\mathcal{C}, \sqsubseteq_\oplus)$. As shown in [Cousot and Cousot 79] the approximation process essentially consists in partitioning the space of constraints so that no distinction is made between equivalent constraints, all approximated by a representant of their equivalence class. The equivalence relation is induced by an upper closure operator ρ: $c_1 \equiv_\rho c_2$ iff $\rho(c_1) = \rho(c_2)$. In [Cousot and Cousot 79] different equivalent methods for specifying abstract domains (i.e. upper closure operators) are presented. However, there are standard techniques in algebraic specifications that allow the definition of abstract constraint systems. For example, cylindrifications can be interpreted as abstractions on the algebra of constraints.

Proposition 4.3 *Let $\Delta \subseteq V$; $\exists_\Delta$ is an auto-weak morphism and upper closure operator on $(\mathcal{C}, \sqsubseteq_\oplus)$.*

Existential quantification is then a way to define abstract domains. The space of approximate constraints can also be specified by adding axioms to the underlying constraint system $\mathcal{A}$. These additional axioms extend the meaning of the diagonal elements $d_{t,t'}$ of the algebra, in effect specifying which objects are to be considered "equivalent" from the perspective of the analysis. This is illustrated by the following example:

Example 4.2 Consider the logic program P

$$
\begin{array}{l}
\mathsf{p}(0). \\
\mathsf{p}(s(x)) \ :- \ \mathsf{q}(x). \\
\mathsf{q}(s(x)) \ :- \ \mathsf{p}(x).
\end{array}
$$

and a simple type (parity) analysis for P. Interpreting P as a constraint logic program on the Herbrand constraint system $\mathcal{A}_\mathcal{H}$, the type analysis can be specified by extending the axioms specifying the constraint system with the additional axiom: $s(s(x)) = x$. The resulting constraint system, denoted by $\mathcal{A}_\mathcal{H}^{ex}$, is trivially Noetherian. The semantics of P in $\mathcal{A}_\mathcal{H}$ is $\{p(x) \ :- \ x = 0 \ \bigvee_{n \geq 1} \ x = s^{2n}(0); \ q(x) \ :- \ \bigvee_{n \geq 1} \ x = s^{2n-1}(0)\}$; whereas the interpretation in $\mathcal{A}_\mathcal{H}^{ex}$ returns $\{p(x) \ :- \ x = 0; \ q(x) \ :- \ x = s(0)\}$. The meaning of P in $\mathcal{A}_\mathcal{H}^{ex}$ captures the type of the predicate p and q, computing even and odd numbers respectively. □

A very useful analysis on the relationships among variables of a program can be specified in our framework [Cousot and Halbwachs 78]. The automatic derivation technique in [Verschaetse and De Schreye 91] for linear size relations among variables in logic programs can be suitably specified as a constraint computation. A constraint system of *affine relationships* (i.e. linear equalities of the form $c_0 =$

$c_1X_1 + ... + c_nX_n$) can be defined by specifying intersection, disjunction and cylindrification (restriction) as given in [Verschaetse and De Schreye 91]. Generalizations considering linear inequalities, as proposed in [Cousot and Halbwachs 78], can still be defined in our framework, thus making explicit the strong connection between automatic detection of linear relationships among variables and $CLP(\Re)$ computations. Applications of this analysis are: compile time overflow, mutual exclusion, constraint propagation, termination *etc.* [Jørgensen *et al.* 91].

4.1 Generalized Rigidity Analysis

There exists a wide class of abstract interpretation techniques for the analysis of *ground dependences* (also named *covering*) of pure logic programs [Barbuti *et al.* 91, Cortesi *et al.* 91]. In this section we extend the ground dependence notion by means of the notion of *rigidity.*

A norm is a function weighting terms. Let us recall some basic concepts about norms. For a more accurate treatment on this subject see [Bossi *et al.* 90].

Definition 4.3 Let $\mathcal{T}$ ba a term system. A norm on $\mathcal{T}$ is a function $|..|_\zeta : \mathcal{T} \rightarrow \mathcal{N}$, mapping any term $t \in \mathcal{T}$ into a natural number. ▮

Example 4.3 The following weighting map is a norm on the Herbrand term system: $|t|_{size} = 0$ if t is a variable or $t = []$, $|t|_{size} = 1 + |tail|_{size}$ if $t = [h|tail]$. □

In order to extend the notion of groundness and ground dependences [Barbuti *et al.* 91, Cortesi *et al.* 91] to deal with a more refined one, able to take into account only the relevant subterms of a given (possibly non-ground) term t, we address the notion of rigidity as introduced in [Bossi *et al.* 90].

Definition 4.4 Let $|..|_\zeta$ be a norm on the term system $\mathcal{T}$. A term $t \in \mathcal{T}$ is rigid with respect to $|..|_\zeta$ iff for any substitution of variables σ: $|\sigma t|_\zeta = |t|_\zeta$. ▮

The rigidity of terms turns out to be important in simplifying termination proofs. If a term is rigid, its weight will not be modified by further substitutions. Rigidity is then strongly related to groundness. Any ground term can not change its weight by instantiation, thus it is always rigid. This notion allows to identify those subterms which are relevant for the analysis purposes. Notice that given a norm $|..|_\zeta$, and a non-rigid term $t \in \mathcal{T}$, there must exist some variable in t whose instantiation affects the weight of t. In the Herbrand case, results in [Bossi *et al.* 90] allow to restrict our attention to a particular class of norms: *semilinear norms* on Herbrand.

Definition 4.5 A norm on $\mathcal{T}_{(\Sigma,V)}$ is *semilinear* iff it may be defined according to the following structure: $|t|_\zeta = 0$ if t is a variable; $|t|_\zeta = c_0 + |t_{i_1}|_\zeta + ... + |t_{i_m}|_\zeta$ if $t = f(t_1, ..., t_n)$, where $c_0 \geq 0$ and $\{i_1, ..., i_m\} \subseteq \{1, ..., n\}$. ▮

Note that the position of the subterms which allow the principal term to change its weight by instantiation depends on the outermost term constructor only (i.e. f). These subterms are then relevant from the analysis viewpoint. All the non-relevant subterms are discarded by the analysis. Semilinear norms allow to reduce the rigidity notion to a syntactical property of terms. Let

$$Vrel_\zeta(t) = \{\ v \in V \mid \exists\ \sigma\ such\ that\ |\sigma t|_\zeta \neq |t|_\zeta\ \}.$$

As shown in [Bossi *et al.* 90], given a semilinear norm $|..|_\zeta$, a term $t \in \mathcal{T}_{(\Sigma,V)}$ is rigid iff $Vrel_\zeta(t) = \emptyset$. The notion of semilinear norms can be generalized to arbitrary term systems in a straightforward way, as follows: given a term system $\mathcal{T}$, we define a function $w : \mathcal{T} \longrightarrow \mathcal{N}$; for each $t \in \mathcal{T}$, an associated finite set of functions $F_t : t \longrightarrow \mathcal{T}$; and an associative and commutative function $\bowtie: \mathcal{N} \times \mathcal{N} \longrightarrow \mathcal{N}$.

Intuitively, for any term t, the value of $w(t)$ is the "initial weight" of the term t, the set of functions F_t correspond to the set of selectors for the "relevant" subterms, and $\bowtie$ indicates how the sizes of the subterms of a term are to be combined. Then, *generalized semilinear norms* can be defined as follows:

$$|t| = w(t) + \bowtie_{f \in F_t} |f(t)|.$$

Example 4.4 The "usual" notion of semilinear norms for Herbrand constraint systems can now be generalized as follows, let $c_0 \in \mathcal{N}$: $w(t) = 0$ if t is a variable, c_0 otherwise; if t is a variable then $F_t = \emptyset$; otherwise F_t consists of selectors for the relevant positions of t; $\bowtie$ is summation.

The "depth norm", which could not be expressed as a semilinear norm in the development of [Bossi *et al.* 90], can be defined as follows: $w(t) = 0$ if t is a variable, 1 otherwise; if t is a variable then $F_t = \emptyset$; otherwise if $t = f(t_1, \ldots, t_n)$ then $F_t = \{f_i | 1 \leq i \leq n\}$, where $f_i(t) = t_i$, i.e. f_i is the selector for the subterm at the i^{th} position; and $\bowtie$ is max. □

Let us consider the set $C(V)$ of finite conjunctions of variables in V (the empty conjunction is denoted ϵ) and a term abstraction map $\alpha_\mathcal{T} : \mathcal{T} \rightarrow C(V)$ such that, given a semilinear norm $|..|_\zeta$ and $t \in \mathcal{T}$, $\alpha_\mathcal{T}(t) = \{\ x_1 \wedge ... \wedge x_m \mid Vrel_\zeta(t) = \{x_1, ..., x_m\}\ \}$. Let $\mathcal{T}_\zeta$ be the corresponding abstract term system where substitutions are performed as usual. Marriott and Søndergaard have proposed an elegant domain, named *Prop*, further studied in [Cortesi *et al.* 91], to represent ground dependences among arguments in atoms. In [Codognet and Filè 91] an interesting application is introduced. *Prop* is formalized as a constraint system, and both groundness and definiteness analysis are specified by executing programs in CLP(*Bool*). The corresponding constraint system does not allow disjunctions of varables, without fully exploiting the expressive power of *Prop*. The general notion of ground dependence corresponding with any *Prop* formula (including disjunctions) cannot be specified.

Let $\mathcal{A}_\zeta = (Prop_\zeta, \vee, \wedge, T, F, \exists_X, t \leftrightarrow t')_{X \subseteq V; t,t' \in \mathcal{T}_\zeta}$ be the algebra of possibly existentially quantified formulas defined on the term system $\mathcal{T}_\zeta$; including the set of connectives $\vee$, $\wedge$, $\leftrightarrow$. Intuitively, the formula $x \wedge y \wedge z \leftrightarrow w \wedge v$ represents an equation $t = t'$ where $Vrel_\zeta(t) = \{x, y, z\}$ and $Vrel_\zeta(t') = \{w, v\}$; $x \wedge y$ represents a term whose rigidity depends upon variables x and y; while $x \vee y$ represents a set of terms whose rigidity depends upon variables x or y. Local variables are hidden by existential quantification, projecting away non-global variables in the computation [Codognet and Filè 91].

Let *Bool* be a boolean algebraic structure; $c \approx_{Bool} c'$ iff $Bool \models c \leftrightarrow c'$. It is easy to prove that $\mathcal{A}_\zeta / \approx_{Bool}$ is an abstract constraint system.

Example 4.5 Let us consider the semilinear norm "*size*" and the following constraint logic program on the Herbrand constraint system

$$\begin{array}{lll} \mathsf{append}(x_1, x_2, x_3) & :- & x_1 = [\,] \wedge x_2 = x_3. \\ \mathsf{append}(x_1, x_2, x_3) & :- & x_1 = [h|y] \wedge x_3 = [h|z] \\ & & \qquad \Box \mathsf{append}(y, x_2, z). \end{array}$$

The corresponding abstract model is: $\{append(x_1, x_2, x_3) :- x_1 \leftrightarrow \epsilon \wedge x_2 \leftrightarrow x_3\}$, generalizing the standard ground behavior (where $Vrel(t) = var(t)$: and the abstract model is $\{append(x_1, x_2, x_3) :- x_3 \leftrightarrow x_1 \wedge x_2\}$) vs. size-rigidity behavior: "the second argument list-size can change iff the third argument does". □

5 Machine-level Traces

In this section, we consider an example non-standard semantics for constraint logic programs, that of machine-level traces (for a discussion of similar non-standard semantics in a denotational context, see [Stoy 77]). Such a semantics is essential, for example, if we wish to reason formally about the correctness of a compiler (e.g. see [Hanus 88]) or the behavior of a debugger or profiler. In this section, we show how the semantics described in earlier sections may be instantiated to describe such low-level behaviors. Instead of constrained atoms where each atom is associated with a constraint, this semantics will associate each atom with a set of machine states (equivalently, instruction sequences) that may be generated on an execution of that atom.

The code generated by a compiler for a constraint language must necessarily depend on both the constraint system and the target machine under consideration. Suppose that each "primitive" constraint $op(t_1, \ldots, t_n)$ in the language under consideration corresponds to (an instance of) a (virtual) machine instruction $\mathsf{op}(t_1, \ldots, t_n)$.[2] For example, corresponding to a constraint '$X = Y + 5$' in the language under consideration, we might have a virtual machine instruction '$\mathsf{eq}(X, Y + 5)$'. Each such machine instruction defines a transformation on machine states, representing the changes that are performed to the heap, stack, registers, *etc.* of the machine by the execution of that instruction (e.g., see [Hanus 88] for a discussion of the WAM along these lines). In other words, let $\mathcal{S}$ be the set of all possible states of the machine under consideration, then an instruction I denotes a function $I : \mathcal{S} \longrightarrow \mathcal{S} \cup \{\mathsf{fail}\}$, where fail denotes a state where execution has failed.

Given a set S, let S^∞ denote the set of finite and infinite sequences of S. Intuitively, with each execution we want to associate a set of finite and infinite sequences of machine states, that might be generated by an OR-parallel interpreter. Thus, we want the universe of our algebra to be $2^{\mathcal{S}^\infty}$, the set of sets of finite and infinite sequences of machine states. One subtlety, however, is that instructions may "fail" at runtime because some constraints may be unsatisfiable. To model this, it is necessary to handle failure explicitly, since "forward" execution cannot continue on failure. To deal with this, we define the notion of concatenation of sequences of machine states as follows: given any two sequences s_1 and s_2 of states in $\mathcal{S} \cup \{\mathsf{fail}\}$, their concatenation $s_1 \odot s_2$ is given by $s_1 \odot s_2$ = **if** s_1 **contains** fail **then** s_1 **else** $concat(s_1, s_2)$, where $concat(s_1, s_2)$ denotes the "usual" notion of concatenation of finite and countably infinite sequences. Thus, the cylindric closed semiring in this case is $(\mathcal{C}, \otimes, \oplus, 1, 0, \exists_\Delta, d_{t,t'})_{\Delta \subseteq V; t,t' \in \mathcal{T}}$ where: $\mathcal{C} = 2^{(\mathcal{S} \cup \{\mathsf{fail}\})^\infty}$ is the set of finite and infinite sequences of machine states; for any $S_1, S_2 \in \mathcal{C}$, $S_1 \otimes S_2 = \{s_1 \odot s_2 \mid s_1 \in S_1, s_2 \in S_2\}$; $\oplus = \cup$; $1 = \{\varepsilon\}$, where ε is the empty sequence; $0 = \emptyset$; $\exists_\Delta$ corresponds to the function that, given any machine state S, yields the machine state obtained by discarding all information about the variables in Δ; and for any $t, t' \in \mathcal{T}$, $d_{t,t'}$ corresponds to the function that, given any machine state S, yields the machine state

[2] In an actual implementation, each such virtual machine instruction may, of course, "macro-expand" to a sequence of lower-level machine instructions.

resulting from constraining t and t' to be equal, and fail if this is not possible.

A simple variation on this semantics is one where failed execution sequences are discarded silently. To obtain such a semantics, it suffices to redefine the operation $\oplus$ as follows:
$S_1 \oplus S_2 = \{ \ s \mid \ s \in S_1 \cup S_2 \ \wedge \text{fail is not in } s \ \}$.

6 Related Work

A related framework is considered in [Codognet and Filè 91] where an algebraic definition of constraint systems is given. Program analysis based on abstract interpretation techniques are considered, like *groundness analysis* and *definiteness analysis* for CLP programs. Only $\otimes$-composition is considered. The notion of "computation system" is introduced but it is neither formalized as a specific algebraic structure nor extended with the join-operator. In particular, because of the underlying semantics construction, mainly based on a generalization of the top-down SLD semantics, a loop-checker consisting in a "tabled"-interpreter is introduced. The use of tabled interpreters allows to keep separate the notion of abstraction from the finiteness required by any static analysis. As a consequence, static analysis can be performed by "running" the program in the standard CLP interpreter with tabulation. In our framework, no tabulation is considered. This makes the semantics construction more general. Finiteness is a specific property of the constraint system (expressed in terms of $\oplus$-chains), thus allowing to specify non-standard computations as standard CLP computations over an appropriate non-standard constraint system. Both the traditional top-down and bottom-up semantics can then be specified in the standard way thus allowing the definition of goal-independent static analysis as an abstract fixpoint computation, without loop-checking. If the constraint system is not Noetherian, a *widening/narrowing* technique [Cousot and Cousot 91] can be applied in the fixpoint computation to get a finite approximation of the T_P^A fixpoint.

In a related paper, Marriott and Søndergaard consider abstract interpretation of CLP. A meta-language is defined to specify, in a denotational style, the semantics of logic languages. Abstract interpretation is performed by abstracting such a semantics [Marriott and Søndergaard 90]. In this framework, both standard and non-standard semantics are viewed as an instance of the meta language specification.

Acknowledgment

The stimulating discussions with Maurizio Gabbrielli, Michael Maher and Nino Salibra are gratefully acknowledged.

References

[Aho *et al.* 74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison Wesley Publishing Company, 1974.

[Barbuti *et al.* 92] R. Barbuti, M. Codish, R. Giacobazzi, and G. Levi. Modelling Prolog Control. In *Proc. Nineteenth Annual ACM Symp. on Principles of Programming Languages*, pages 95–104, 1992.

[Barbuti *et al.* 91] R. Barbuti, R. Giacobazzi, and G. Levi. A General Framework for Semantics-based Bottom-up Abstract Interpretation of Logic Programs. Technical Report TR 12/91, Dipartimento di Informatica, Università di Pisa, 1991. To appear in *ACM Transactions on Programming Languages and Systems.*

[Barbuti and Martelli 83] R. Barbuti and A. Martelli. A Structured Approach to Semantics Correctness. *"Science of Computer Programming"*, 3:279–311, 1983.

[Bossi *et al.* 90] A. Bossi, N. Cocco, and M. Fabris. Proving Termination of Logic Programs by Exploiting Term Properties. In S. Abramsky and T. Maibaum, editors, *Proc. TAPSOFT'91*, volume 494 of *Lecture Notes in Computer Science*, pages 153–180. Springer-Verlag, Berlin, 1991.

[Cirulis 88] J. Cirulis. An Algebraization of First Order Logic with Terms. *Colloquia Mathematica Societatis Jànos Bolyai*, 54, 1991.

[Codognet and Filè 91] P. Codognet and G. Filè. Computations, Abstractions and Constraints. Technical Report 13, Dipartimento di Matematica Pura e Applicata, Università di Padova, Italy, 1991.

[Cortesi *et al.* 91] A. Cortesi, G. Filè, and W. Winsborough. *Prop* revisited: Propositional Formulas as Abstract Domain for Groundness Analysis. In *Proc. Sixth IEEE Symp. on Logic In Computer Science*, pages 322–327. IEEE Computer Society Press,1991.

[Cousot and Cousot 77] 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.

[Cousot and Halbwachs 78] P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints Among Variables of a Program. In *Proc. Fifth ACM Symp. Principles of Programming Languages*, pages 84–96, 1978.

[Cousot and Cousot 79] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proc. Sixth ACM Symp. Principles of Programming Languages*, pages 269–282, 1979.

[Cousot and Cousot 91] P. Cousot and R. Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. *Preliminary draft*, ICLP'91 Pre-conference workshop, Paris, 1991.

[Debray and Ramakrishnan 91] S. Debray and R. Ramakrishnan. Generalized Horn Clause Programs. Technical report, Dept. of Computer Science, The University of Arizona, 1991.

[Falaschi *et al.* 89] 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.

[Gabbrielli and Levi 91] M. Gabbrielli and G. Levi. Modeling Answer Constraints in Constraint Logic Programs. In K. Furukawa, editor, *Proc. Eighth Int'l Conf. on Logic Programming*, pages 238–252. The MIT Press, Cambridge, Mass., 1991.

[Hanus 88] M. Hanus. Formal Specification of a Prolog Compiler. In P. Deransart, B. Lorho, and J. Maluszyński, editors, *Proc. International Workshop on Programming Languages Implementation and Logic Programming*, volume 348 of *Lecture Notes in Computer Science*, pages 273–282. Springer-Verlag, Berlin, 1988.

[Henkin *et al.* 85] L. Henkin, J.D. Monk, and A. Tarski. *Cylindric Algebras. Part I and II.* North-Holland, Amsterdam, 1971. (Second edition 1985)

[Jaffar and Lassez 87] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proc. Fourteenth Annual ACM Symp. on Principles of Programming Languages*, pages 111–119, 1987.

[Jørgensen *et al.* 91] N. Jørgensen, K. Marriott, and S. Michaylov. Some Global Compile-Time Optimizations for CLP($\Re$). Technical report, Department of Computer Science, Monash University, 1991.

[Kemp and Ringwood 90] R. Kemp and G. Ringwood. An Algebraic Framework for the Abstract Interpretation of Logic Programs. In S. Debray and M. Hermenegildo, editors, *Proc. North American Conf. on Logic Programming'90*, pages 506–520. The MIT Press, Cambridge, Mass., 1990.

[Marriott and Søndergaard 90] K. Marriott and H. Søndergaard. Analysis of Constraint Logic Programs. In S. Debray and M. Hermenegildo, editors, *Proc. North American Conf. on Logic Programming'90*, pages 531–547. The MIT Press, Cambridge, Mass., 1990.

[Saraswat *et al.* 91] V.A. Saraswat, M. Rinard, and P. Panangaden. Semantic foundation of concurrent constraint programming. In *Proc. Eighteenth Annual ACM Symp. on Principles of Programming Languages*, 1991.

[Scott 82] D. Scott. Domains for Denotational Semantics. In *Proc. ICALP*, volume 140 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1982.

[Stoy 77] J.E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, 1977.

[Verschaetse and De Schreye 91] K. Verschaetse and D. De Schreye. Automatic Derivation of Linear Size Relations. Technical report, Dept. of Computer Science, K.U. Leuven, 1991.