GENERALIZED SEMANTICS AND ABSTRACT INTERPRETATION FOR CONSTRAINT LOGIC PROGRAMS *

ROBERTO GIACOBAZZI, SAUMYA K. DEBRAY, AND GIORGIO LEVI

▷ We present a simple and powerful generalized algebraic semantics for constraint logic programs that is parameterized with respect to the underlying constraint system. The idea is to abstract away from standard semantic objects by focusing on the general properties of any—possibly non-standard—semantic definition. In constraint logic programming, this corresponds to a suitable definition of the constraint system supporting the semantic definition. An algebraic structure is introduced to formalize the notion of a constraint system, thus making classical mathematical results applicable. Both top-down and bottom-up semantics are considered. Non-standard semantics for constraint logic programs can then be formally specified using the same techniques used to define standard semantics. Different non-standard semantics for constraint logic languages can be specified in this framework. In particular abstract interpretation of constraint logic programs can be viewed as an instance of the constraint logic programming framework itself. ◁

*The work of R. Giacobazzi has been partly supported by the EEC *Human Capital and Mobility* individual grant: "Semantic Definitions, Abstract Interpretation and Constraint Reasoning", N. ERB4001GT930817 and by the Esprit Basic Research Action 3012 - Compulog I. The work of G. Levi has been partly supported by "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo" of C.N.R. under grants no. 9100880.PF69. The work of S. Debray was supported in part by the National Science Foundation under grants CCR-8901283 and CCR-9123520.

Address correspondence to Roberto Giacobazzi, Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, E-mail: `giaco@di.unipi.it`

THE JOURNAL OF LOGIC PROGRAMMING

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

1. INTRODUCTION

Constraint logic programming is a generalization of the pure logic programming paradigm, having similar model-theoretic, declarative and operational semantics [45]. Since the fundamental linguistic aspects of constraint logic programming can be separated from the details specific to particular constraint systems, it seems natural to parameterize the semantics of constraint logic programming languages with respect to the underlying constraint systems. We refer to such a semantics as a *generalized semantics*. Such 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. In this paper we focus on algebraic properties that characterize (possibly non-standard) semantic constructions in constraint logic programming. In particular we will focus on computed answer constraint semantics and abstraction. Our framework is therefore suitable to analyze successful computations and call patterns (the latter case can be obtained with a *magic*-like transformation as in [15]) of constraint logic programs. The algebraic approach we take to constraint interpretation makes it easy to identify a suitable set of operators that can be instantiated in different ways to obtain both standard and non-standard interpretations, relying on some simple axioms to ensure that desirable semantic properties are satisfied.

This work has two main technical contributions. The first is the definition of a structure for constraint interpretation that is weak enough to have general applicability, thus dealing with a variety of non-standard interpretations, and at the same time strong enough to ensure that relevant properties of the standard semantic construction, such as the existence of the fixed-point semantics and the equivalence between the top-down and the bottom-up semantics, still hold. The second is to show how a wide class of analysis techniques developed for pure and constraint-based logic programs can themselves be viewed as instances of the constraint logic programming paradigm. Indeed, the approximation of the meaning of programs by means of relations among the variables involved in the computation is a well known technique for specifying a space of approximate assertions for program analysis. We argue that the ability of the constraint logic programming paradigm to handle relations on a variety of semantic domains (e.g., real arithmetics, boolean algebras, *etc.*) allows this paradigm to be used for program analysis, both as a tool for the formal specification of abstract domains, and for the rapid prototyping of static analysis systems. Of course, in order that abstract domains and operators form constraint systems, we require additional (and orthogonal) conditions than simple correctness for semantic operators. Correctness is obviously necessary to get sound approximations in abstract interpretation (most frameworks for static analysis of logic programs require only correctness [5, 11, 15]), so apparently our approach may result a simple restriction of the standard abstract interpretation theory (at the current "state of the art", some analyses like *freeness* are not directly definable as constraint systems). However, lot of interesting semantic properties of analysis (e.g. the equivalence between forward and backward evaluations) are orthogonal to soundness. We prove that all the (abstract) domains and operators that satisfy the constraint system conditions, provide (abstract) interpretations that satisfy the standard results of constraint logic programming semantics as proposed by Jaffar and Lassez [45]. This approach has some interesting practical applications, such as the ability to compile the dataflow analysis directly to an abstract machine for constraint logic programs—a logical extension of the "abstract compilation" scheme discussed by Hermenegildo *et al.* [42]. This removes the overhead of program interpretation incurred by keeping separate abstract and concrete interpretations, and can lead to significant improvements in the speed of analysis (e.g., see [17, 42, 68]). Our approach also makes it possible to close the gap that often exists between the formalization of dataflow analyses in terms of abstract interpretation and the realization of efficient implementations by means of appropriate data-structures and efficient algorithms. Applications of our framework to systematically derive efficient algorithms for dataflow analysis (e.g., by means of constraint propagation techniques for constraint solving) have been recently studied in [4]. Moreover, while non-standard semantics such as those considered in various dataflow analyses, are typically more abstract than the standard semantics, it is also possible, in our framework, to define non-standard semantics that are much more concrete than the standard semantics, i.e., take into account details of a particular implementation. Such semantics, illustrated in Section 4.3, can be used, for example, for reasoning about the correctness of compilers, debuggers, and other low-level tools for program manipulation.

The paper is structured as follows: in Section 2 we introduce the basic mathematical notations used throughout the paper. Section 3 introduces an incremental step-by-step algebraic specification for constraint systems. Section 4 provides both a top-down and a bottom-up semantics for constraint logic programs, parameterized with respect to the constraint system. In Section 5 we consider generalized semantics for constraint logic programs as a framework for semantics-based analyses for constraint logic programs. An example, namely *rigidity analysis*, is considered associating Boolean constraints with standard equations on terms. Some results on approximating constraints by means of *upper closure operators* on constraint systems are also given. This approach points out how some well-known program analysis techniques can be obtained by evaluating an abstract program into a variation of some existing CLP systems, such as $CLP(Bool)$ for rigidity analysis; and, as shown in Section 6, $CLP(\mathcal{R})$, where a weaker notion of "constraint system" for program analysis purposes is introduced by illustrating how a compile-time analysis problem, *linear relationships analysis*, can be formulated in terms of constraint logic programming over an appropriate constraint system. Section 7 contains a survey of the most important related works, and a discussion on limitations of our approach. Section 8 concludes. For continuity and ease of readability, the proofs of most of the results, together with auxiliary lemmata, have been moved to the appendix. This paper is an extended version of [36] and [37].

2. PRELIMINARIES

Throughout the paper we will assume familiarity with the basic notions of lattice theory (Birkhoff's text [8] provides the necessary background) and abstract interpretation (see [22, 24]). In the following we summarize some of the mathematical notation used in the paper.

The set of natural numbers and reals are denoted by $\mathcal{N}$ and $\Re$ respectively. The cardinality of a set A is denoted $|A|$. Given sets A and B, $A \setminus B$ denotes the set A

where the elements in B have been removed. The powerset of a set S is denoted by $\wp(S)$. The class of finite (possibly empty) subsets of a set S is denoted $\wp^f(S)$. Let Σ be a possibly infinite set of symbols. Sequences are typically denoted by $\langle a_1, ..., a_n \rangle$ or simply $a_1, ..., a_n$, where $a_i \in \Sigma$ and $n \geq 0$. The *empty sequence* is denoted by ε. The set (family) of objects a_i indexed on a set of symbols Σ is denoted $\{a_i\}_{i \in \Sigma}$. The set of n-tuples of symbols in Σ is denoted Σ^n. When the length of sequences is fixed, sequences and tuples will be often considered equivalent notions. We occasionally abuse notation and treat sequences as sets. The transitive closure of a binary relation R is denoted by R^*. Syntactic identity is denoted $\equiv$.

A *partial ordering* is a binary relation that is reflexive, transitive and antisymmetric. A set P equipped with a partial order $\leq$ is said to be *partially ordered*, and sometimes written $\langle P, \leq \rangle$. Let $\langle P, \leq \rangle$ be a partially ordered set (poset), $S \subseteq P$ is *convex* iff for each $c, c'' \in S$, $c' \in P$ such that $c \leq c' \leq c''$ then $c' \in S$. A *chain* is a (possibly empty) subset X of a partially ordered set P such that for all $x, x' \in X$: $x \leq x'$ or $x' \leq x$. Given a partially ordered set $\langle P, \leq \rangle$ and $X \subseteq P$, $y \in P$ is an *upper bound* for X iff $x \leq y$ for each $x \in X$. An upper bound y for X is the *least upper bound* iff for every upper bound y': $y \leq y'$; *lower bounds* and *greatest lower bounds* are defined dually. A *complete lattice* is a partially ordered set L such that every subset of L has a least upper bound and a greatest lower bound. A complete lattice L with partial ordering $\leq$, least upper bound $\vee$, greatest lower bound $\wedge$, least element $\bot = \vee\emptyset = \wedge L$, and greatest element $\top = \wedge\emptyset = \vee L$, is denoted $(L, \leq, \bot, \top, \vee, \wedge)$. In the following we will often abuse notation by denoting lattices with their poset notation. We write $f : A \to B$ to mean that f is a total function of A into B. Function composition is denoted $\circ$. Let $f : A \to B$ be a mapping, for each $C \subseteq A$ we denote by $f(C)$ the image of C by f: $\{f(x) \mid x \in C\}$. Functions from a set to the same set are usually called *operators*. The identity operator $\lambda x.x$ is often denoted *id*. Given partially ordered sets $\langle A, \leq_A \rangle$ and $\langle B, \leq_B \rangle$, a function $f : A \to B$ is *monotonic* if for all $x, x' \in A$: $x \leq_A x'$ implies $f(x) \leq_B f(x')$. If A and B are complete lattices, f is *continuous* iff for each non-empty chain $X \subseteq A$: $f(\vee_A X) = \vee_B f(X)$. A function f is *additive* iff the previous condition is satisfied for each non-empty set $X \subseteq A$ (f is also called *complete join-morphism*). An *upper closure operator* on a partially ordered set $\langle A, \leq \rangle$ is a function $\rho : A \to A$ that is idempotent, i.e., $\rho(\rho(c)) = \rho(c)$; extensive, i.e., $c \leq \rho(c)$; and monotonic (more on closure operators can be found in [23]).

Let $(L, \leq, \bot, \top, \vee, \wedge)$ be a non-empty complete lattice. Let $f : L \to L$ be a function. The *ordinal powers* of f are defined as follows for $x \in L$:

$$
\begin{array}{ll}
f{\uparrow}0(x) = x & \\
f{\uparrow}\alpha(x) = f(f{\uparrow}(\alpha-1)(x)) & \text{for every successor ordinal } \alpha\text{; and} \\
f{\uparrow}\alpha(x) = \bigvee_{\delta<\alpha} f{\uparrow}\delta(x) & \text{for every limit ordinal } \alpha.
\end{array}
$$

The first limit ordinal equipotent with the set of natural numbers is denoted by ω. The Knaster-Tarski fixed-point theorem states that the set of fixed-points $fp(f)$ of a monotonic function f over a complete lattice is itself a complete lattice [69]; in particular, this implies that a monotonic function f over a complete lattice has a least fixed-point $lfp(f)$. Moreover, if f is continuous then $lfp(f) = f{\uparrow}\omega(\bot)$.

An *algebraic structure* [41] is a pair $(\mathcal{C}, \mathcal{Q})$ where $\mathcal{C}$ is a non-empty set, called the *universe* of the structure and $\mathcal{Q}$ is a function ranging over an index set $\mathcal{I}$ such that for each $i \in \mathcal{I}$, $\mathcal{Q}_i$ are finitary operations on and to elements of $\mathcal{C}$. Algebraic structures are also denoted as $(\mathcal{C}, \mathcal{Q}_i)_{i \in \mathcal{I}}$. In addition to the operations $\mathcal{Q}_i$, some special symbols (e.g., $\otimes$, $\oplus$, 0,...) will be used to denote algebraic operations, including constants. With an abuse of notation, we will often denote *distinguished elements* of $\mathcal{C}$ as constant operations $\mathcal{Q}_i$ on $\mathcal{C}$. A structure is $(\mathcal{Q}_i)$ α*-complete* for some $i \in \mathcal{I}$ and an infinite cardinal number α, if $\mathcal{Q}_i(X)$ is defined for every set $X \subseteq \mathcal{C}$ such that $|X| \leq \alpha$. It is *complete* if it is α-complete for any α. Given algebraic structures $(A, \mathcal{Q}_A)$ and $(B, \mathcal{Q}_B)$ with universes A and B and provided with a common set of basic operators $\mathcal{Q}$, (we denote $\mathcal{Q}_A$ and $\mathcal{Q}_B$ the operators in $\mathcal{Q}$ defined on A and B respectively) a *(homo)morphism* σ from $(A, \mathcal{Q}_A)$ to $(B, \mathcal{Q}_B)$, denoted by $\sigma : (A, \mathcal{Q}_A) \xrightarrow{m} (B, \mathcal{Q}_B)$ is a function $\sigma : A \rightarrow B$ such that: $\sigma(f_A) = f_B$ for each constant symbol in $\mathcal{Q}$ and $\sigma(f_A(a_1, ..., a_n)) = f_B(\sigma(a_1), ..., \sigma(a_n))$ for each n-ary operation f in $\mathcal{Q}$ and $a_1...a_n \in A$. Let $(A, \mathcal{Q}_A)$ and $(B, \mathcal{Q}_B)$ as above. Given partially ordered sets $\langle A, \leq_A \rangle$ and $\langle B, \leq_B \rangle$, a *semimorphism* is a function $\sigma : A \rightarrow B$ such that $\sigma(f_A) \leq_B f_B$, for each constant symbol f in $\mathcal{Q}$, and $\sigma(f_A(a_1, ..., a_n)) \leq_B f_B(\sigma(a_1), ..., \sigma(a_n))$, for each n-ary operation symbol f in $\mathcal{Q}$.

3. CONSTRAINT ALGEBRAS

As defined by Jaffar and Lassez [45], and Jaffar and Maher [46], the semantics of constraints are given in terms of an algebraic structure that interprets constraint formulae, while the semantics of a constraint logic program is given in terms of fixed-point, model-theoretic and operational characterizations. In this section we introduce an incremental algebraic specification for constraint systems: our interest is in the algebraic properties on which the semantic constructions are based. Constraints are then viewed as elements of an algebraic structure, providing a uniform treatment of semantic domains (collections of constraints) and domain-dependent operators. For this purpose we adapt a widely known algebraic definition of first order logic, namely, *cylindric algebras* [41].

We start from a general notion of *term system* that provides an algebraic treatment for the data objects of a program. *Constraint systems* are then defined as algebraic structures whose universe represents constraints and whose operations include term substitution, constraint composition and projection. The use of such structures in the definition of operational and fixpoint semantics for constraint logic programs is discussed in Section 4. Such a construction has several advantages. First, it provides a uniform algebraic treatment of data objects and domain dependent operators. This is particularly helpful in dataflow analysis by abstract interpretation, as it allows the derivation of the standard properties of (possibly abstract) semantics from few simple axioms (see Section 5). Second, it treats terms explicitly in the algebraic treatment of constraints. This provides a treatment of notions such as *term substitution*, which play a fundamental role in logic programming, that is general enough to be applicable to abstract data descriptions as well as concrete ones. This includes a formal treatment of *variable renaming* on abstract data objects, something that is glossed over in much of the literature on abstract interpretation of logic languages. Moreover, this corresponds precisely to the algebraic generalization of the original CLP framework of Jaffar and Lassez [45], where equality is applied on arbitrary terms to provide (for instance) parameter passing in procedure calls. Finally, it distinguishes between two typical processes in semantic abstraction: *term abstraction* and *constraint abstraction*. The first provides standard data-abstraction (e.g., type information, groundness *etc.*) while the second is actually oriented to interpret relations between abstract data objects as (abstract) constraints.

It is worth noting that cylindric algebras, as formulated by Henkin, Monk and Tarski [41], are actually oriented towards languages without function symbols, thus ignoring all terms but variables. There is a great deal of literature devoted to extending cylindric algebras to deal with terms (see [13] for some references). The idea is that the algebraic definition of a system deals not only with its formulae (the elements of the underlying structure) but also with terms. To motivate this choice, we follow Cirulis [13] and see what arises in logic. Given a first order language with equality (note that equality is always assumed in any constraint system [45]), denote by F and T the sets, respectively, of formulae and terms in the language and by $V \subseteq T$ the set of variables. Let furthermore Θ be a theory in the language, i.e., a set of sentences closed with respect to logical consequence. A Boolean algebra can be obtained by considering $F/\equiv_\Theta$, which can be extended by defining a unary operator $\exists_x : F/\equiv_\Theta \longrightarrow F/\equiv_\Theta$ for $x \in V$, specifying existential quantification as in [41]. To obtain a cylindric algebra, we have to specify *diagonal elements*, i.e., equations of the form $x = y$ for arbitrary variables x and y. If we consider a more extensive set of possible equations including terms, such as $s = t$ for $s, t \in T$, then it is easy to see that the structure $(F/\equiv_\Theta, \exists_x, (s = t))_{x \in V, s, t \in T}$ does not reflect the information Θ contains about equality of terms. In fact Θ gives rise also to an equivalence on T, denoted with abuse of notation $\equiv_\Theta$. Hence a more adequate structure is: $(F/\equiv_\Theta, \exists_x, (s = t))_{x \in V, s, t \in T/\equiv_\Theta}$. Therefore, to axiomatically characterize this extension, we have to take into account the structure of $T/\equiv_\Theta$ in the whole construction of the algebra. Cirulis solves this problem by specifying $T/\equiv_\Theta$ as a *term system* and by making cylindric algebras parametric on it [13]. We follow this construction in our definition of constraint systems.

3.1. 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 ([13]). We are interested in term systems where each term depends only on a finite number of variables (also called *finitary term systems*). They represent the first basic definition in the semantics construction.

Definition 3.1. [term systems [13]]

A *term system* of dimension α is an algebraic structure (τ, S, V) (later abbreviated by τ) where τ is a set of objects called τ-*terms* (*terms* for short); V is a countable set of τ-*variables* (*variables*, for short) in τ; $|V| = \alpha$; S is a countable set of binary operations on τ, indexed by V; and the following conditions are satisfied, for all $x, y \in V$ and $t, t', t'' \in \tau$:

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

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

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

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

independent composition

where a τ-term t is *independent* on the τ-variable x, denoted by "x *ind* t", if $s_x(t', t) = t$ for any $t' \in \tau$. If $X \subseteq V$ then X *ind* t iff x *ind* t for all $x \in X$. We say that a variable v *occurs in* a term t if $\neg(x \ ind \ t)$. We denote the set of variables occurring in a term t as $var(t)$. If $\tau = V$, the term system is said to be *trivial*.

Observe that all trivial term systems with same dimension are isomorphic [13]. In the following we will often omit the specification of the dimension in term systems, when this is obvious from the context.

Intuitively, $s_x(t, t')$ denotes the operation "substitute t for every occurrence of the variable x in t'". It is easy to see that axioms T_1–T_4 are indeed satisfied by the standard notion of substitutions as finite mappings from variables to terms (e.g., [2]). In particular: *renaming* (T_3) specifies that renaming a variable x in a term t' with y ($x \neq y$) makes the resulting term invariant under further substitutions on x; while *independent composition* (T_4) specifies the independency on the order of substitution composition. Notice that in general, the substitution operators do not perform idempotent substitutions. For notational convenience, we often denote $s_x(t, t')$ as $[t/x]\,t'$. This notation can be extended to substitutions on multiple (but finitely many) variables, by writing $s_{x_1}(t_1, s_{x_2}(t_2, \cdots s_{x_k}(t_k, t) \cdots))$ as $[t_1/x_1 \ \ldots \ t_k/x_k]\,t$, where $i \neq j$ implies $x_i \neq x_j$. Notice that, from T_4, if also x *ind* t'' then $[t'/x]\,[t''/y]\,t = [t''/y]\,[t'/x]\,t$. Moreover, by T_2, for each $x, y \in V$: x *ind* y iff $x \neq y$. The condition that terms depend on a finite number of variables can be formalized by requiring that the set $\{x \in V \mid [t/x]\,t' \neq t' \text{ for some } t \in \tau\}$ is finite for every $t' \in \tau$. Our interest in finitary term systems is not related only to their common use in logic programming. Finitary term systems in fact can be induced (built) from any free algebra τ with generators V. This is important in the context of this work, where we need a generalized notion of terms. Define a term system $(\tau, s_x, V)_{x \in V}$ to be *algebraic* if there exists a relatively free algebra τ with generators V (i.e., where each element of τ is generated by a finite subset of V and any mapping $f : V \longrightarrow \tau$ can be extended to an endomorphism of τ) such that $s_x(t, t') = s_x^t t'$ where s_x^t is the endomorphism of τ that takes x into t and agrees with the identity everywhere else. Then, a crucial result on term systems states that a term system is algebraic iff it is finitary [13]. Standard properties of term systems and substitutions, such as the properties of composition, can be found in [13].

Example 3.1. Let Σ be a finite collection of function symbols. $T(\Sigma, V)$ denotes the family of first-order terms defined on Σ and V. The standard term system $\mathcal{T}_{(\Sigma,V)} = (T(\Sigma, V), \mathsf{Sub}, V)$ is a term system provided that substitutions in Sub perform standard substitutions.

Atoms are constructed in the standard way on an arbitrary term system, as specified by the following

Definition 3.2.
Let Π be a finite collection of predicate symbols and τ be a term system. A (τ,Π)-*atom* has the form $p(t_1,...,t_n)$ where $p \in \Pi$ and $t_i \in \tau$, for each $i = 1,...,n$. If t_i are distinct variables, we say that the atom is *flat*.

When clear from the context, we sometimes denote by $\bar{o}$ both a tuple and a set of syntactic objects o (terms, atoms, *etc.*). In particular we denote by $\bar{x}$ a tuple (set) of distinct variables.

The following example shows a non-standard instance of the term system algebraic structure. It provides an adequate term system for the *ground dependency analysis* discussed in Section 5.1.

Example 3.2. Let Σ be a finite set of symbols. Let $\tau_\Sigma = (\wp^f(\Sigma), \mathcal{S}, \Sigma)$, where $\mathcal{S}$ is the family of basic operators s_x, for $x \in \Sigma$, such that for each $X_1, X_2 \in \wp^f(\Sigma)$:

$$s_x(X_1,X_2) = \begin{cases} (X_2 \setminus \{x\}) \cup X_1 & \text{if } x \in X_2 \\ X_2 & \text{otherwise} \end{cases}$$

In this case, for each $x \in \Sigma$ and finite set $X \subseteq \Sigma$: x *ind* X iff $x \notin X$. Then, τ_Σ is a term system. It is straightforward to show that τ_Σ satisfies the axioms of identity (T_1), annihilation (T_2), and renaming (T_3). To show that it satisfies the axiom of independent composition (T_4), assume that $X, X', X'' \subseteq \Sigma$ are finite sets, $x, y \in \Sigma$, $x \neq y$ and $y \notin X'$. If any of X, X' and X'' is empty, the proof is trivial. If $y \notin X$ or $x \notin X \cup X''$, the proof is straightforward. Assume $y \in X$ and $x \in X \cup X''$:

$$\begin{aligned} s_x(X', s_y(X'',X)) &= (((X \setminus \{y\}) \cup X'') \setminus \{x\}) \cup X' \\ &\quad [\text{ definition }] \\ &= (((X \setminus \{x\}) \cup X') \setminus \{y\}) \cup ((X'' \setminus \{x\}) \cup X') \\ &\quad [\text{ distributing } \{x\} \cup X' \text{ and } x \notin X \text{ }] \\ &= s_y(s_x(X',X''), s_x(X',X)) \\ &\quad [\text{ definition }] \end{aligned}$$

3.2. An Algebraic Framework for Constraint Systems

We give now a formal algebraic specification for the language of constraints on a given term system. The process of building constraints in any fixed-point evaluation of a given CLP program is mainly based on set union and conjunction. We want to give an algebraic characterization of this process in order to provide a framework for generalized interpretations of constraint logic programs.

Definition 3.3. [closed semirings [1, 30]]
A *closed semiring*[1] is an algebraic structure $(\mathcal{C}, \otimes, \oplus, \mathbf{1}, \mathbf{0})$ satisfying the following:

[1]They are known as *(join) complete semirings* in the literature (e.g., see [30]). Note that, with respect to [30], in our construction we assume $\oplus$ be idempotent on infinite applications of $\oplus$. We will use the slightly naive name of *closed semirings* adopted from [1] to distinguish them from the more general *complete* case.

R_1. $(\mathcal{C}, \otimes, \mathbf{1})$ and $(\mathcal{C}, \oplus, \mathbf{0})$ are monoids.

R_2. $\oplus$ is commutative and idempotent.

R_3. $\mathbf{0}$ is an *annihilator* for $\otimes$, i.e., for every $c \in \mathcal{C}$, $c \otimes \mathbf{0} = \mathbf{0} \otimes c = \mathbf{0}$.

R_4. for any possibly infinite family $\{a_i\}_{i \in I}$ of elements in $\mathcal{C}$: the sum $a_1 \oplus a_2 \oplus \cdots$, denoted $\sum_{i \in I} a_i$ exists and is unique, i.e., it is a well defined element in $\mathcal{C}$. Moreover associativity, commutativity and idempotence of $\oplus$ apply to infinite as well as to finite applications of $\oplus$.

R_5. $\otimes$ is *left-* and *right-distributive* over finite and infinite applications of $\oplus$, i.e., if $C = \{a_i\}_{i \in I}$ is a possibly infinite family of elements in $\mathcal{C}$ and $c \in \mathcal{C}$, then $c \otimes (\sum C) = \sum(\{c \otimes a_i \mid i \in I\})$ and $(\sum C) \otimes c = \sum(\{a_i \otimes c \mid i \in I\})$, where $\sum C = \sum_{i \in I} a_i$.

Closed semirings provide an algebraic characterization of multiplicity in automata [30]. This phenomenon is evident when multiple paths (or computations) are possible for a given automata. Ioannides and Wong have also shown that the class of relational operators form a closed semiring [43], thus providing a formalization of recursion in the database context. In logic programming, closed semirings summarize, in an algebraic framework, all aspects of dealing with composition of terms, such as unification and set union. The idea is that of finding the (possibly infinite) set of all paths in the semantic construction. From a semantic viewpoint, each path is a sequence of constraints between vertices in the call graph associated with the program. Each successful path constitutes a computation, and will be a sequence (conjunction) of constraints. The multiplicity of paths corresponds to multiple solutions for a query. Idempotence, associativity and commutativity are necessary to allow the operator $\oplus$ (*join*) to model, in a general way, the "merging" together of information via set union. The operator $\otimes$ (*meet*) corresponds to conjunction of constraints and plays the important role of collecting information during computation. Distributivity allows the representation of constraints as possibly infinite joins of finite meets (also called *simple constraints*). Distributivity plays a fundamental role in the equivalence between the bottom-up and the top-down semantics constructions. Closure on infinite sequences of elements in $\mathcal{C}$ is necessary to admit constraints that are infinite joins of constraints (this is important in the semantic development given in Section 4). Closed semirings are thus an appropriate algebraic generalization to model constraint construction as an observable property. Indeed, the asymmetry between joins (disjunctions) and meets (conjunctions) corresponds precisely to the traditional interpretation of observables: infinite disjunctions of observable properties are still observables—to see that $\bigvee_{i \in \mathcal{I}} a_i$ holds of a process we only need to observe that any one of the a_i holds—while infinite conjunctions clearly cannot be observed on the basis of a finite amount of information (e.g., see [66]). A topology for closed (complete) semirings has been recently studied in [51].

Semirings can be *naturally ordered* by defining a binary relation $\leq$ such that for any $a, b \in \mathcal{C}$: $a \leq b$ iff $a \oplus c = b$ for some $c \in \mathcal{C}$ ([30]). In our case, since $\oplus$ is idempotent, there exists a unique natural order for a semiring:

Definition 3.4.
Given a closed semiring $(\mathcal{C}, \otimes, \oplus, \mathbf{1}, \mathbf{0})$, the relation $\trianglelefteq \subseteq \mathcal{C} \times \mathcal{C}$ is defined as follows: for any $c_1, c_2 \in \mathcal{C}$, $c_1 \trianglelefteq c_2$ iff $c_1 \oplus c_2 = c_2$.

Proposition 3.1. Closed semirings are continuous, namely for any infinite family $\{a_i\}_{i \in \mathcal{I}}$ *of elements in* $\mathcal{C}$ *and* $c \in \mathcal{C}$*:*

$$\text{if } \sum_{i \in F} a_i \trianglelefteq c \text{ for all } F \in \wp^f(\mathcal{I}) \text{ then } \sum_{i \in \mathcal{I}} a_i \trianglelefteq c.$$

Karner gives a general treatment of continuous complete semirings [50]. Continuity here corresponds to requiring that $\sum\{a_i \mid i \in I\}$ is the least upper bound of all $\sum\{a_i \mid i \in F\}$ for any finite subset F of I, and is essential for proving the following proposition.

Proposition 3.2.
$\mathcal{C}$ is partially ordered by $\trianglelefteq$, and forms a complete lattice.

A semantic definition necessarily implies some notion of "observable behavior": programs that have the same semantics must not be observationally different. Modelling the semantics of constraint logic programs in terms of answer constraints corresponds to considering answer constraints as the appropriate observable property (this approach to semantics has been considered in [33]), and requires the ability to restrict an answer constraint to the variables appearing in the query. Closed semirings are too weak to capture this restriction operation. We follow Saraswat *et al.* [64] in handling this using a family of "hiding" operators. Cylindric algebras, formed by enhancing Boolean algebras with a family of unary operations called *cylindrifications*, provide a suitable framework for this [41]. The intuition here is that given a constraint c, the cylindrification operation $\exists_S(c)$ yields the constraint obtained by "projecting out" from c all information about the variables in S. 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. To this end, we allow possibly countably many cylindrifications. *Diagonal elements* [41], which represent equations on elements of the underlying term system, are considered as a way to provide parameter passing. However, cylindric algebras, which are oriented towards first-order languages without function symbols, are not adequate as an algebraic semantic framework for general constraint logic programs where parameter passing between procedures is defined by syntactic equality on terms (as in [45]). Therefore, we extend diagonal elements to deal with generic terms, following the approach of Cirulis [13]. This provides an algebraic generalization for syntactic equality on terms as parameter passing applied in [45].

Definition 3.5. [constraint systems]
Given a term system τ of dimension α with variables V, a *τ-based constraint system $\mathcal{A}$* of dimension α is an algebraic structure $(\mathcal{C}, \otimes, \oplus, \mathbf{1}, \mathbf{0}, \exists_X, d_{t,t'})_{X \subseteq V; t,t' \in \tau}$ where $\mathcal{C}$ is a set of $\mathcal{A}$-constraints generated by a given set of *atomic constraints* over terms from τ, and is called the *universe* of $\mathcal{A}$; $\mathbf{0}, \mathbf{1}, d_{t,t'}$ are distinct (atomic) elements of $\mathcal{C}$, for each $t, t' \in \tau$; $\{\exists_X\}_{X \subseteq V}$ is a family of unary operations on $\mathcal{C}$;

$\otimes, \oplus$ are binary operations on $\mathcal{C}$; such that the following postulates are satisfied for any $c, c' \in \mathcal{C}$; $\{x\}, X, Y \subseteq V$ and $t, t', t'' \in \tau$:

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

C_1. $\exists_X \mathbf{0} = \mathbf{0}$

C_2. $c \oplus \exists_X c = \exists_X c$;

C_3. $\exists_X (c \otimes \exists_X c') = \exists_X (\exists_X c \otimes c') = \exists_X c \otimes \exists_X c'$;

C_4. $\exists_X \exists_Y c = \exists_{(X \cup Y)} c$;

C_5. $\exists_X$ distributes over finite and infinite joins;

D_1. $d_{t,t} = \mathbf{1}$;

D_2. $d_{t,t'} = d_{t',t}$;

D_3. $\exists_{\{x\}} (d_{x,t} \otimes d_{t',t''}) = d_{[t/x]t',[t/x]t''}$ for x *ind* t;

D_4. $\exists_{\{x\}} (d_{x,t} \otimes (c \otimes c')) = \exists_{\{x\}} (d_{x,t} \otimes c) \otimes \exists_{\{x\}} (d_{x,t} \otimes c')$.

Where the underlying term system τ for a τ-based constraint system is unimportant or is obvious from the context, we will omit reference to it.

The meaning of cylindrification is given by the axioms from C_1 to C_5, while diagonal elements are specified by the axioms from D_1 to D_4. Notice that Axioms D_3 and D_4 relate the notion of substitution in the term system τ with diagonal elements of $\mathcal{C}$ (which intuitively correspond to the notion of equality constraints) in the expected way. We follow Henkin, Monk and Tarski [41] in considering a family of (derived) operations ∂_x^t, defined on $\mathcal{C}$ for $x \in V$ and $t \in \tau$ such that x *ind* t:

$$\partial_x^t c = \exists_{\{x\}} (d_{x,t} \otimes c).$$

We call these operations *substitutions*, since intuitively they extend the notion of substitution from the underlying term system to the universe of constraints. With an abuse of notation, we denote $\partial_x^t(c)$ as $[t/x]\, c$ when the meaning is clear from the context.

In the following we distinguish between *constraints* and *simple constraints*. A constraint is any object in the universe of a constraint system, while a *simple constraint* is an atomic constraint, or the cylindrification of a simple constraint, or a finite conjunction (i.e., meet) of simple constraints. Therefore, simple constraints do not contain joins. The *compact constraints* of a constraint system $\mathcal{A}$ are the compact elements in $\mathcal{C}$, namely the finite joins of simple constraints. As we will see later in the semantic construction, an answer of a query to a program will be a compact constraint, corresponding to a single finite computation for the query. The join operator is applied to model the non-deterministic clause choice in logic programs, which may provide possibly multiple answers for a query.

The function *var* and the notions of "independence" and "occurrence" of variables extend in the obvious way from terms in τ to constraints in $\mathcal{C}$. Let $c \in \mathcal{C}$ and $x \in V$: x *ind* c iff $\partial_x^t c = c$ for any $t \in \tau$ such that x *ind* t. A variable x is *bound* in c iff it is existentially quantified in c; x is *free* in c iff $x \in var(c)$ and x is not bound in c. The set of *free variables* in a constraint c is denoted by $FV(c)$. A *renaming* of c with respect to x is a constraint $\partial_x^y c$ such that $x \neq y$ and y *ind* c.

Let τ be a term system with variables V and $(\mathcal{C}, \otimes, \oplus, \mathbf{1}, \mathbf{0})$ be a closed semiring. $(\mathcal{C}, \otimes, \oplus, \mathbf{1}, \mathbf{0})$ can be extended to a constraint system by letting $d_{t,t'} = \mathbf{1}$ for each $t, t' \in \tau$ and $\exists_X c = c$ for each $c \in \mathcal{C}$ and $X \subseteq V$ (here $\partial_x^t c = c$ for each $x \in V$ and $t \in \tau$). Following Henkin *et al.* [41], we refer to these as *discrete constraint systems*. Let $X \subseteq V$, in the following we will denote $\exists_{var(c) \setminus X} c$, i.e., hiding all the variables in c except X, as $\exists(c)_X$. We will often omit parentheses in cylindrifications on sets of variables. We also denote by $d_{\langle t_1, \ldots, t_n \rangle, \langle t'_1, \ldots, t'_n \rangle}$ the element $d_{t_1, t'_1} \otimes \ldots \otimes d_{t_n, t'_n}$, where $t_1, \ldots, t_n, t'_1, \ldots, t'_n \in \tau$.

In the following we use $\mathcal{A}$ to denote an arbitrary constraint system.

Theorem 3.1 (elementary properties of constraint systems).
Let $\mathcal{A}$ be an arbitrary constraint system. For any $c, c' \in \mathcal{C}$, $x \in V$, $X \subseteq V$ and $t, t', t'' \in \tau$ such that x ind t, the following properties hold:

$\exists_X \exists_X c = \exists_X c$;
P2: $c \trianglelefteq c' \;\Rightarrow\; \exists_X c \trianglelefteq \exists_X c'$;
P3: $\forall c, c' \in \mathcal{C} : \; c' \trianglelefteq \exists_X c \;\Leftrightarrow\; \exists_X c' \trianglelefteq \exists_X c$;[2]
P4: $\forall c, c' \in \mathcal{C} : \; c \trianglelefteq c' \wedge c' \trianglelefteq \exists_X c \;\Rightarrow\; \exists_X c = \exists_X c'$;
P5: $\exists_{\{x\}} c = c$ *iff* $\exists_{\{x\}} \tilde{c} = c$ *for some* $\tilde{c} \in \mathcal{C}$;
P6: $\exists_{\{x\}} c = c$ *if* x *ind* c *(in particular* $\exists_{\{x\}} d_{t',t''} = d_{t',t''}$ *when* x *ind* t', t''*)*;
P7: $d_{t,t'} = \exists_{\{x\}} (d_{t,x} \otimes d_{x,t'})$ *where* x *ind* t, t',
P8: $c \trianglelefteq c' \;\Rightarrow\; \partial_x^t c \trianglelefteq \partial_x^t c'$;
P9: $\partial_x^t \exists_{\{x\}} c = \exists_{\{x\}} c$;
P10: $\partial_x^t \; c = c$ *iff* $\partial_x^t \tilde{c} = c$ *for some* $\tilde{c} \in \mathcal{C}$;
P11: $\exists_X \mathbf{1} = \mathbf{1}$; $\exists_X c = \mathbf{0}$ *iff* $c = \mathbf{0}$;
P12: $\exists_{\{x\}} d_{x,t} = \mathbf{1}$;
P13: $(d_{t,t'} \otimes d_{t',t''}) \oplus d_{t,t''} = d_{t,t''}$ *(transitivity)*.

In particular, from properties P1, P2, and axioms C_2 and C_5, $\exists_X$ is an additive *upper closure operator* on $\mathcal{C}$ for each $X \subseteq V$. Moreover, by properties P8, P10, and from the distributivity of $\exists$ and $\otimes$ over $\oplus$, the substitution operator on constraints defines an additive *retraction* on $\mathcal{C}$, where a *retraction* on a partially ordered set A is an idempotent and monotonic mapping over A. Notice that substitution is not, in general, extensive. Other elementary properties of constraint systems can be derived from similar properties of cylindric algebras in [41]. Notice that (by P9) $\partial_x^t c \trianglelefteq \exists_{\{x\}} c$ and if x is bound in c then x *ind* c. Therefore, if c is a renaming apart of c' with respect to x, then x *ind* c.

The following lemma describes the interaction of variable projection with (hidden) variables in constraints, thus extending the elementary property P6 to conjunctions of constraints.

Lemma 3.1 (independence).
For any constraint system $\mathcal{A}$, if c and c' are $\mathcal{A}$-constraints and X is a set of variables such that x ind c for every $x \in X$, then $\exists_X (c \otimes c') = c \otimes \exists_X (c')$.

[2] This property corresponds to Morgado's characterization of closure operators by means of a single axiom [61].

The following lemma shows an important relation between cylindrification (hiding variables) and renaming apart of constraints with "fresh" variables.

Lemma 3.2.
For any constraints c and c' in a constraint system $\mathcal{A}$, $c \otimes \exists_{\{x\}} c' = \exists_{\{y\}}(c \otimes \tilde{c}')$, where y ind c, c'; $y \neq x$ and $\tilde{c}' = \partial_x^y c'$.

The following examples show some standard constraint systems.

Example 3.3. $[CLP(\mathcal{H})]$
Let Σ be a finite collection of function symbols. Atomic constraints are equations on the term system $\tau_{(\Sigma,V)}$ (see Example 1). Let $\mathcal{E}_{\mathcal{H}}$ be the set of possibly existentially quantified finite conjunctions of equations over $\tau_{(\Sigma,V)}$, and let $\mathcal{I}_{\mathcal{H}}$ represent the Herbrand interpretation structure, interpreting diagonal elements as syntactic equality [45]. In this case, a solution θ for a possibly quantified finite conjunction (set) of equations $\exists_X E = \exists_X \{s_1 = t_1, ..., s_n = t_n\}$ is a grounding substitution for the free variables in E such that there exists a grounding substitution σ for the bound variables X, and $s_1\sigma\theta \equiv t_1\sigma\theta$, ..., $s_n\sigma\theta \equiv t_n\sigma\theta$. $\mathcal{I}_{\mathcal{H}} \models E\theta$ denotes that θ is a solution for E. We extend this definition to deal with possibly infinite joins: θ is a solution for $\bigcup_{i\in I} E_i$ iff there exists $i \in I$ such that θ is a solution for E_i. $\exists$ is existential quantification, which is assumed to be distributive (as well as conjunction) over arbitrary joins: if $X \subseteq V$, θ is a solution for $\exists_X(\bigcup_{i\in I} E_i)$ iff θ is a solution for $\exists_X E_i$ for some $i \in I$; *true* denotes any constraint having every grounding substitution as a solution while *false* denotes any constraint having an empty set of solutions. Note that ∂_x^t, for x not occurring in t, performs idempotent substitutions on constraints, by extending in the obvious way the term substitution notion to constraints. Moreover, for each $c_1 = \bigcup_{i\in I_1} E_i$ and $c_2 = \bigcup_{i\in I_2} E'_i$ denoting possibly infinite joins of (finite) quantified sets of atomic constraints (equations) E_i and E'_i:

$$c_1 \sim_{EQ} c_2 \ \text{ iff } \ \bigcup_{i\in I_1} \{ \ \vartheta \mid \mathcal{I}_{\mathcal{H}} \models E_i\vartheta \ \} = \bigcup_{i\in I_2} \{ \ \vartheta \mid \mathcal{I}_{\mathcal{H}} \models E'_i\vartheta \ \}.$$

Then, the *Herbrand constraint system* $\mathcal{H}$ is the quotient algebra

$$(\wp(\mathcal{E}_{\mathcal{H}}), \wedge, \cup, true, false, \exists_X, \{t = t'\})_{X \subseteq V; t,t' \in \tau_{(\Sigma,V)}} / \sim_{EQ},$$

Example 3.4. $[CLP(\mathcal{LR}_n)]$
This example describes the case of $CLP(\mathcal{R})$ [45] on linear constraints, where the number of variables is restricted *a priori* to some fixed value n, as an instance of our framework (the case with $n = \omega$ is of little interest in our construction since constraint logic programs can define only finitary predicates). This constraint system will be used for static analysis of $CLP(\mathcal{H})$ programs in Section 6.1. In the following $\vec{x} = (x_1, \ldots, x_n)$ is a point in $\Re^n$ and x_i is its i-th element. A *hyperplane* (atomic constraint) is the set of points $\vec{x} \in \Re^n$ satisfying an equation of the form $a_1x_1 + \cdots + a_nx_n = b$, and defines two *halfspaces* in the obvious way. A *convex polyhedron* is the (possibly unbounded) set of points constituting the intersection of a finite number of halfspaces. For any finite n, the constraint

system of n-dimension linear constraints (the non-linear case is a straightforward extension), denoted by $\mathcal{LR}_n$, is: $(\mathcal{P}, \cap, \cup, \Re^n, \emptyset, \hat{\exists}_X, [t_1 = t_2])_{X \subseteq V_n; t_1, t_2 \in \tau_{Exp}}$, where $V_n = \{x_1, ..., x_n\}$ is a set of n variables, τ_{Exp} is a term system of linear expressions on V_n (an example of definition for τ_{Exp} is in Section 6.1) and $\mathcal{P}$ is the set of all space regions in $\Re^n$ defined as possibly infinite unions of *convex polyhedra*. Each constraint $c \in \mathcal{P}$ can be represented as a possibly infinite set of finite conjunctions of linear equations and disequations on V_n. The variable restriction operation $\hat{\exists}$ is performed by *cylindrification parallel to an axis* [41]: if c is a constraint in $\Re^n$ and $i \leq n$, we define:

$$\hat{\exists}_{x_i} c = \left\{ \ \vec{y} \in \Re^n \ \middle| \ y_j = x_j \ for \ \vec{x} \in c \ and \ j \neq i \ \right\}.$$

$\hat{\exists}_{x_i} c$ is the cylinder generated by moving the point set c parallel to the x_i axis. For any two linear expressions $t, t' \in \tau_{Exp}$ and $R \in \{=, \geq, \leq, >, <\}$ we denote by $[t \ R \ t']$ the corresponding space. It is not difficult to show that the resulting structure is a constraint system (see [34]).

4. GENERALIZED SEMANTICS

Constraint logic programming was defined by Jaffar and Lassez to specify relations on a constraint language by means of constraint-based Horn clauses. We follow this approach by defining Horn-like clauses on constraint systems. Constraint logic programs are defined in the usual way: let $\mathcal{A}$ be a constraint system on a term system τ and Π be a finite set of predicate symbols. An $\mathcal{A}$-*goal* is a formula '$c \,[\!]\, B_1, ..., B_n$,' with $n \geq 0$, where c is a compact $\mathcal{A}$-constraint and $B_1, \ldots, B_n$ is a sequence of (τ, Π)-atoms. An $\mathcal{A}$-*clause* is a formula of the form '$H \ :- \ c \,[\!]\, B_1, ..., B_n$' where H (the *head*) is a (τ, Π)-atom and '$c \,[\!]\, B_1, ..., B_n$' (the *body*) is an $\mathcal{A}$-goal. If the body is empty, the clause is a *unit clause*. Given a set of clauses S, we use $preds(S)$ to denote the set of predicate symbols in the heads of clauses in S. A *(generalized) constraint logic program*, also called $\mathcal{A}$-program, is a finite set of clauses. If the constraint system under consideration is obvious from the context, we will sometimes not indicate it explicitly in the various semantic functions. The family of $\mathcal{A}$-programs is denoted by $CLP(\mathcal{A})$. Finally, the notion of renamings of variables in constraints and terms, as well as the function var and the notion of independence, extend their meaning in the obvious way to syntactic objects such as atoms, goals, clauses, and programs.

4.1. Top-Down Operational Semantics

Let $\mathcal{A}$ be a constraint system and $P \in CLP(\mathcal{A})$. Define $\leadsto_P$ (an $\mathcal{A}$-*derivation step*) to be the least relation on $\mathcal{A}$-goals such that $G \leadsto_P G'$ iff the following hold:

(i) $G = c_0 \,[\!]\, p_1(\bar{t}_1), ..., p_n(\bar{t}_n)$;

(ii) there is a renamed version of a clause in P: $p_1(\bar{t}'_1) \ :- \ c_1 \,[\!]\, \bar{B}_1$, such that $var(G) \cap var(\bar{B}_1 \cup \bar{t}'_1) = \emptyset$;

(iii) $G' = c_0 \otimes d_{\bar{t}_1, \bar{t}'_1} \otimes \exists(c_1)_{var(\bar{B}_1 \cup \bar{t}'_1)} \,[\!]\, \bar{B}_1, p_2(\bar{t}_2), ..., p_n(\bar{t}_n)$.

An $\mathcal{A}$-derivation from an $\mathcal{A}$-goal G is a finite or infinite sequence of $\mathcal{A}$-goals such that every 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 constraint obtained from a successful derivation is the *answer constraint*. Notice that, since projection of the local variables is performed after the whole computation, an accurate definition of the operational semantics requires a denumerable set of variables on which to perform renamings (a different solution can be obtained by extending the scope of cylindrification to clause bodies).

The goal-dependent success set semantics of a program P is defined in terms of a function $\mathcal{J}_P$ that yields the set of computed answer constraints for any $\mathcal{A}$-goal, such that $\mathcal{J}_P(G) = \{\exists(c)_{var(G)} \mid G \leadsto_P^* c \,\|\, \varepsilon\}$. Since the operator $\otimes$ in a constraint system may not be commutative, the independence of the selection rule does not hold in general in these semantic characterizations, and for simplicity we have assumed a left-to-right selection rule. If $\otimes$ is commutative it is straightforward to prove the independence on the selection rule for the success set [54].

The following lemma specifies an important equivalence between syntactically different goals. This result will be useful later in proving the equivalence between top-down and bottom-up semantics. Here, a variable is said to be "used" in a derivation if it occurs in some goal in that derivation.

Lemma 4.1.
$\mathbf{1} \,\|\, p(\bar{t}) \leadsto_P^* c \,\|\, \varepsilon$ *iff* $d_{\bar{x},\bar{t}} \,\|\, p(\bar{x}) \leadsto_P^* c' \,\|\, \varepsilon$ *and* $\exists_{\{\bar{x}\}} c' = c$, *where no variable in* $\bar{x}$ *is used in the derivation* $\mathbf{1} \,\|\, p(\bar{t}) \leadsto_P^* c \,\|\, \varepsilon$.

It is worth noting that a similar argument can be applied to prove that if P is a program and P' is obtained from P by transforming each clause $C =$ '$p(\bar{t}) \;:-\; c \,\|\, \bar{B}$' $\in P$ to '$p(\bar{x}) \;:-\; d_{\bar{x},\bar{t}} \otimes c \,\|\, \bar{B}$' for x *ind* C, then for any goal G it is the case that $\mathcal{J}_P(G) = \mathcal{J}_{P'}(G)$. Both this observation and Lemma 4.1 are consequences of the constraint system structure, extending diagonal elements (i.e., parameter passing) to terms. Because of this observation, in the following we will always write program clauses with flat heads.

Observation 4.1. The explicit treatment of terms in constraint systems also provides a characterization for a number of expected equivalences among syntactically different programs. All of these are consequences of the axioms and therefore are satisfied in any constraint system. For example, it is easy to prove from the axioms that the following two programs have the same goal dependent success set semantics for any goal.

$$\left\{ \begin{array}{l} p(t) \;:-\; q(x) \\ q(t') \;:-\; \mathbf{1} \end{array} \right\} \qquad \left\{ \begin{array}{l} p([t'/x]\,t) \;:-\; \mathbf{1} \\ q(t') \;:-\; \mathbf{1} \end{array} \right\}$$

This is a typical consequence of the equivalence induced by the constraint system structure on formulae including terms, like those obtained from term substitution and parameter passing.

4.2. Success-Set and Bottom-up Fixed-point Semantics

In this section we define a bottom-up fixed-point semantics that is proved to be equivalent to the operational semantics of successful computations for any constraint system. We also study a *condensing* operator which will be useful in abstract interpretation of CLP programs by abstraction of constraints. The approach we take follows that of Falaschi *et al.* [31] and Gabbrielli and Levi [33], and derives a bottom-up fixed-point based semantics from the operational notion of *computed answer constraint* for atomic goal.

Definition 4.1.
Let $\mathcal{A}$ be a constraint system. A *constrained atom* has the form '$A \ :- \ c$' where A is an (τ, Π)-atom, c is an $\mathcal{A}$-constraint, and $FV(c) \subseteq var(A)$. We denote $\mathcal{B}^{\mathcal{A}}$ the set of constrained atoms on a constraint system $\mathcal{A}$.

The (operational) *computed answer constraint semantics* is defined in terms of the the set of successful computations specified by the transitive closure of the derivation relation $\rightsquigarrow$ on atomic $\mathcal{A}$-goals:

$$\mathcal{O}(P) = \left\{ \ p(\bar{x}) \ :- \ \exists(c)_{\bar{x}} \ \middle| \ \mathbf{1} \ [\![\ p(\bar{x}) \rightsquigarrow_P^* c \ [\![\ \varepsilon \ \right\} .$$

This generalizes the computed answer constraint semantics of Gabbrielli and Levi [33] to arbitrary constraint systems. Intuitively, a constrained atom '$p(\bar{x}) \ :- \ c$' in $\mathcal{O}(P)$ represents the set of instances $p(\theta(\bar{x}))$, where θ is a solution to the *answer constraint* c. The following lemma proves the AND-compositionality for the operational semantics of constraint logic programs, providing a characterization of answer constraints for conjunctive goals in terms of the computed answer constraint semantics (an equivalent lemma is proved in [33] for the classical constraint structure of Jaffar and Lassez [45]).

Lemma 4.2.
Let $G = c_0 \ [\![\ p_1(\bar{t}_1), ..., p_n(\bar{t}_n)$ be an $\mathcal{A}$-goal and $P \in CLP(\mathcal{A})$. $\mathcal{J}_P(G) = c$ iff there exist $p_i(\bar{x}_i) \ :- \ c_i \in \mathcal{O}(P)$, such that $\bar{x}_i$ ind G and $\bar{x}_i \cap \bar{x}_j = \emptyset$ for $1 \le i, j \le n$, $i \ne j$; and $c = \exists(c_0 \otimes d_{\bar{x}_1, \bar{t}_1} \otimes c_1 ... \otimes d_{\bar{x}_n, \bar{t}_n} \otimes c_n)_{var(G)}$.

It can be shown that the unfolding of a clause (goal) with constrained atoms is independent from the variable names used in constrained atoms (see Lemma A.1 in Appendix). This can be expressed in the semantics by a relation $\sim$ that captures the notion of equivalence upto renaming on constrained atoms. Define the binary relation $\sim$ on $\mathcal{B}^{\mathcal{A}}$ as follows: given $A_1 \equiv$ '$p(\bar{x}_1) \ :- \ c_1$' and $A_2 \equiv$ '$p(\bar{x}_2) \ :- \ c_2$' in $\mathcal{B}^{\mathcal{A}}$, $A_1 \sim A_2$ if and only if there exist "renaming apart" variables $\bar{x}'$ such that $\bar{x}'$, $\bar{x}_1$, and $\bar{x}_2$ are mutually disjoint; $\bar{x}'$ *ind* c_1, c_2; and $\partial_{\bar{x}_1}^{\bar{x}'} c_1 = \partial_{\bar{x}_2}^{\bar{x}'} c_2$. It is easy to show that $\sim$ is an equivalence relation.

Definition 4.2.
The $\mathcal{A}$-*base of interpretations* is $\mathcal{B}^{\mathcal{A}}/\sim$.

In the remainder of the paper we will be concerned primarily with the quotient structure $\mathcal{B}^{\mathcal{A}}/\sim$, and for notational simplicity, denote this by $\mathcal{B}^{\mathcal{A}}$. Given a syntactic object o, we denote by '$p(\bar{x}) \ :- \ c \ll_o I$' a variant of a constrained atom '$p(\bar{x}) \ :- \ c$' in I that has been renamed apart from o, i.e., such that $[p(\bar{x}) \ :- \ c]_\sim \in I$ and $\bar{x}$ *ind* o. We extend this to specify tuples of syntactic objects that have been renamed apart, so that $\langle A_1, \ldots, A_n \rangle \ll_o I$ represents a tuple $\langle A_1', \ldots, A_n' \rangle$ where each of the A_i' is a variant of an element A_i in I that has been renamed apart from o, and where $i \ne j$ implies A_i and A_j are variable-disjoint.

The fixed-point semantics is defined in terms of an immediate consequence operator on the complete lattice $\langle \wp(\mathcal{B}^{\mathcal{A}}), \subseteq \rangle$, in the style of van Emden and Kowalski [71].

Definition 4.3.
Let $\mathcal{A}$ be a constraint system and $P \in CLP(\mathcal{A})$. The mapping $T_P : \wp(\mathcal{B}^{\mathcal{A}}) \rightarrow \wp(\mathcal{B}^{\mathcal{A}})$, is defined as follows

$$T_P(I) = \bigcup_{C \in P} \left\{ [p(\bar{x}) \ :- \ \exists(\tilde{c})_{\bar{x}}]_{\sim} \ \middle| \ \begin{array}{l} C \equiv \text{`}p(t) \ :- \ c \ [\!] \ p_1(\bar{t}_1), \ldots, p_n(\bar{t}_n)\text{'} \\ n \geq 0, \bar{x} \ ind \ C \text{ and for each } i = 1...n : \\ \qquad p_i(\bar{x}_i) \ :- \ c_i \ll_{C, \bar{x}_1, \ldots, \bar{x}_{i-1}} I \\ \qquad \bar{x}_i \cap \bar{x} = \emptyset, c'_i = d_{\bar{x}_i, \bar{t}_i} \otimes c_i, \\ \tilde{c} = d_{\bar{x}, \bar{t}} \otimes c \otimes c'_1 \otimes \cdots \otimes c'_n \end{array} \right\}$$

Interestingly, it turns out that the fixed-point semantics of a program can always be computed into a finite dimension constraint system. This follows from the properties of cylindrification with respect to substitution (see Theorem 3.1). Intuitively, the hiding operator allows the definition of "local environments" that cannot be influenced by substitution, and allows hidden variables to be "recycled" outside the scope of the hiding operator, making it possible to get by with only a finite set of variables that are recycled over and over. This is useful for program analysis purposes, since it simplifies the construction of Noetherian abstract domains (e.g., see the affine relation analysis in Section 6.1).

By analogy with the operator $\oplus$, which expresses the notion of "merging together" the information present in two constraints, we define a *condensing* operator $^{\flat} : \wp(\mathcal{B}^{\mathcal{A}}) \longrightarrow \wp(\mathcal{B}^{\mathcal{A}})$ such that for any $I \in \mathcal{B}^{\mathcal{A}}$:

$$I^{\flat} = \left\{ [p(\bar{x}) \ :- \ \sum \{ \partial^{\bar{x}}_{\bar{y}} c' \mid p(\bar{y}) \ :- \ c' \ll_{\bar{x}} I \}]_{\sim} \mid p \in preds(I) \right\}$$

The operator $^{\flat}$ captures the notion of merging together the information present in a set of constrained atoms. The result of condensing is an interpretation containing at most one constrained atom for each predicate symbol p in the program. Such constrained atoms have the form '$p(\bar{x}) \ :- \ \sum c_j$' and represent the join (intuitively corresponding to disjunction) of all the answer constraints c_j for the goal $p(\bar{x})$. Since the number of such answer constraints can be infinite, infinite joins of constraints are allowed in constrained atoms. This is modeled by having the universe $\mathcal{C}$ of a constraint system be closed under infinite joins. Note that this closure property cannot be specified by any finitary first-order formula.

To specify the relation between interpretations and condensed interpretations, we consider a *lower powerdomain* preorder $\sqsubseteq$. Let $a \ll I$ denote a variant of an object $a \in I$ that has been renamed apart from all elements of I. The preorder $\sqsubseteq$ is defined as follows: $I \sqsubseteq I'$ iff for each $p(\bar{x}) \ :- \ c \ll I$ there exists $p(\bar{x}) \ :- \ c' \ll I'$ such that $c \trianglelefteq c'$. Let $\approx$ denote the induced equivalence relation: $I_1 \approx I_2$ iff $I_1 \sqsubseteq I_2$ and $I_2 \sqsubseteq I_1$. In the discussion that follows, we will be concerned primarily with the partial order over $\wp(\mathcal{B}^{\mathcal{A}})/\approx$ induced by $\sqsubseteq$. For simplicity of exposition, we abuse notation and use $\sqsubseteq$ to denote this partial order and $\wp(\mathcal{B}^{\mathcal{A}})$ to denote the set $\wp(\mathcal{B}^{\mathcal{A}})/\approx$. It is easy to prove that $\langle \wp(\mathcal{B}^{\mathcal{A}}), \sqsubseteq \rangle$ is a complete lattice, with join operator $\sqcup$ defined as $I \sqcup I' = (I \cup I')^{\flat}$.

Proposition 4.1.
$^\flat$ *is an upper closure operator on* $\langle \wp(\mathcal{B}^{\mathcal{A}}), \sqsubseteq \rangle$.

We denote by $\wp^\flat(\mathcal{B}^{\mathcal{A}})$ the set of condensed interpretations $(\wp(\mathcal{B}^{\mathcal{A}}))^\flat$. It is easy to prove that for any $I, I' \in \wp^\flat(\mathcal{B}^{\mathcal{A}})$: $I \sqsubseteq I'$ iff $(I \cup I')^\flat = I'$, and that $\langle \wp^\flat(\mathcal{B}^{\mathcal{A}}), \sqsubseteq \rangle$ is a complete lattice.

An analogous operator $T^\flat_P : \wp^\flat(\mathcal{B}^{\mathcal{A}}) \longrightarrow \wp^\flat(\mathcal{B}^{\mathcal{A}})$ on condensed interpretations can be defined as $T^\flat_P(I) = (T_P(I))^\flat$. The existence and uniqueness of the least fixpoints of these operators is, in both cases, a consequence of continuity of T_P and $T^\flat_P$:

Lemma 4.3.
Let $\mathcal{A}$ be a constraint system and $P \in CLP(\mathcal{A})$. For any $I \in \wp(\mathcal{B}^{\mathcal{A}})$: $(T_P(I^\flat))^\flat = (T_P(I))^\flat$.

Proposition 4.2.
Let $\mathcal{A}$ be a constraint system and $P \in CLP(\mathcal{A})$. T_P is a continuous function on the complete lattice $\langle \wp(\mathcal{B}^{\mathcal{A}}), \subseteq \rangle$ and $T^\flat_P$ is continuous on the complete lattice $\langle \wp^\flat(\mathcal{B}^{\mathcal{A}}), \sqsubseteq \rangle$.

Definition 4.4. [fixed-point semantics]
The *fixed-point semantics* of a program P over a constraint system $\mathcal{A}$ is given by $\mathcal{F}(P) = \mathit{lfp}(T_P)$ and $\mathcal{F}^\flat(P) = \mathit{lfp}(T^\flat_P)$.

The following result states the equivalence between the operational and the (possibly condensed) fixed-point semantics, for any constraint system $\mathcal{A}$.

Theorem 4.1.
Let $\mathcal{A}$ be a constraint system with dimension ω, and $P \in CLP(\mathcal{A})$, then $\mathcal{F}(P) = \mathcal{O}(P)/{\sim}$ and $\mathcal{F}^\flat(P) = (\mathcal{O}(P)/{\sim})^\flat$.

It is worth noting that the condensing operator $^\flat$ is actually an abstract interpretation. Indeed, with any condensed interpretation $I^\flat$, there are many (possibly) different interpretations J such that $J^\flat = I^\flat$. While the non-condensed semantics associates with each predicate the collection of *all* possible constraints that one may obtain for it, the condensed one associates a single constraint with each predicate defined in the program. The latter case is particularly useful for specifying termination conditions in terms of ascending chains, ordered by entailment ($\trianglelefteq$), of constraints (see Section 5).

Observation 4.2. Note that from Theorem 4.1, the semantics $\mathcal{F}(P)$ corresponds precisely to the s-semantics, which is well known to be fully abstract with respect to computed answer substitutions in (pure) logic programming ([10, 31]). This because $\mathcal{O}$ characterizes precisely the set of computed answer constraints for arbitrary atomic goals. In this case, when an atomic goal $p(x)$ has the two answer constraints $x = a$ and $true$ in the Herbrand constraint system, we obtain the denotation $\{[p(x)\ :-\ x = a]_\sim, [p(x)\ :-\ true]_\sim\}$. A similar approach to characterize computed answer constraints in constraint logic programming is *also considered in [33]. The condensed semantics instead, corresponds to the so called Clark's semantics [14] (c-semantics in [31]), which characterizes* correct answer substitutions *in logic programming. In this case, for the atomic goal* $p(x)$ *above, we obtain the denotation* $\{[p(x)\ :-\ true]_{\sim}\}$. *This semantics is proved to be* optimal *for* ground dependency *and* covering *analysis in [35]. The relation between collecting semantics for logic programs and abstract interpretation has been recently studied in [35] for a number of different observable properties.*

The semantics given thus far in this section generalize the corresponding results for traditional logic programs to arbitrary constraint systems. We conclude this section with an example that shows that they can be used for other, very different, purposes as well.

4.3. Machine-level Traces

This example illustrates a non-standard semantics for constraint logic programs, that of machine-level traces, as an instance of the framework of this paper (Stoy discusses similar non-standard semantics in a denotational context [67]). Such a semantics is essential, for example, if we wish to reason formally about the correctness of a compiler (e.g., see [39]), low-level compiler optimizations, or about the behavior of debuggers or profilers. Instead of constrained atoms where each atom is associated with a constraint, this semantics will associate with each atom a set of instruction sequences that may be generated on an execution of that atom.

Suppose we are given some low-level WAM-like abstract machine for the execution of CLP programs. Let Instr denote the (possibly infinite) set of all possible machine instructions (by "instruction" we mean an instruction name—the opcode—together with the values of the operands). A computation is defined by a sequence of states obtained as instructions are executed. If each instruction is a function over states, and we assume that all programs start execution in some given (fixed) initial state, then the results of a computation can be specified simply by the sequence of instructions executed. We refer to such a sequence as a *trace*. The set of all traces is denoted by $\mathsf{Trace} = \mathsf{Instr}^*$. The meaning of a program is given by the set of all of its possible executions, i.e., by a set of traces. In the case of a low-level trace semantics for constraint logic programs, therefore, the universe is given by $\mathcal{C} = \wp(\mathsf{Trace})$.

In general, certain minimal capabilities are necessary in any low-level instruction set in order to execute a constraint logic program. To this end, we assume the following:

1. Corresponding to each primitive constraint c of the language there is a sequence of machine instructions $impl(c)$ that realizes c at the machine level.

2. There is an instruction $\mathtt{hide}(x)$ with the following behavior: for any variable x, $\mathtt{hide}(x)$ removes any constraint on the variable x in the data structures representing the computed constraint at that point.

The basic operations on sets of traces are defined as follows: given $S, S_1, S_2 \in \mathcal{C}$:

1. $S_1 \oplus S_2 = S_1 \cup S_2$.

2. $\otimes$ is pointwise concatenation: let '::' denote the concatenation operation on sequences, then $S_1 \otimes S_2 = \{s_1 :: s_2 \mid s_1 \in S_1, s_2 \in S_2\}$.

3. $\mathbf{0} = \emptyset$.

4. $\mathbf{1} = \{\varepsilon\}$.

5. Let $X = \{x_1, \ldots, x_n\}$, then $\exists_X S = \{s :: \langle \mathtt{hide}(x_1), \mathtt{hide}(x_2), \cdots, \mathtt{hide}(x_n)\rangle \mid s \in S\}$.

6. $d_{t,t'} = \mathit{impl}(t = t')$.

In a low level machine, the constraints manipulated and accumulated during the execution of a program are necessarily represented in terms of machine-level entities, e.g., by means of data structures constructed in memory. It follows that references to constraints c in the derivation relation $\rightsquigarrow_P$ or the immediate consequence operator T_P will, in the low-level semantics, be replaced by references to $\mathit{impl}(c)$. The corresponding high-level constraints can be reconstructed where necessary, e.g., for displaying a computed answer constraint to the user, or for debugging purposes. Given our assumption that there is a single initial state that every program begins execution in, given a trace s it is possible to reconstruct the constraint obtained in the state resulting from the execution of s: this is denoted by $\mathit{constraint}(s)$. This extends in the obvious way to sets of traces: given any $S \in \mathcal{C}$, $\mathit{constraint}(S) = \{\mathit{constraint}(s) \mid s \in S\}$. Define the relation $\simeq \subseteq \mathcal{C} \times \mathcal{C}$ as follows: for any $S_1, S_2 \in \mathcal{C}$: $S_1 \simeq S_2$ if and only if $\mathit{constraint}(S_1) = \mathit{constraint}(S_2)$. $\simeq$ is an equivalence relation.

It is easy to see that the structure $(\mathcal{C}, \otimes, \oplus, \mathbf{1}, \mathbf{0})$ satisfies the axioms of a closed semiring, so Axiom R in the definition of constraint systems is satisfied. The remaining axioms, namely $C_1 - C_5$ and $D_1 - D_4$, are satisfied modulo the equivalence relation $\simeq$. Thus, the machine level semantics presented forms a constraint system modulo $\simeq$.

As a simple example of an application of such a semantics, consider the following program over the Herbrand constraint domain:

```
p(X) :- X = a, q(Z).
q(Y) :- Y = a.
q(Y) :- Y = b.
```

The only primitive constraint over this domain is '='/2: suppose that the low-level instruction set under consideration contains an instruction `unify` such that $\mathit{impl}(t_1 = t_2) = \mathtt{unify}(t_1, t_2)$. In addition, assume the instructions `call`, `return`, and `fail` for managing procedure calls. The (operational) semantics of the procedure `q` is then given by

$$\mathtt{q}(\mathtt{Y}_i) \ :- \ \{\langle \mathtt{unify}(\mathtt{Y}_i, \mathtt{a}), \mathtt{return}\rangle,\ \langle \mathtt{unify}(\mathtt{Y}_i, \mathtt{b}), \mathtt{return}\rangle\} \mid i \geq 0\}.$$

Here, the subscripts on the variables denote alphabetic variants of the program clauses that may be used at runtime: the idea is that there is a trace describing the execution of every possible variant of the clause appearing in the source program. Thus, the meaning of a predicate is an infinite set of traces representing instruction sequences that may be obtained at runtime, rather than finite sets of instruction sequences that may be generated by a compiler. The semantics for the procedure `p` can similarly be obtained as:

$$\begin{aligned}\texttt{p}(\texttt{X}_i) \;\texttt{:-}\; \{&\langle \texttt{unify}(\texttt{X}_i,\texttt{a}), \texttt{unify}(\texttt{Z}_i,\texttt{Y}_i), \texttt{call q/}1, \texttt{unify}(\texttt{Y}_i,\texttt{a}), \texttt{return},\\ &\quad \texttt{hide}(\texttt{Z}_i), \texttt{return}\rangle,\\ &\langle \texttt{unify}(\texttt{X}_i,\texttt{a}), \texttt{unify}(\texttt{Z}_i,\texttt{Y}_i), \texttt{call q/}1, \texttt{unify}(\texttt{Y}_i,\texttt{b}), \texttt{fail}, \texttt{fail}\rangle \mid i \geq 0\}\end{aligned}$$

We have deliberately kept the instruction set under consideration here small, in order to simplify the presentation. It is not difficult to see how such an instruction set could be embellished to be more realistic. For example, argument passing through a fixed set of registers, as in the WAM, can be modelled by requiring that the arguments in the head of each clause of an n-ary predicate be distinct variables $\texttt{A}_1, \ldots, \texttt{A}_n$; if a is a constant and a variable x occurs for the first time in a trace for a procedure in an instruction '$\texttt{unify}(x, a)$', we could replace this instruction by a more specialized one of the form '$\texttt{get\_constant}(x, a)$' (and similarly for function symbols of nonzero arity); and so on.

5. ABSTRACT CONSTRAINT SYSTEMS

The definition of an abstract constraint system, which specifies a non-standard semantics for a constraint programming language, is performed in two steps: *term abstraction* and *constraint abstraction*. In the first step new syntactic objects are introduced to represent concrete terms. In the second one, constraints on the abstracted term system are defined.

In general, a constraint system is an interpretation (in a closed semiring) for constraint formulae. To relate constraint systems, we follow the approach to "static semantic correctness" in [7]. Correctness of non-standard semantic specifications can be handled in an algebraic way through the notion of morphism (see [70]). The algebraic notion of morphism can be made less restrictive by assuming that the carriers of the algebras involved are partially ordered sets. We use this weaker notion of morphism between algebraic structures, capturing the approximation possibly induced by abstract interpretations or by any approximate semantics defined in the framework. This provides, at the same time, a characterization for domain correctness conditions (traditionally specified by Galois connections) and the correctness of abstract operations.

Definition 5.1. [morphism, semimorphism]
Let τ and τ' be term systems over sets of variables V and V', and with substitution operators s and s' respectively. A morphism $\kappa : \tau \xrightarrow{m} \tau'$, is a function mapping terms of τ to terms of τ' such that for any $t_1, t_2 \in \tau$ and $x \in V$: $\kappa(s_x(t_1, t_2)) = s'_{\kappa(x)}(\kappa(t_1), \kappa(t_2))$. Consider constraint systems $\mathcal{A}$ and $\mathcal{A}'$, where

$$\mathcal{A} = (\mathcal{C}, \otimes, \oplus, \mathbf{1}, \mathbf{0}, \exists_X, d_{t_1,t_2})_{X \subseteq V; t_1, t_2 \in \tau}$$

and

$$\mathcal{A}' = (\mathcal{C}', \otimes', \oplus', \mathbf{1}', \mathbf{0}', \exists'_X, d'_{t_1,t_2})_{X \subseteq V'; t_1, t_2 \in \tau'}.$$

A mapping $\alpha_\kappa : \mathcal{A} \xrightarrow{s} \mathcal{A}'$ is a semimorphism iff there is a morphism of term systems $\kappa : \tau \xrightarrow{m} \tau'$ such that for each $c, c_1, c_2 \in \mathcal{C}$, $C \subseteq \mathcal{C}$, $X \subseteq V$ and $t_1, t_2 \in \tau$, the following hold:

1. $\alpha_\kappa(\mathbf{0}) = \mathbf{0}'$;

2. $\alpha_\kappa(\mathbf{1}) \trianglelefteq' \mathbf{1}'$;
3. $\alpha_\kappa(\sum C) \trianglelefteq' \sum' \alpha_\kappa(C)$;
4. $\alpha_\kappa(\exists_X c) \trianglelefteq' \exists'_{\kappa(X)} \alpha_\kappa(c)$;
5. $\alpha_\kappa(c_1 \otimes c_2) \trianglelefteq' \alpha_\kappa(c_1) \otimes' \alpha_\kappa(c_2)$;
6. $\alpha_\kappa(d_{t_1,t_2}) \trianglelefteq' d'_{\kappa(t_1),\kappa(t_2)}$.

The intuition behind this definition may be understood as follows. Recall that the natural order $\trianglelefteq'$ over $\mathcal{C}'$ is defined as $x \trianglelefteq' y$ iff $x \oplus' y = y$, where $\oplus'$ intuitively denotes some kind of "merge" operation. For the purposes of abstract interpretation, the objects that are merged in this manner represent possible program behaviors, and the smaller the set of behaviors denoted by an object the more information it conveys. Thus, $x \trianglelefteq' y$ denotes that x provides more information than y, i.e., is a more precise description of program behavior. The requirements for a semimorphism given above, therefore, state simply that for each of the operations in the (concrete) constraint system $\mathcal{A}$, operating on objects in $\mathcal{C}$ and then applying the semimorphism (i.e., abstracting) is no worse than applying the semimorphism first and then applying the corresponding operation in the (abstract) constraint system $\mathcal{A}'$. The following proposition states that semimorphisms correctly abstract the (derived) notion of substitution into constraints:

Proposition 5.1 (substitution correctness).
Let $\mathcal{A}$ and $\mathcal{A}'$ be constraint systems as above. Let also $c \in \mathcal{C}$, $x \in V$ and $t \in \tau$ such that x ind t. If $\alpha_\kappa : \mathcal{A} \xrightarrow{s} \mathcal{A}'$ is a semimorphism then $\alpha_\kappa(\partial^t_x c) \trianglelefteq' \partial'^{\kappa(t)}_{\kappa(x)} \alpha_\kappa(c)$.

For notational simplicity in the discussion that follows, we will sometimes omit the subscript from a semimorphism when the morphism κ on the underlying term system need not be considered explicitly.

We are now able to provide a notion of correctness for constraint systems. It corresponds precisely to the Galois insertion-based notion of domain and operator correctness belonging to the classical framework of abstract interpretation [22], as specified by Proposition 5.3 below. Here, the unifying framework of constraint systems provides a uniform treatment for domain and operator correctness, both specified by the simple notion of semimorphism.

Definition 5.2.
Let $\mathcal{A}$ and $\mathcal{A}'$ be constraint systems as above. $\mathcal{A}'$ is *correct* with respect to $\mathcal{A}$ iff there exists a semimorphism α_κ (i.e., $\kappa : \tau \xrightarrow{m} \tau'$ and $\alpha : \mathcal{A} \xrightarrow{s} \mathcal{A}'$) that is a surjective and additive mapping of $\langle \mathcal{C}, \trianglelefteq \rangle$ into $\langle \mathcal{C}', \trianglelefteq' \rangle$.

The following proposition provides the basis for designing abstract constraint systems by consecutive approximations.

Proposition 5.2.
For any constraint system $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{A}''$: if $\mathcal{A}''$ is correct with respect to $\mathcal{A}'$ and $\mathcal{A}'$ is correct with respect to $\mathcal{A}$, then $\mathcal{A}''$ is correct with respect to $\mathcal{A}$.

Additivity and surjectivity allow the semimorphism to associate the "best" approximating constraint in $\mathcal{A}'$ with any concrete constraint in $\mathcal{A}$. This is captured by the notion of *Galois insertion*, where a pair of functions (α, γ)—denoting *abstraction* and *concretization* respectively—is a Galois insertion of $\langle \mathcal{C}', \trianglelefteq' \rangle$ into $\langle \mathcal{C}, \trianglelefteq \rangle$ iff α and γ are monotonic, $\alpha(\gamma(c')) = c'$ and $c \trianglelefteq \gamma(\alpha(c))$ for each $c \in \mathcal{C}$ and $c' \in \mathcal{C}'$ ([22, 24, 60]). The following proposition relates the notion of semimorphism with the notion of Galois insertion:

Proposition 5.3.
Let $\mathcal{A}$ and $\mathcal{A}'$ be constraint systems with universes $\mathcal{C}$ and $\mathcal{C}'$ respectively. If $\mathcal{A}'$ is correct with respect to $\mathcal{A}$ by means of a semimorphism α, there exists a mapping $\gamma : \mathcal{C}' \longrightarrow \mathcal{C}$ such that (α, γ) is a Galois insertion of $\langle \mathcal{C}', \trianglelefteq' \rangle$ into $\langle \mathcal{C}, \trianglelefteq \rangle$.

Notice that, as observed in [24], by additivity and surjectivity, $\sum\{c \mid \alpha(c) \trianglelefteq' c'\} = \sum\{c \mid \alpha(c) = c'\}$.

In the framework of abstract interpretation, correctness of fixed-point approximations requires some additional conditions on correctness of the non-standard (abstract) semantic operators [22]. With the assumption of additivity, semimorphisms are adequate for specifying both Galois insertions, as seen in Proposition 5.3, and operator-correctness. Let $\mathcal{A}'$ be a constraint system that is correct with respect to $\mathcal{A}$, by means of a semimorphism α_κ. Let $P = \{C_1, ..., C_m\}$ be a program in $CLP(\mathcal{A})$. The *corresponding program on $\mathcal{A}'$*, denoted $\mathcal{T}_{\alpha_\kappa}(P)$ is a set of clauses $\{C'_1, ..., C'_m\}$ such that for each $i = 1, ..., m$ if $C_i =$ '$p(\bar{t}) \;:-\; c \;[\!]\; p_1(\bar{t}_1), \ldots, p_n(\bar{t}_n)$' then $C'_i =$ '$p(\kappa(\bar{t})) \;:-\; \alpha(c) \;[\!]\; p_1(\kappa(\bar{t}_1)), \ldots, p_n(\kappa(\bar{t}_n))$' where κ extends elementwise on tuples of terms. Therefore, if P specifies a set of relations on $\mathcal{A}$, then $\mathcal{T}_{\alpha_\kappa}(P)$ specifies a corresponding set of relations on $\mathcal{A}'$. Correctness of $\mathcal{A}'$ with respect to $\mathcal{A}$ provides the correctness of the relations defined in $\mathcal{T}_{\alpha_\kappa}(P)$ (the semantics of $\mathcal{T}_{\alpha_\kappa}(P)$) with respect to those defined in P (the semantics of P). The following theorem relates the semantics of a program with that of a corresponding one defined on a correct constraint system.

Theorem 5.1.
Let $P \in CLP(\mathcal{A})$ and $P' \in CLP(\mathcal{A}')$ be the corresponding program on $\mathcal{A}'$. If $\mathcal{A}'$ is correct with respect to $\mathcal{A}$, there exists $\beta : \wp(\mathcal{B}^{\mathcal{A}}) \rightarrow \wp(\mathcal{B}^{\mathcal{A}'})$ such that $\beta(\mathcal{F}(P)) \sqsubseteq' \mathcal{F}(P')$ and $\beta(\mathcal{F}^\flat(P)) \sqsubseteq' \mathcal{F}^\flat(P')$.

It is worth noting that the relation between the semantics of a program and that of the corresponding one on a correct constraint system corresponds precisely to the correctness condition in abstract interpretation. Therefore, dataflow analysis for a program can be obtained by transforming it (by $\mathcal{T}_\alpha$) into a corresponding program defined on an abstract (approximated) constraint system (see later Section 5.2 for a formal treatment of constraint approximation). The key point here is that both the concrete program P and the corresponding abstract one $\mathcal{T}_{\alpha_\kappa}(P)$ are CLP programs (i.e, $\mathcal{T}_{\alpha_\kappa}$ is a program transformation), and the corresponding semantic interpretations are instances, over two different constraint systems, of the same generalized semantics for CLP, as shown in Figure 1 (see [42] for a discussion of implemented systems that use this transformational approach for the analysis of Prolog programs).

Given a (fixed-point) concrete semantics, dataflow analysis usually requires computing the limit of Kleene chains. Convergence to the least fixed-point in finitely

$$\begin{tikzcd}
P \arrow[r, "\mathcal{F}^{(\flat)}"] \arrow[d, "\mathcal{T}_\alpha"'] & \mathcal{F}^{(\flat)}(P) \arrow[d, "\beta"] \\
\mathcal{T}_\alpha(P) \arrow[r, "\mathcal{F}^{(\flat)}"'] & \mathcal{F}^{(\flat)}(\mathcal{T}_\alpha(P))
\end{tikzcd}$$

Figure 1. Abstract interpretation by program transformation.

many steps can be obtained either by requiring the abstract domain to satisfy the ascending chain condition, or by using *widening* operators to force convergence [22]. In the following we consider the conditions on the constraint system that ensure that the resulting abstract domain satisfies the ascending chain condition. We will focus primarily on abstract interpretation by condensing interpretations. This is because condensing provides a description of the multiplicity of answer constraints in terms of (possibly infinite) joins in the constraint system. This is essential in abstract interpretation by program transformation, where the termination of the analysis has to follow from the structure of the constraint system (this condition is satisfied by several well known constraint systems useful for analysis, e.g., see Section 6.1 below). We introduce the ascending chain condition on constraint systems and we show how this condition ensures finiteness in fixed-point computations. This approach is more closely related to the constraint system structure than the widening one, which is in turn more related with the (semantic) fixed-point computation.
A set of constraints $\{c_1, ..., c_n, ..\}$ is said to be *free-variable bounded* if there is a finite set of variables V such that $FV(c_i) \subseteq V$ for each $i \geq 1$. The following definition is important for abstract interpretation purposes:

Definition 5.3.
A constraint system $\mathcal{A}$ is *Noetherian* iff its universe $\mathcal{C}$ does not contain any infinite ascending chain of free-variable bounded constraints.

The free-variable-boundedness condition here is important, for otherwise any constraint system with a denumerable set of variables is not Noetherian. To see this, consider the constraints $c_i \equiv X_1 \vee \cdots \vee X_i$: the set of constraints $\{c_i \mid i \geq 1\}$, ordered by entailment, forms an infinite ascending chain even on a two-valued boolean interpretation. However, it is easy to see that this set is not free-variable-bounded.

Given a Noetherian constraint system $\mathcal{A}$, it is easy to prove that the set of $\mathcal{A}$-interpretations $\wp^\flat(\mathcal{B}^{\mathcal{A}})$ is Noetherian. An *abstract constraint system* is then a Noetherian constraint system. Let $\mathcal{A}$ be a constraint system, then a correct abstract interpretation for constraints in $\mathcal{A}$ is a tuple $(\mathcal{A}, \alpha_\kappa, \mathcal{A}^a)$ where $\mathcal{A}^a$ is an abstract constraint system and α_κ is a semimorphism which specifies the correctness of the abstraction process.

Different semantic characterizations lead to different abstract evaluation strategies. *Top-down* abstract interpretation corresponds to the abstraction of the standard operational semantics discussed in Section 4.1. Our approach to top-down abstract interpretation encompasses various abstract interpretation frameworks defined in the literature. For example, Bruynooghe's top-down abstract interpretation scheme for positive logic programs [11], based on an AND/OR-tree construction, encodes our interpretation structure in a corresponding tree-structure where AND-nodes interpret the $\otimes$ operator and OR-nodes implement the $\oplus$ operation on constraints. As usual, abstract unification is encoded by appropriately defining the $\otimes$ operator. The search strategy is the same as the one given in [11]. *Bottom-up* abstract interpretation, on the other hand, allows the computation of finite approximations to the fixed-point semantics associated with a given constraint logic program (this approach has been applied to static analysis of pure logic programs in [5]). Given an abstract constraint system, the corresponding abstract transformation map is defined as in the concrete case, by considering the corresponding abstract operators instead of the concrete ones. As in the pure logic programming case, the correctness of a suitable set of operators implies the correctness of the entire framework (both top-down and bottom-up). In the constraint logic programming case, the correctness of the analysis corresponds then to the correctness of the constraint system, as shown in Theorem 5.1. In the following we will concentrate on bottom-up (fixed-point-based) abstract interpretations only. For any Noetherian lattice $\wp^\flat(\mathcal{B}^{\mathcal{A}})$, we have:

Proposition 5.4.
Let $\mathcal{A}$ be an abstract constraint system. If $P \in CLP(\mathcal{A})$, there is a finite $k \geq 0$ such that $\mathcal{F}^\flat(P) = T^\flat_P \uparrow k(\emptyset)$.

Proposition 5.4 does not hold in general for non-condensed interpretations, unless the constraint system is finite.

Observation 5.1. It is worth noting that our constraint system construction imposes some restrictions on the traditional lattice-based theory of abstract interpretation [24]. However, most of these restrictions come from the standard interpretation of the fundamental (domain-dependent) operators involved in logic programming. Indeed, from the domain viewpoint, the basic restriction is only the distributivity law of closed semirings. The impact of this law in the semantics of CLP programs and the application of a weaker structure in dataflow analysis is discussed later in Section 6. The remaining laws are associated with domain-dependent operators and formalize their expected behaviour. These operators are quite common in most of the frameworks for abstract interpretation of logic programs (e.g., see [5, 11]). Our approach has the advantage of axiomatically unifying all these operators into a single general structure: the constraint system. The definition of a common structure underlying the construction of abstract domains and operators has many important benefits, in particular: (1) it summarizes the general *properties of domain dependent operators, which should be invariant with respect to abstraction, in order to preserve the standard properties of the semantics; and (2) it provides an immediate correspondence between well known structures of constraints and the intended dataflow analysis (e.g., see the constraint system of propositional formulae in the next section or the system of linear equalities in Section 6.1). Of course, abstract domains which are not complete lattices, and operators that do not satisfy the axioms, cannot be modeled as constraint systems in our framework. The* rigid *structure of constraint systems can be weakened to include more analyses, as discussed in Section 6.*

5.1. An Example: Rigidity Analysis

A number of researchers have considered abstract interpretation techniques for the analysis of *ground dependences* for pure logic programs (see, for examples, [5, 21, 40, 58, 59]); this notion can be generalized to that of *rigidity* with respect to size measures, or "norms", for terms. Intuitively, a *norm* is a function from the set of terms to the set of natural numbers such that the norm of a term depends only on the its principal functor and (some of) its subterms. In the following we consider the *length* and *size* norms on the Herbrand term system:

$|t|_{length} = 0$ if t is a variable or $t = []$,
$|t|_{length} = 1 + |tail|_{length}$ if $t = [h|tail]$,

$|t|_{size} = 1$ if t is a variable or a constant,
$|t|_{size} = 1 + |t_1|_{size}, ..., |t_n|_{size}$ if $t = f(t_1, ..., t_n)$.

Given a norm $|\cdot|$, a variable x is said to be *relevant* to a term t with respect to $|\cdot|$ if there is some term t' such that $|\ t\ | \neq |\ s_x(t', t)\ |$.

Definition 5.4. [rigidity [9]]
Given a term system $\tau = (T, Sub, V)$, a term $t \in \tau$ is rigid with respect to a norm $|\cdot|_S$ on τ iff $|\sigma(t)|_S = |t|_S$ for every substitution $\sigma \in Sub$.

Consider the term system $\tau_{(\Sigma,V)}$ being defined over a finite set of variables V. Let us consider the term system τ_Σ as defined in Example 2, where $\Sigma = V$: terms are finite sets of (relevant) variables with respect to a given norm. Rigid terms are denoted by the empty set of variables. Given a norm $|\cdot|_S$, consider the mapping $Vrel_S : \tau \rightarrow \tau_V$:

$$Vrel_S(t) = \left\{\ v \in V \ \middle|\ v \text{ is relevant to t with respect to } |\cdot|_S\ \right\}.$$

It is easy to see that the traditional notion of *groundness* is a special case of rigidity under the selection of the norm *size*, since $Vrel_{size}(t) = \emptyset$ iff t is ground.

Proposition 5.5.
$Vrel_S$ is a morphism of term systems.

Marriott and Søndergaard have proposed an elegant domain, named *Prop*, to represent ground dependences among arguments in atoms ([21, 56, 58, 59]). This domain can be expressed as an instance of our framework using the algebra of propositional formulae with disjunction. Let $Prop = (Prop_V, \wedge, \vee, true, false, \exists_X, \wedge(t) \leftrightarrow \wedge(t'))_{X \subseteq V; t,t' \in \tau_V \cup \{\emptyset\}}$ be the algebra of possibly existentially quantified disjunctions of formulae, defined on the term system τ_V, by the connectives $\wedge$ and $\leftrightarrow$; where, for each finite set of variables $\{x_1, ..., x_m\} \in \tau_V$: $\wedge(\{x_1, ..., x_m\}) = x_1 \wedge ... \wedge x_m$, and $\wedge(\emptyset) = true$. Intuitively, the formula $x \wedge y \wedge z \leftrightarrow w \wedge v$ represents an equation $t = t'$ where $Vrel_S(t) = \{x, y, z\}$ and $Vrel_S(t') = \{w, v\}$; $x \wedge y$ 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. Since $x \leftrightarrow true$ is equivalent to x, a variable x that is guaranteed to be bound to a ground term is denoted x (i.e., the expression x denotes that x is rigid). It is easy to prove that, because of the finiteness of V, $Prop/\leftrightarrow$ is a finite constraint system.

In this section we outline the proof of correctness for the constraint system $Prop/\leftrightarrow$ with respect to $\mathcal{H}$. Recall that an equation set is in *solved form* if it has the form $\{v_1 = t_1, ..., v_n = t_n\}$ where the v_i's are distinct variables that do not occur in the right hand side of any equation [53]. Any simple equational constraint can be transformed into an equivalent constraint of the form $\exists_X c$ where c is in solved form. In particular we say that a quantified set of equations is in solved form if it has the form $\exists_X \{v_1 = t_1, ..., v_n = t_n\}$ where $\{v_1 = t_1, ..., v_n = t_n\}$ is in solved form and $X \subseteq \cup\{var(t_i) \mid 1 \leq i \leq n\}$. Given a norm $|\cdot|_S$, each set of equational constraints $c = \{x_1 = t_1, ..., x_n = t_n\}$ in $\mathcal{H}$ is associated with a boolean expression specifying rigidity relationships among (relevant) variables by means of a mapping α_S that is defined as follows:

$$\alpha_S(c) = \bigwedge_{i=1}^{n} (x_i \leftrightarrow \wedge(Vrel_S(t_i))).$$

Let E, E' be two equivalent (finite) sets of equations and let $sol(E)$ and $sol(E')$ denote the corresponding (quantifier-free) sets of equations in solved form. In this case, correctness follows from the observation that any two sets of equations in solved form are equivalent iff they are isomorphic, where a solved form equation set E is isomorphic to E' iff there is a subset $\{x_1 = y_1, ..., x_k = y_k\}$ of E where y_i's are distinct variables such that $E' = E[y_1/x_1, ..., y_k/x_k, x_1/y_1, ..., x_k/y_k]$ (see Theorem 3.13, page 81 in [53]). It is straightforward to prove that if E and E' are equivalent sets of equations then $\alpha_S(sol(E)) \leftrightarrow \alpha_S(sol(E'))$. Since $Prop_V$ is finite, we can extend α_S to be an additive semimorphism from the constraint system $\mathcal{H}$ to $Prop$: if $c = \cup\{\exists_{X_i} c_i \mid i \in I\}$ is an arbitrary $\mathcal{H}$-constraint (where c_i are simple constraints), for a possibly infinite set of indices I, we define $\alpha(c) = \vee\{\exists_{X_i} \alpha_S(sol(c_i)) \mid i \in I\}$.

Theorem 5.2.
α is an additive semimorphism from the constraint system $\mathcal{H}$ to $Prop$.

Example 5.1. Notice that, because of the use of solved form equation sets, α behaves as a semimorphism. Consider the equation $e = \{[x|y] = [z|[w|h]]\}$ with the norm "*length*" (l). While $\alpha(e) = \alpha_l(\{x = z, y = [w|h]\}) = \{x \leftrightarrow z, y \leftrightarrow h\}$, the diagonal element is $\{Vrel_l([x|y]) \leftrightarrow Vrel_l([z|[w|h]])\} = \{y \leftrightarrow h\}$. It is easy to see that the diagonal element is weaker than the abstraction of the corresponding concrete constraint.

Example 5.2. Consider the norm "*length*" and the following constraint logic program on the Herbrand constraint system P specifying the *append* procedure:

```
append([], L, L).
append([H|Y], X2, [H|Z]) :- append(Y, X2, Z).
```

The abstract semantics for length-rigidity analysis is

$$\begin{array}{rcl} T_P^\flat\!\uparrow\!0(\emptyset) & = & \emptyset \\ T_P^\flat\!\uparrow\!1(\emptyset) & = & append(x_1,x_2,x_3) \ :-\ x_1 \wedge (x_2 \leftrightarrow x_3) \\ T_P^\flat\!\uparrow\!2(\emptyset) & = & append(x_1,x_2,x_3) \ :-\ (x_1 \wedge x_2 \leftrightarrow x_3) \vee \\ & & \qquad \exists_{\{x_1',x_2',x_3'\}}(x_1 \leftrightarrow x_1' \wedge x_2 \leftrightarrow x_2' \leftrightarrow x_3' \leftrightarrow x_3) \\ & = & append(x_1,x_2,x_3) \ :-\ x_1 \wedge (x_2 \leftrightarrow x_3) \qquad\qquad \text{(fixed-point)} \end{array}$$

The abstract semantics obtained above generalizes the standard ground behavior to length-rigidity behavior: "the second argument list-length can change iff the third argument does". In ground dependence analysis $Vrel_{size}(t) = var(t)$ and the abstract meaning of *append* is described by the relation

$$append(x_1,x_2,x_3) \ :-\ x_3 \leftrightarrow (x_1 \wedge x_2).$$

This result can be obtained by size-rigidity analysis. It is worth noting that all the standard semantic properties are still valid in $Prop$, since $Prop$ is a constraint system. Therefore, given the abstract goal $G = \mathtt{append}(\{H,X\},\{Y\},\{H,Z\})$ (which abstracts $\mathtt{append}([H|X],Y,[H|Z])$), by Lemma 4.2 and Theorem 4.1 we obtain from the size-rigidity analysis: $\vee\mathcal{J}_{\alpha(P)}(G) = \{(X \wedge Y) \leftrightarrow Z\}$.

5.2. The Approximation Operator on Constraint Systems

A space of approximate constraints can be specified using upper closure operators on a domain of constraints [24]. This is justified by observing that (by extensivity) they map any constraint into a weaker one. In this section we discuss basic properties of upper closure operators on constraint systems such that the image of a constraint system under such an operator is also a constraint system. This provides a systematic way to construct the abstract operators of an abstract constraint system, given any such closure operator on the universe of the concrete constraint system. This class of closure operators includes those associated with any abstraction α that behaves as a morphism of constraint systems. As shown later in this section, this way of defining abstract constraint systems is applicable to many, but not all, abstractions. This limitation drives our interest in weaker constraint structures, discussed in Section 6.

We first observe that any upper closure approximation of a constraint system defines a partition of the universe of constraints into convex sets, i.e., if ρ is an upper closure on the universe of constraints $\mathcal{C}$, the set $\{c' \in \mathcal{C} \mid \rho(c) = \rho(c')\}$ is convex. As a consequence, the image of a universe of constraints $\mathcal{C}$ under a given upper closure operator ρ is a set of "abstract" constraints each representing a convex space of "concrete" solutions. However, in general, the abstract constraints so obtained may not satisfy the axioms for constraint systems: additional conditions have to be applied to ensure that they still provide a constraint system structure.

Definition 5.5.
Let $\mathcal{A}$ be a constraint system with universe $\mathcal{C}$, term system τ and set of variables V. An upper closure operator ρ on $\langle \mathcal{C}, \trianglelefteq \rangle$ is $\exists$-*consistent* if for each $c \in \mathcal{C}$ and $X \subseteq V$: $\rho(\exists_X c) = \exists_X \rho(\exists_X c)$. An upper closure operator ρ on $\langle \mathcal{C}, \trianglelefteq \rangle$ is ∂-*consistent* if for each $c \in \mathcal{C}$, $x \in V$ and $t \in \tau$ such that x *ind* t: $\rho(\partial_x^t c) = \partial_x^t \rho(\partial_x^t c)$.

$\exists$-consistency for a closure operator ensures that the approximation of a constraint where the variables in X are hidden, have the same set X hidden.From this condition we prove that ρ satisfies the similar condition of ∂-consistency, the $\exists$-quasi-morphism condition (see Lemma 5.2) and that $\rho \circ \exists_X$ is an upper closure operator.

Lemma 5.1.
$\rho \circ \exists_X$ is an upper closure operator.

Notice that $\exists_X \circ \rho$ is not idempotent, unless $\exists_X$ and ρ commute. This is in accordance with a classical result of the theory of closure operators saying that any composition of two upper closure operators is an upper closure operator iff they commute [62].

Lemma 5.2 (∂-consistency, $\exists$-quasi-morphism).
Let ρ be an $\exists$-consistent upper closure operator on the constraint system $\mathcal{A}$ with universe $\mathcal{C}$, term system τ and set of variables V. Then:

1. *for each $c \in \mathcal{C}$, $x \in V$ and $t \in \tau$ such that x ind t: $\rho(\partial_x^t c) = \partial_x^t \rho(\partial_x^t c)$; and*
2. *for each $c \in \mathcal{C}$, $X \subseteq V$: $\rho(\exists_X c) = \rho(\exists_X \rho(c))$.*

In the remainder of this section we discuss some conditions to systematically specify abstract constraint systems. This characterizes the class of abstract constraint systems (analyses) which can be systematically obtained as images of closure operators. As we will see, this program is not applicable to a number of abstract interpretations. This problem is addressed in Section 6.

Definition 5.6.
Let $\mathcal{A}$ be a constraint system with universe $\mathcal{C}$. A $\exists/\otimes$-*consistent upper closure operator* (*consistent* for short) ρ on $\mathcal{A}$ is an $\exists$-consistent upper closure operator on $\langle \mathcal{C}, \trianglelefteq \rangle$ that is a $\otimes$-quasi morphism, namely for each $c, c' \in \mathcal{C}$: $\rho(c \otimes c') = \rho(\rho(c) \otimes \rho(c'))$.

In addition to $\exists$-consistency, $\otimes$-quasi morphism relates meets of abstract constraints with meets of concrete constraints (recall that an upper closure operator is also a *quasi-complete join-morphism*, namely for each $C \subseteq \mathcal{C}$, $\rho(\sum C) = \rho(\sum \rho(C))$ [74]).

Lemma 5.3.
Let ρ be a consistent upper closure operator on the constraint system $\mathcal{A}$, with

universe $\mathcal{C}$, term system τ and set of variables V. Then for each $c \in \mathcal{C}$, $x \in V$ and $t \in \tau$ such that x ind t: $\rho(\partial_x^t c) = \rho(\partial_x^t \rho(c))$.

As observed in [24], any Galois insertion (α, γ) defines an upper closure operator $\rho = \gamma \circ \alpha$ on the corresponding (concrete) complete lattice. The following propositions provide some sufficient conditions for consistency of upper closures induced by a Galois insertion.

Proposition 5.6.
Let $\mathcal{A}$ and $\mathcal{A}^\sharp$ be constraint systems with universes $\mathcal{C}$ and $\mathcal{C}^\sharp$ respectively, such that $\mathcal{A}^\sharp$ is correct with respect to $\mathcal{A}$ by means of a surjective and additive semimorphism α. Let $\gamma : \mathcal{C}^\sharp \longrightarrow \mathcal{C}$ be defined as $\gamma(c^\sharp) = \sum\{c \mid \alpha(c) \trianglelefteq^\sharp c^\sharp\}$ and $\rho = \gamma \circ \alpha$. Then:

$\rho(\mathcal{C})$ is isomorphic to $\mathcal{C}^\sharp$;
2. if $\alpha(\exists_X c) = \exists^\sharp_{\kappa(X)} \alpha(\exists_X c)$ for every $X \subseteq V$ and $c \in \mathcal{C}$, then $\gamma \circ \alpha$ is $\exists$-consistent.

For example note that condition (2) is satisfied by the additive semimorphism associated with the abstract constraint system *Prop*.
The consistency of $\gamma \circ \alpha$ can be proved when α is actually a morphism of constraint systems.

Proposition 5.7.
Let $\mathcal{A}$ and $\mathcal{A}^\sharp$ be constraint systems with universes $\mathcal{C}$ and $\mathcal{C}^\sharp$ respectively, such that $\mathcal{A}^\sharp$ is correct with respect to $\mathcal{A}$ by means of a surjective and additive semimorphism α. Let $\gamma : \mathcal{C}^\sharp \longrightarrow \mathcal{C}$ be defined as $\gamma(c^\sharp) = \sum\{c \mid \alpha(c) \trianglelefteq^\sharp c^\sharp\}$ and $\rho = \gamma \circ \alpha$. Let $X \subseteq V$ and $c, c_1, c_2 \in \mathcal{C}$. If α_κ is a morphism on constraint systems then:

$\exists_X \rho(c) = \rho(\exists_X c)$; and
2. $\rho(\rho(c_1) \otimes \rho(c_2)) = \rho(c_1 \otimes c_2)$.

This result gives also a sufficient condition on $\mathcal{A}^\sharp$ such that the composition of $\exists$ and ρ is a closure, i.e., that $\exists$ and ρ commute.

Let $\mathcal{A} = (\mathcal{C}, \otimes, \oplus, \mathbf{1}, \mathbf{0}, \exists_X, d_{t_1,t_2})_{X \subseteq V; t_1, t_2 \in \tau}$ be a constraint system and ρ be an upper closure operator on $\mathcal{A}$. We define:

$$\rho(\mathcal{A}) = (\rho(\mathcal{C}), \tilde{\otimes}, \tilde{\oplus}, \mathbf{1}, \rho(\mathbf{0}), \rho \circ \exists_X, \rho(d_{t_1,t_2}))_{X \subseteq V; t_1, t_2 \in \tau}$$

where $\rho(\mathcal{C}) = \{c \in \mathcal{C} \mid c = \rho(c)\}$; $c_1 \tilde{\otimes} c_2 = \rho(c_1 \otimes c_2)$ for each $c_1, c_2 \in \rho(\mathcal{C})$; and and $\tilde{\oplus}$ is defined, for possibly infinite families $C \subseteq \mathcal{C}$, as: $\tilde{\sum} C = \rho(\sum C)$. In the following we denote by $\tilde{\partial}$ the induced substitution operator in $\rho(\mathcal{A})$.

Observation 5.2. It is worth noting that $\rho(\mathcal{A})$ corresponds (i.e., is isomorphic) to any structure of abstract constraints such that (α, γ) is a Galois insertion between the concrete and the abstract universe of constraints, $\rho = \gamma \circ \alpha$ and where the abstract operators of meet, join and cylindrification are defined as the corresponding best correct approximations *with respect to α and γ (see [24]), namely: $\lambda c_1, c_2.\alpha(\gamma(c_1) \otimes \gamma(c_2))$, $\lambda c_1, c_2.\alpha(\gamma(c_1) \oplus \gamma(c_2))$ and for any $X \subseteq V$*

$\lambda c.\alpha(\exists_X \gamma(c))$, respectively. However, $\tilde{\partial}$ may not correspond to the best approximation for substitution (i.e., $\rho \circ \partial$) unless ρ is consistent (see Lemma 5.4 below) or satisfies other properties (see Secion 6). However, for any closure ρ, $c \in \rho(\mathcal{C})$, $x \in V$ and $t \in \tau$ such that x ind t, it is easy to prove by extensivity that $\rho(\partial^t_x c) \trianglelefteq \tilde{\partial}^t_x c$.

Lemma 5.4. *Let ρ be a consistent upper closure operator on the constraint system $\mathcal{A}$, with universe $\mathcal{C}$, term system τ and set of variables V. Then for each $c \in \rho(\mathcal{C})$, $x \in V$ and $t \in \tau$ such that x ind t: $\tilde{\partial}^t_x c = \rho(\partial^t_x c)$.*

Theorem 5.3.
If ρ is a consistent upper closure operator on $\mathcal{A}$, then $\rho(\mathcal{A})$ is a constraint system.

By $\otimes/\oplus$-quasi-morphism and Lemmata 5.2, 5.3 and 5.4: $\rho(\mathcal{A})$ is correct with respect to $\mathcal{A}$ by means of the morphism ρ_{id}.

Example 5.3. Cylindrifications are monotonic operators, while idempotence and extensivity are specified by axioms C_4 and C_2 respectively. Moreover, cylindrifications commute, so if X and Y are sets of variables and c is a constraint: $\exists_X \exists_Y \exists_X c = \exists_X \exists_Y c$. However, for each set of variables X: $\exists_X$ is not a consistent upper closure operator on the constraint system because it does not satisfy the $\otimes$-quasi morphism condition (see Axiom C_3).

Example 5.4. Another example of non-consistent closure is given by the well known *interval approximation*. Consider the concrete constraint system $\mathcal{LR}_n$ in Example 4. An extensive operator on $\mathcal{LR}_n$ can be obtained by approximating any convex polyhedron with a *hypercube*, which is a polyhedron whose facets are parallel to the axes (similar techniques have been used for static array bound checking by interval approximation in [22]). For any set of polyhedra $c \in \mathcal{P}$, define $box(c)$ as the least hypercube containing c. box is clearly an upper closure operator on the domain of convex polyhedras ordered by set inclusion. It is worth noting that $box(\emptyset) = \emptyset$ and for each $X \subseteq V_n$: $box(\hat{\exists}_X c) = \hat{\exists}_X box(c)$, but box is not a $\otimes$-quasi morphism. A similar behaviour is shared by the *convex hull* operator in [27] combining convex polyhedra for linear restraint analysis. Both the interval and the convex polyhedron abstractions can be used to statically detect *future redundant constraints* in $CLP(\mathcal{R})$ computations (this problem has been studied in the context of compiler optimization in [48]). Intuitively, a constraint c in a clause is future redundant if, once c has been tested for satisfiability, it does not matter whether c is added to the constraint *store*, because the computation will inevitably add constraints stronger than c to the store. Here we sketch a formalization of this analysis as a non-standard CLP computation using a slightly different notion of redundancy. Consider the constraint system $\mathcal{LR}_n$ of Example 4. Let $P \in CLP(\mathcal{LR}_n)$ and ρ be any extensive operator on $\mathcal{LR}_n$. Assume p be a predicate symbol defined in P and let $C =$ '$p(\bar{t}) \ :- \ \widehat{c} \cap c' \ [\!] \ B$' $\in P$ be a clause defining p. Let $P' = (P \setminus \{C\}) \cup \{p(\bar{t}) \ :- \ c' \ [\!] \ B\}$. If $p(\bar{x}) \ :- \ c_p$ is in $\mathcal{F}^\flat(P')$, i.e., c_p is the answer constraint for p in the modified program, $c_p \cap \widehat{c} \neq \emptyset$ (i.e., $c_p \wedge \widehat{c}$ is solvable) and for each convex polyhedron $c \in c_p$: $\rho(c) \subseteq \widehat{c}$ (i.e., $\widehat{c}$ is weaker than $\rho(c)$), then $\widehat{c}$ is future redundant in C. To prove this claim we just note that by ρ-extensivity, for each constraint c: $c \subseteq \rho(c)$.

It is worth noting that the hypothesis that α is a morphism of constraint systems in Proposition 5.7, is often too strong for reasonable analyses (e.g., it is easy to see that the abstraction in *Prop* is not a morphism). More generally, when the concrete semantics is defined on constraint systems where $\otimes$ is idempotent and $\mathbf{1}$ is the annihilator for $\oplus$, any consistent abstraction becomes a $\otimes$-morphism. For this family of constraint systems, any meet of closed constraints is still closed: i.e., $\rho(c_1) \otimes \rho(c_2) = \rho(\rho(c_1) \otimes \rho(c_2))$. Therefore $\tilde{\otimes}$ is equivalent to $\otimes$ in $\rho(\mathcal{A})$.

Theorem 5.4.
Let ρ be a consistent upper closure operator for a constraint system $\mathcal{A}$ with universe of constraints $\mathcal{C}$ and let $c_1, c_2 \in \mathcal{C}$. Suppose that $\rho(c_1) \otimes \rho(c_2) \trianglelefteq \rho(c_1 \otimes c_2)$. If $\mathcal{A}$ is $\otimes$-idempotent and $\mathbf{1}$ is the annihilator for $\oplus$, then $\rho(c_1 \otimes c_2) = \rho(c_1) \otimes \rho(c_2)$.

The behavior of consistent closures is too restrictive for most of the abstract interpretations, where the intended *meet approximation* does not support the $\otimes$-quasi morphism condition. However weakening consistency may result in a structure of constraints $\rho(\mathcal{A})$ that is not, in general, a constraint system. Therefore, more general abstractions require a weaker notion of constraint system. In the following section we consider $\otimes$-idempotent constraint systems where $\mathbf{1}$ is annihilator for $\oplus$, as these conditions are satisfied in most "concrete" constraint systems, e.g., $CLP(\mathcal{H})$. These structures turn out to be distributive lattices [38].[3]

6. NON-DISTRIBUTIVE CONSTRAINT SYSTEMS

In this section we discuss the impact of different closure operators (abstractions) on the general properties of constraint systems. Let $\mathcal{A}$ be a constraint system. By a quick inspection of Theorem 5.3 we can observe that, for any upper closure operator ρ, axioms R_1, R_2, R_4 of Definition 3 and C_2, D_1 and D_2 of Definition 5 are satisfied in $\rho(\mathcal{A})$. Therefore, we identify the remaining axioms: R_3, R_5 (of Definition 3), C_1, C_3, C_4, C_5, D_3 and D_4 (of Definition 5) as those possibly affected by a generic abstraction (later we abuse terminology by referring to these as the *distributivity laws*). A *non-distributive constraint system* with universe $\mathcal{C}$, term system τ and set of variables V is then a structure similar to a constraint system, as defined in Definition 5, except that the distributivity laws are replaced by the following respectively, where $c, c' \in \mathcal{C}$, $C \subseteq \mathcal{C}$, $t, t', t'' \in \tau$ and $\{x\}, X \subseteq V$ such that x *ind* t:

[3] Commutativity of $\otimes$ is not needed to show that $\langle \mathcal{C}, \trianglelefteq, \mathbf{0}, \mathbf{1}, \oplus, \otimes \rangle$ is a lattice, this being a consequence of R_1, R_2, R_3, R_5, $\otimes$-idempotence and annihilation for $\mathbf{1}$. In particular it is possible to prove from these hypotheses that $a \otimes b$ and $b \otimes a$ are both the greatest lower bounds of a and b, whence, by uniqueness, $\otimes$ is commutative [52]. This extends the result in [38] which requires commutativity of $\otimes$.

R_3. $\mathbf{0} \trianglelefteq \mathbf{0} \otimes c$

R_5. $c \otimes (\sum C) \trianglelefteq \sum\{c \otimes c' \mid c' \in C\}$

C_1. $\mathbf{0} \trianglelefteq \exists_X \mathbf{0}$

C_3. $\exists_X (c \otimes \exists_X c') \trianglelefteq \exists_X c \otimes \exists_X c'$

C_4. $\exists_{X \cup Y} c \trianglelefteq \exists_X \exists_Y c$

C_5. $\sum\{\exists_X c' \mid c' \in C\} \trianglelefteq \exists_X (\sum C)$

D_3. $d_{[t/x]t',[t/x]t''} \trianglelefteq \partial_x^t (d_{t',t''})$;

D_4. $\partial_x^t (c \otimes c') \trianglelefteq \partial_x^t c \otimes \partial_x^t c'$.

In the following we identify a set of reasonable restrictions for a generic upper closure operator ρ on a constraint system $\mathcal{A}$. They provide a characterization for the analyses that can be captured in some non-distributive constraint system. We list them below, each one provided with the set of non-distributive laws satisfied in $\rho(\mathcal{A})$. In the following, $t, t' \in \tau$, $x \in V$ and $x\ ind\ t$. The proofs of the following claims can be easily derived by inspection of the proof of Theorem 6.1 below.

P_1. $\underline{\rho(\mathbf{0}) = \mathbf{0}}$. The abstraction of inconsistent constraints is still inconsistent. This extends the *consistency check* from concrete to abstract computations. It is applied in common analysis such as: *Prop*, linear equalities (see Section 6.1 and [49]), inequalities (see [27]), $\exists$-approximation and interval approximations in Example 4, *etc.* The constraint system $\rho(\mathcal{A})$ is R_5, C_4, C_5, D_3 and D_4 non-distributive, but C_3 does not hold.

P_2. $\underline{\rho(d_{t,t'}) = d_{t,t'}}$. Diagonal elements are invariant under abstraction. This is a typical assumption in static analysis by approximating numerical relations between variables of a program, such as: interval approximation, linear equalities and inequalities. The constraint system $\rho(\mathcal{A})$ is R_3, R_5, C_1, C_4, C_5 and D_4 non-distributive, but C_3 does not hold.

P_3. $\underline{\rho \text{ is additive}}$. The universe of abstract constraints is isomorphic to a sublattice of the concrete one. An additive closure can be obtained by lifting the abstraction on the powerset (see [26]). This provides a more precise interpretation for disjunction. The $\exists$-approximation in Example 4 is additive. The constraint system $\rho(\mathcal{A})$ is R_3, C_1, C_4, D_3 and D_4 non-distributive, but C_3 does not hold.

Axiom C_3 can be satisfied, in its distributive or non-distributive form, provided that one of the following *existential conditions* is verified:

E_1. $\underline{\rho \circ \exists = \exists \circ \rho \circ \exists}$. This is the $\exists$-consistency condition in Definition 5. The constraint system $\rho(\mathcal{A})$ is R_3, R_5, C_3, D_3 and D_4 non-distributive.

E_2. $\underline{\exists \circ \rho = \rho \circ \exists \circ \rho}$. $\exists$ preserves the closure, i.e., existentially quantified closed constraints are still closed. In particular, $\exists \circ \rho$ is a closure operator. This condition is satisfied in numerical abstract domains of constraints such as linear equalities and interval analysis. The constraint system $\rho(\mathcal{A})$ is R_3, R_5, C_1, C_5, D_3 and D_4 non-distributive.

E_3. $\underline{\exists \circ \rho = \rho \circ \exists}$. This condition is true iff both E_1 and E_2 are true. In this case, both $\rho \circ \exists$ and $\exists \circ \rho$ are closure operators [62]. This property is shared by most of the well known abstractions such as: *Prop*, linear equalities and inequalities, $\exists$ and interval abstractions, *etc.* The constraint system $\rho(\mathcal{A})$ is R_3, R_5, D_3 and D_4 non-distributive.

Properties P_1–P_3 can be combined with the existential conditions E_1–E_3 in order to satisfy more distributivity laws. For example, we notice that for any upper closure operator ρ satisfying P_2 and E_1, $\rho(\mathcal{A})$ is only R_3, R_5 and C_3 non-distributive, while if it satisfies P_2 and E_3 then it is only R_3 and R_5 non-distributive. The following section shows an application of such closure operators to dataflow analysis of CLP programs.

Theorem 6.1.

Let $\mathcal{A}$ be a constraint system with universe $\mathcal{C}$, variables V and term system τ. If ρ is an upper closure operator satisfying any of the existential conditions E_1–E_3 and a (possibly empty) combination of properties P_1–P_3, then $\rho(\mathcal{A})$ is a non-distributive constraint system.

It is worth noting that, under the hypothesis of the previous theorem, $\rho(\mathcal{A})$ is always distributive in C_4. The following example shows that we can prove the D_3 distributivity, by combining P_1 and E_3 with a particular definition for diagonal elements (namely the best corresponding approximation).

Example 6.1. In this example we sketch the systematic derivation of abstract constraint systems from a given data-abstraction. This corresponds to generate the best approximating operator (see [24]) for each basic operator in the constraint system, including diagonal elements. As we will see, this abstraction reduces the loss of distributivity in the abstract constraint system. In the following we assume that $\mathcal{A}$ is a $\otimes$ idempotent (distributive) constraint system on the term system τ with dimension α: $\mathcal{A} = (\mathcal{C}, \otimes, \oplus, \mathbf{1}, \mathbf{0}, \exists_X, d_{t_1,t_2})_{X \subseteq V; t_1,t_2 \in \tau}$ where $\mathbf{1}$ is an annihilator for $\oplus$.

Term systems:

let τ^a be a set of objects including a set $\mathcal{V}$ such that $|\mathcal{V}| = |V| = \alpha$. Let $\kappa : \tau \longrightarrow \tau^a$ be a surjective function such that, by defining $\delta : \tau^a \longrightarrow \wp(\tau)$ as $\delta(a) = \{t \in \tau \mid \kappa(t) = a\}$ for $a \in \tau^a$, then: $\kappa \circ \delta(a) = a$ and $\{t\} \subseteq \delta \circ \kappa(\{t\})$ for $a \in \tau^a$ and $t \in \tau$. Assume also that κ satisfies the following conditions on the structure of the constraint system τ: $\kappa(V) = \mathcal{V}$ (therefore V and $\mathcal{V}$ are isomorphic by κ), and for each $t, t_1, t_2, t_3 \in \tau$, $x, y \in V$: $\kappa(s_x(t, t_1)) = \kappa(s_y(t_2, t_3))$ when $\kappa(x) = \kappa(y)$, $\kappa(t) = \kappa(t_2)$ and $\kappa(t_1) = \kappa(t_3)$, namely substitution is *compatible* with the equivalence relation induced by κ.

For any $y \in \mathcal{V}$, $a, b \in \tau^a$, define $s^a_y(a, b) = \kappa(s_x(t, t_1))$ where $\kappa(x) = y$, $t \in \delta(a)$ and $t_1 \in \delta(b)$. From the previous hypothesis it is easy to see that s^a is well defined on τ^a, and

Proposition 6.1.

$(\tau^a, S^a, \mathcal{V})$ is a term system of dimension α, and κ is a morphism from τ into τ^a.

Constraint systems:

let $\langle \mathcal{X}, \leq, \top, \bot, \vee, \wedge \rangle$ be a complete lattice containing a set of objects $d_{a,b}$ for $a, b \in \tau^a$. Assume (α, γ) be a Galois insertion of $\langle \mathcal{X}, \leq \rangle$ into $\langle \mathcal{C}, \trianglelefteq \rangle$ such that $\gamma(\bot) = \mathbf{0}$. We also assume that: $\alpha(d_{t,t'}) = d_{\kappa(t),\kappa(t')}$ and $\gamma(d_{a,b}) = \oplus\{d_{t,t'} \mid \kappa(t) = a \textit{ and } \kappa(t') = b\}$. This corresponds to require that (α, γ) is also a Galois insertion between the corresponding sublattices generated by the diagonal elements. Finally, we assume E_3 as existential condition for $\gamma \circ \alpha$.

It is straightforward, from the previous hypothesis, that $\gamma \circ \alpha$ is an upper closure on $\mathcal{C}$ satisfying condition P_1. However, the abstract constraint system $\rho(\mathcal{A})$ is only R_5 and D_4 non-distributive, namely we can prove D_3 distributivity, which cannot be derived from P_1 and E_3 only.

Proposition 6.2.
$\rho(\mathcal{A})$ is a correct R_5 and D_4 non-distributive constraint system.

We conclude, from the previous proposition, that the behaviour of abstract diagonal elements with respect to substitution is preserved when they are derived systematically from the abstract term system. Here, the construction of abstract diagonal elements helps in proving an important distributive property.

6.1. Non-distributive Analysis: Linear Relationships

This section considers a quite common form of non-distributivity for constraint systems and applies it to the problem of inferring linear size relationships between the arguments of procedures. We consider a constraint system $\mathcal{A}$ with universe $\mathcal{C}$, where only axiom R_5 of Definition 3 is replaced by the weaker relation: $c \otimes (\sum C) \trianglelefteq \sum\{c \otimes c' \mid c' \in C\}$ for $c \in \mathcal{C}$ and $C \subseteq \mathcal{C}$. We abuse terminology by referring to these systems as *non-distributive*. Axiom R_5 is needed to prove the continuity of $T_P^\flat$, with the eventual objective of showing the equivalence of the fixed-point and operational semantics. However the weaker property of monotonicity can be proved for any non-distributive constraint system. The following proposition follows from the monotonicity of $\exists$ and $\otimes$.

Proposition 6.3.
$T_P^\flat$ is monotonic in any non-distributive constraint system $\mathcal{A}$.

It follows that for Noetherian non-distributive constraint systems, $T_P^\flat$ is also continuous. Moreover, as far as equivalence of semantics is concerned, the operational semantics is, in some sense, an "all solutions" semantics where the join is taken at the end of all the possible computations; in the fixed-point case, by contrast, the join operator is applied at each partial computation step (an equivalent operational semantics can be easily defined: this would correspond to the bottom-up execution strategy of deductive databases rather than the standard operational interpretation of logic programs [54]). In this case, as the constraint system is not distributive any more, we can only have a further approximation level by applying bottom-up instead of top-down, i.e., $(\mathcal{O}(P))^\flat \sqsubseteq \mathcal{F}^\flat(P)$. This behavior was already observed by Jacobs and Langen [44] in the analysis of pure logic programs with *condensing*. In the following we study this class of constraint systems by means of an example: the *linear relationship analysis*.

The linear relationship analysis is useful for a variety of applications such as compile-time overflow detection, integer subrange checking, array bound checking, termination analysis, *etc.*, has been considered by a number of researchers (e.g., [27, 49, 72, 73]). The approach of Verschaetse and De Schreye [73] for automatic inference of linear size relations among variables in logic programs can be specified as a constraint computation in our framework.

Let $\tau_{(\Sigma,V)}$ be defined as in Example 1, over a finite set of variables V. Let $|\cdot|_S$ be a norm on the term system $\tau_{(\Sigma,V)}$. We define a term system τ_{Exp} of linear expressions where terms are first order terms in the language $\{+,0,1,V\}$ (i.e., terms in $\tau_{(\{+,0,1\},V)}$). Since we are interested only in relations having finite arity, we can always represent any answer constraint as a constraint on the finite dimensional space of its free variables. Moreover, the use of a bottom-up semantics construction does not require any infinite set of variables for renamings. Therefore, the set of variables V can be assumed to be a finite set $V_n = \{x_1, ..., x_n\}$. Substitutions are performed as standard substitutions. In the following, if $f(t_1, ..., t_n)$ is a term, then $t_1, ..., t_n$ are its 1-subterms.

Proposition 6.4.
τ_{Exp} is a term system.

The mapping $Exp_S : \tau_{(\Sigma,V)} \to \tau_{Exp}$ associates a linear expression with each term in $\tau_{(\Sigma,V)}$, as follows: let t be a term and S_t be the set of *selectors* for the "relevant" subterms of t, i.e., $s \in S_t$ iff $s(t)$ is a 1-subterm of t and $s(t)$ is not rigid.

$$Exp_S(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ c_0 + \sum_{s \in S_t} Exp_S(s(t)) & \text{otherwise} \end{cases}$$

It is straightforward to prove that Exp_S is a morphism.

Example 6.2. With *length* and *size* norms we have: $Exp_{length}([X[a|Z]]) = 1 + 1 + Z$ and $Exp_{size}([X[a|Z]]) = 1 + X + 1 + Z$ respectively.

Karr [49] shows that size relations among variables in a program can be obtained by manipulating *affine relationships* i.e., linear equalities of the form $c_0 = c_1X_1 + ... + c_nX_n$. In our framework, this corresponds to a constraint system as follows: let $\mathcal{L}$ be the set of *affine subspaces* corresponding to linear equalities on a fixed n-dimensional space (e.g., $\Re^n$); the universe of constraints is $\wp(\mathcal{L})$; the meet operation $\otimes$ is simply intersection of affine subspaces; the join operation is set union; cylindrification, which corresponds to the variable restriction of Verschaetse and De Schreye, corresponds to "projection" parallel to an axis, which maps a set of affine subspaces into a set of affine subspaces; let S be a set of affine subspaces and $x \in V$, $t \in \tau_{Exp}$, then the substitution of x with t in S is the affine subspace $\hat{\exists}_{\{x\}}([x = t] \cap S)$. The elements $\mathbf{0}$ and $\mathbf{1}$ are defined as $\emptyset$ and the entire space $\Re^n$ respectively. Diagonal elements are (single) equations on the term system τ_{Exp}. As usual, for each equation $t_1 = t_2$, we denote by $[t_1 = t_2] \subseteq \Re^n$ the corresponding affine subspace.

Proposition 6.5.
$Rel = (\wp(\mathcal{L}), \cap, \cup, \Re^n, \emptyset, \exists_X, [t = t'])_{X \subseteq V_n; t,t' \in \tau_{Exp}}$ is a constraint system.

Given a norm S, the abstraction function α_S can be defined by extending Exp_S, similarly to that of Section 5.1, therefore Rel is correct with respect to $\mathcal{H}$. Note however that Rel is not Noetherian, therefore it is not directly applicable for static analysis of $CLP(\mathcal{H})$ programs.

The approximation introduced in Karr [49] corresponds precisely to the abstraction of Rel given by an upper closure operator ρ_{aff}, mapping any set of points into the smallest affine subspace containing them. It is immediate to observe that: $\rho_{aff}(\emptyset) = \emptyset$, $\exists \circ \rho_{aff} = \rho_{aff} \circ \exists$ (in particular: cylindrification maps affine subspaces into affine subspaces) and $\rho_{aff}(\{c\}) = c$ for $c \in \mathcal{L}$ (i.e., diagonal elements are not affected by abstraction). Therefore, ρ_{aff} satisfies P_1, P_2 and E_3, and $\rho_{aff}(Rel)$ is a R_5 non-distributive constraint system, which is correct with respect to $\mathcal{H}$. The join of two affine subspaces A_1 and A_2, given by $\rho_{aff}(A_1 \cup A_2)$, is here the smallest affine subspace containing A_1 and A_2 (since the union of two affine subspaces is not, in general, an affine subspace). To prove that $\rho_{aff}(Rel)$ is a non-distributive closed semiring we observe that $(x_1 = 0.5, x_3 = 0.5 + x_2) \cap \rho_{aff}((x_1 = 0, x_2 = x_3) \cup (x_1 = 1, x_3 = 1 + x_2)) = (x_1 = 0.5, x_3 = 0.5 + x_2)$ while $(x_1 = 0.5, x_3 = 0.5 + x_2) \cap (x_1 = 0, x_2 = x_3) = \emptyset$ and $(x_1 = 0.5, x_3 = 0.5 + x_2) \cap (x_1 = 1, x_3 = 1 + x_2) = \emptyset$. As pointed out in [49], there are no infinitely ascending chains of free-variable bounded constraints in $\rho_{aff}(Rel)$ (i.e., bounded dimension affine spaces), otherwise in any properly ascending chain of subspaces $U_1 \trianglelefteq U_2 \trianglelefteq \ldots$, the subspaces U_i must have a dimension of at least one greater than U_{i-1}. $\rho_{aff}(Rel)$ is therefore Noetherian.

Example 6.3. Consider the logic program defining the predicate *append* in Example 2, together with the norm *length*. The corresponding abstract program and semantics are:

$$\begin{array}{l} append(x_1, x_2, x_3) \;:-\; x_1 = 0, x_2 = x_3. \\ append(x_1, x_2, x_3) \;:-\; x_1 = 1 + y, x_3 = 1 + z \;[\!]\; append(y, x_2, z). \end{array}$$

$$\begin{array}{lcl} T_P^\flat \uparrow 0(\emptyset) & = & \emptyset \\ T_P^\flat \uparrow 1(\emptyset) & = & \{append(x_1, x_2, x_3) \;:-\; x_1 = 0, x_2 = x_3\} \\ T_P^\flat \uparrow 2(\emptyset) & = & \{append(x_1, x_2, x_3) \;:-\; \rho_{aff}((x_1 = 0, x_2 = x_3) \cup \\ & & \qquad (x_1 = 1, x_3 = 1 + x_2))\} \\ & = & \{append(x_1, x_2, x_3) \;:-\; x_1 + x_2 = x_3\} \end{array}$$

The affine subspace $x_1 + x_2 = x_3$ specifies the relationship among the lengths of the arguments of the predicate *append* in the expected way. For example, a solution for the length of the tail X in the goal $\mathtt{append}([H|X], [d|e|f], [a|b|c|d|e|f])$ can be found by solving the corresponding abstract goal $\mathtt{append}(1 + X, 3, 6)$, resulting in $X = 2$. A possible implementation can be obtained by slightly modifying the $CLP(\mathcal{R})$ interpreter in [47] to cope with affine relations. This corresponds to implement (at the meta level) the join operator for affine subspaces so as to combine the computed answer constraints generated by the interpreter. Thus, abstract interpretation for linear size relationships can be joined to a concrete interpretation on $CLP(\mathcal{R})$ of a modified program.

7. DISCUSSION AND RELATED WORKS

Our definition of constraint systems was motivated by earlier work of Debray and Ramakrishnan [29], which gives an algebraic formulation for standard and non-standard semantics of logic programs, but over a very different algebraic structure. In particular, we modify closed semirings (already used in [29]) to cope with constraint-like objects including cylindrification of constraints, and diagonal elements as atomic constraints. This provides a direct definition, at the constraint system level, for standard semantic notions like variable projection and unification.

Our definition of constraint logic programs is close to the original one of Jaffar and Lassez [45]. We generalize the notion of constraint system so as to apply it to possibly non-standard (e.g. abstract) interpretations. We follow [45] by defining parameter passing as generic term equations, and we generalize this notion to any possibly non-standard term system. This corresponds precisely to generalize CLP including non-standard objects (trace sequences, abstract constraints *etc.*) as constraints. With respect to [45], we also give an algebraic presentation for constraint systems, involving variable projection. This approach is more suitable to reason about abstract interpretation, in particular when studying closure operators on constraint systems (e.g. see Section 5.2 and Section 6).

Saraswat *et al.* define the semantics of concurrent constraint languages in terms of cylindric algebras [64], specifying constraint systems in the style of Scott's information systems [65] via a set of "primitive" constraints $\mathcal{C}$ and an entailment relation $\vdash \subseteq \wp(\mathcal{C}) \times \mathcal{C}$. Composition of constraints is defined in terms of set union, hiding in terms of cylindrification, and parameter passing using diagonal elements. There is a fundamental difference between our work and that of Saraswat *et al.* in the underlying algebraic structure. Information systems are general structures where the primary role of entailment provides a very convenient mechanism for modelling *blocking-ask* synchronization in concurrent constraint programming languages [64]. By contrast, we are interested less with entailment as a primitive notion, than with identifying algebraic structures that make it easier to generalize the standard semantic results for constraint logic programming. In our case, the constraint system is based on closed semirings and is parametric with respect to a given term system (it is easy to associate an information system with a closed semiring[4] $(\mathcal{C}, \otimes, \oplus, \mathbf{1}, \mathbf{0})$ if $\otimes$ is commutative and idempotent, but of course this may not hold in general). This makes it possible to define non-standard constraint systems, e.g., for abstract interpretation, in a simpler and more structured way. In our opinion, it is easier to specify standard logical and arithmetic operators as an instance of a closed semiring than as an instance of an information system, making closed semirings a more natural basis for generalizing constraint systems to deal with standard and non-standard semantics. This is because the lattice-structure of usual abstract domains provides a suitable abstract interpretation for $\otimes$, $\oplus$, *etc.* (see $Prop$ or the affine relationship analysis). Moreover, the join operator can often be interpreted as a widening of constraints, and this can be easily characterized in a closed semiring structure. More recently, the framework of concurrent constraint programming has been extended to cope with function symbols (terms). Technically this is handled by a *hyperdoctrinal* account of existential and diagonal notions in [63]. However, the use of hyperdoctrines in the context of the present work does not add significant results, if compared with the classic, and somehow more standard, treatment of cylindric algebras by Cirulis [13].

The idea of generalized semantics has been recently applied to the family of *cc*

[4] An interesting work on the relation between the Scott's topology and a topology for closed semirings is in [51].

languages by Zaffanella *et al.* in [75]. The extension of our framework to *cc* is not straightforward, as we cannot (in general) provide a correct approximation of program's behaviour by abstractly evaluating abstract versions of *cc* programs. This is a consequence of synchronization based on *blocking ask*. Intuitively a correct approximation of the program meaning generates weaker answers for any possible program behaviour. Therefore, in order to correctly characterize answers associated with suspended computations, we must guarantee that whenever a concrete computation suspends the corresponding abstract computation suspends too. This can only be obtained by replacing *ask* constraints with stronger constraints, which is usually not the case in abstract interpretation. Some solutions to this problem are addressed in [75].

Abstract interpretation of (sequential) constraint logic programs was considered firstly by Marriott and Søndergaard [57]. Their treatment is based on abstracting a denotational semantics for constraint logic programs. A meta-language based on the typed λ-calculus is used to specify the semantics of logic languages in a denotational style, and both the standard and non-standard semantics are viewed as instances of the meta-language specification. In our case, instead of defining a meta-language for dataflow analysis, we consider the constraint specification on which the CLP paradigm is defined. Non-standard semantics for a given constraint-based program can thus be obtained simply by appropriately modifying the underlying constraint system. This gives a formal account for abstract compilation, which is a quite standard technique in dataflow analysis implementation [68], as constraint-based computation.

A related approach is also considered by Codognet and Filè, who firstly give an algebraic definition of constraint systems and consider abstract interpretation of constraint logic programs as constraint abstraction [19]. However, the algebraic structure considered by these authors is very different: only $\otimes$-composition is considered, and while a notion of "computation system" is introduced, the underlying structure is not provided with a join operator. Because of this construction, mainly based on a generalization of the top-down SLD semantics, they cannot characterize, at the constraint level, the "condensing" of multiple solutions, which is very useful in abstract interpretation (e.g., see $Prop$ and the linear relationships analysis). Thus, by applying a loop-checker consisting in a "tabled" interpreter, only finite abstract domains can be handled. In our framework, by contrast, extraneous devices such as loop checking and tabulation are not considered. Instead, finiteness is treated simply as a property of the constraint system, expressed in terms of $\trianglelefteq$-chains. This allows non-standard computations to be specified as standard CLP computations over an appropriate (possibly Noetherian) constraint system (e.g., affine subspaces represent a suitable abstraction for linear relationships, providing an approximation that is inherently Noetherian but is not finite). Moreover, both the traditional top-down and bottom-up semantics can be specified in the standard way thus providing goal-independent static analysis of CLP programs.

Recently, constraint programming techniques have been applied to the abstract interpretation of Prolog programs. In [20] a new language: *Toupie* is introduced to compile the abstract semantics of Prolog into a constraint based language, where constraints over finite domains are implemented as decision diagrams. In [17], an efficient implementation of ground dependency analysis is obtained by implementing the constraint solver for propositional formulae as a Datalog program, as suggested earlier by Dart [28]. While the approach does not encode disjunction of propositional formulae, it provides a simple and powerful tool for static analysis of groundness in Prolog. Magic-like transformations are applied to get call patterns.

It should be noted here that while the framework described can describe a wide variety of program analyses, there are some kinds of analyses that it cannot express. Specifically, it cannot capture analyses where the join operator $\oplus$ is not commutative, since this would violate the axioms of closed semirings. Non-commutative join operators may be found in analyses that model the depth-first execution strategy of Prolog (e.g. see [6]). It is also interesting to observe that $\oplus$ idempotence is in contrast with the typical multiplicity of solutions for a Prolog-like system. Weaker structures can be studied for these cases.

At the constraint system level, we abstract a system of constraints which actually contains the standard (logical) interpretation of constraints: i.e., constraints as lower-closed sets of formulae, and where the approximation order is the same as entailment. This is of course a restriction, and weaker constraint systems and abstractions can be studied. In particular, by dropping axiom C_2, we can obtain a weaker structure which can be instantiated with (possibly non lower-closed) powerset constructions. This may be useful to associate (at the constraint system level) the set of possible computed answer constraints with each predicate[5]. In this case, it is easy to see that the computed answer constraint semantics and and the condensed one coincide. However, we believe that axiom C_2 is essential for a "logical" interpretation of constraints and hiding. This is a key-point in our approach to abstract interpretation of constraint logic programs, where data-flow analysis is computed in a CLP-like way. This task is obtained by requiring that both concrete and abstract constraints share a similar "logical" interpretation. This restriction allows us to join some abstract domain with suitable constraint systems. As shown in Section 4, more concrete observable behaviours (e.g., the set of computed answer constraints) can be obtained at the semantic level, by applying different semantic constructions (e.g., see the semantics in Section 4 modeling sets of computed answer constraints, without condensing). However, notice that some observable properties which are different from success patterns, such as: *failure*, *call patterns* and *partial answers* cannot be modeled by applying directly the semantic construction in Section 4. As for call patterns, both the *magic*-like transformation in [15] and the semantic-based approximation in [32] can be easily extended to CLP languages in view of the present paper. The machinery of partial answers instead may require an additional layer of abstraction, like the one applied in [16] for the compositional analysis of modular logic programs. We believe that our constraint system notion and abstraction can be easily applied to semantic constructions characterizing different observable behaviours, like those described in [10].

8. CONCLUSIONS

We have defined an algebraic framework for a generalized semantics for constraint logic programs. Such an approach to program semantics allows a formal treatment for correctness conditions in any non-standard interpretation, e.g., for abstract interpretation, or reasoning about compiler correctness, and provides a basis for the study of the general algebraic properties of the semantics construction. The

[5] We thank an anonymous referee for this comment.

ability to represent the condensing process as an operator in the constraint system simplifies the abstract semantic construction, and provides a formal axiomatic treatment of abstraction. Moreover, the use of variable hiding operators (such as cylindrifications) in the T_P definition allows the use of finite dimension constraint systems and provides a formal treatment of renaming in abstract interpretation. Finite dimension constraint systems are particularly useful to provide finite upper approximations to the semantics, such as in the case of linear relationships analysis, where the finiteness is strongly related with the (finite) dimension of the space of solutions.

Further generalizations are possible in view of abstract interpretation. Weaker constraint systems can be considered, where for example distributivity does not hold. The distributivity restriction is not applicable to a wide class of static analysis problems including linear relationships, as shown in Section 6.1, and *range variable analysis*, based on an abstract lattice of intervals specifying the range of program variables [3]. Non-distributive constraint systems can be studied as a more general framework for constraint-based program analysis. A classification of the different constraint systems which are useful in dataflow analysis can be based on the set of properties they hold. A comparison with our framework can be helpful to systematically derive those properties of the semantics construction that may be affected by a different constraint system definition. Moreover, the notion of abstraction can be refined even more by considering semimorphisms of term systems, where terms are ordered by instantiation. This can be suitable to characterize term abstraction in abstract interpretation.

ACKNOWLEDGMENTS

The stimulating discussions with Roberto Bagnara, Roberto Barbuti, Veroniek Dumortier, Maurizio Gabbrielli, Georg Karner, Michael Maher, Nino Salibra, and Gert Smolka are gratefully acknowledged. We thank the anonymous referees for many helpful comments.

REFERENCES

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison Wesley Publishing Company, 1974.

2. K. R. Apt. Introduction to Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 495–574. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

3. R. Bagnara, R. Giacobazzi, and G. Levi. Static Analysis of CLP Programs over Numeric Domains. In *Actes Workshop on Static Analysis, WSA'92*, number 81-82 in Bigre, pages 43–50, 1992.

4. R. Bagnara, R. Giacobazzi, and G. Levi. An Application of Constraint Propagation to Data-flow Analysis. In *Proc of Ninth IEEE Conference on AI Applications*, pages 270–276. IEEE Computer Society Press, 1993.

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. R. Barbuti, M. Codish, R. Giacobazzi, and G. Levi. Modeling Prolog Control. *Proc. Nineteenth ACM Symposium on Principles of Programming Languages*, Jan. 1992, pages 95–104. ACM Press.

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

8. G. Birkhoff. Lattice Theory. In *AMS Colloquium Publication, third ed.*, 1967.

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

10. A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics approach: theory and applications. *Journal of Logic Programming*, 19 & 20:149–197, 1994.

11. M. Bruynooghe. A Practical Framework for the Abstract Interpretations of Logic Programs. *Journal of Logic Programming*, 10:91–124, 1991.

12. M. Bruynooghe, G. Janssens, B. Demoen, and A. Callebaut. Abstract Interpretation: Towards the Global Optimization of Prolog Programs. In *Proc. Fourth IEEE International Symp. on Logic Programming*, pages 192–204. IEEE Comp. Soc. Press, 1987.

13. J. Cirulis. An Algebraization of First Order Logic with Terms. *Colloquia Mathematica Societatis Jànos Bolyai*, 54:125–146, 1991.

14. K. L. Clark. Predicate logic as a computational formalism. Technical Report Dept. of Computing, Imperial College, 1979.

15. M. Codish, D. Dams, and E. Yardeni. Bottom-up Abstract Interpretation of Logic Programs. *Theoretical Computer Science*, 124(1):93–126, 1994.

16. M. Codish, S. K. Debray, and R. Giacobazzi. Compositional Analysis of Modular Logic Programs. In *Proc. Twentieth Annual ACM Symp. on Principles of Programming Languages*, pages 451–464. ACM Press, 1993.

17. M. Codish and B. Demoen. Analysing logic programs using "`prop`"-ositional logic programs and a magic wand. In D. Miller editor, *Proc. of the 1993 International Logic Programming Symposium*, pages 114–129, MIT Press 1993.

18. M. Codish, M. Falaschi, and K. Marriott. Suspension Analyses for Concurrent Logic Programs. Technical Report TR 12/92, Dipartimento di Informatica, Università di Pisa, 1992. To appear in *ACM Transactions on Programming Languages and Systems*.

19. P. Codognet and G. Filè. Computations, Abstractions and Constraints. In *Proc. IEEE International Conference on Computer Languages*, ICCL'92, IEEE Press, 1992.

20. M.-M. Corsini, K. Musumbu, A. Rauzy, and B. Le Charlier. Efficient bottom-up abstract interpretation of Prolog by means of constraint solving over symbolic finite domains. In M. Bruynooghe and J. Penjam, editors, *Programming Language Implementation and Logic Programming* - Proceedings PLILP'93, volume 714 of *Lecture Notes in Computer Science*, pages 75–91. Springer-Verlag, Berlin, 1991.

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

22. 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.

23. P. Cousot and R. Cousot. A constructive characterization of the lattices of all retracts, pre-closure, quasi-closure and closure operators on a complete lattice. *Portugaliæ Mathematica*, 38(2):185–198, 1979.

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

25. P. Cousot and R. Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In M. Bruynooghe and M. Wirsing, editors, *Proc. of PLILP'92*, volume 631 of *Lecture Notes in Computer Science*, pages 269–295. Springer-Verlag, Berlin, 1992 (To appear in *Acta Informatica*).

26. P. Cousot and R. Cousot. Abstract Interpretation and Applications to Logic Programs. *Journal of Logic Programming*, 13(2 & 3):103–179, 1992.

27. 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.

28. P. Dart. On Derived Dependencies and Connected Databases. *Journal of Logic Programming*, 11(2):163–188, 1991.

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

30. S. Eilenberg. *Automata, Languages, and Machines.* Academic Press, volume A, 1974.

31. 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.

32. M. Gabbrielli and R. Giacobazzi. Goal Independency and Call Patterns in the Analysis of Logic Programs. In E. Deaton, D. Oppenheim, J. Urban and H. Berghel editors, Proc. of the *Ninth ACM Symposium on Applied Computing*, pages 394–399, ACM Press, Phoenix AZ 1994.

33. M. Gabbrielli and G. Levi. Modeling Answer Constraints in Constraint Logic Programs. In K. Furukawa, editor, *Proc. Eighth International Conference on Logic Programming*, pages 238– 252. The MIT Press, Cambridge, Mass., 1991.

34. R. Giacobazzi. *Semantic Aspects of Logic Program Analysis.* Ph.D. Dissertation, Universitá di Pisa, March 1993. Also available as Technical Report TD-18/93, Dip. di Informatica, Università di Pisa, Pisa, Italy.

35. R. Giacobazzi. "Optimal" collecting semantics for analysis in a hierarchy of logic program semantics. Technical Report LIX, Ecole Polytechnique, LIX/RR/94, 1994.

36. R. Giacobazzi, S. K. Debray, and G. Levi. A Generalized Semantics for Constraint Logic Programs. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pages 581–591, 1992.

37. R. Giacobazzi, S. Debray, and G. Levi. Joining Abstract and Concrete Computations in Constraint Logic Programmimg. In M. Nivat, C. Rattray, T. Rus and G. Scollo, editors, *Proc. Third International Conference on Algebraic Methodology and Software Technology*, AMAST'93, Workshops in Computing Series, pages 109–126. Springer-Verlag, London 1993.

38. J.S. Golan. The theory of semirings with applications in mathematics and theoretical computer science. Longman, Harlow, 1992.

39. 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.

40. M. Hanus. Analysis of Nonlinear Constraints in CLP($\mathcal{R}$). In *Proc. Tenth International Conference on Logic Programming*, pages 83–99. MIT Press.

41. L. Henkin, J.D. Monk, and A. Tarski. *Cylindric Algebras. Part I and II.* North-Holland, Amsterdam, 1971.

42. M. Hermenegildo, R. Warren, and S.K. Debray. Global flow analysis as a practical compilation tool. *Journal of Logic Programming*, 13(4):349–366, 1992.

43. Y.E. Ioannidis and E. Wong. An Algebraic Approach to Recursive Inference. In L. Kerschberg, editor, *Proc. First International Conference on Expert Database Systems - Charleston SC*, pages 295–309, 1987.

44. D. Jacobs and A. Langen. Static Analysis of Logic Programs for Independent AND Parallelism. *Journal of Logic Programming*, 13(2 & 3):291–314, 1992.

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

46. J. Jaffar and M.J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19 & 20:503–581, 1994.

47. J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP($\mathcal{R}$) Language and System. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.

48. N. Jørgensen, K. Marriot, and S. Michaylov. Some Global Compile-Time Optimizations for CLP($\mathcal{R}$). In *Proc. 1991 International Symposium on Logic Programming*, pages 420–434, 1991.

49. M. Karr. Affine Relationships Among Variables of a Program. *Acta Informatica*, 6:133–151, 1976.

50. G. Karner. On limits in complete semirings. *Semigroup Forum* 45:148–165, 1992.

51. G. Karner. A topology for complete semirings. In P. Enjalbert, E.W. Mayr and K.W. Wagner, editors, *11th Annual Symposium on Theoretical Aspects of Computer Science* - Proceedings STACS'94, *Lecture Notes in Computer Science*, pages 389–400. Springer-Verlag, Berlin, 1994.

52. G. Karner. Personal communication. 1994.

53. J.-L. Lassez, M. J. Maher, and K. Marriott. Unification Revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, Los Altos, Ca., 1988.

54. J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1987. Second edition.

55. K. Marriott and H. Søndergaard. Notes for a tutorial on Abstract Interpretation of Logic Programs. Informal Proc. of the North American Conference on Logic Programming'89, 1989.

56. K. Marriott and H. Søndergaard. Abstract Interpretation of Logic Programs: the Denotational Approach. In A. Bossi, editor, *Proc. Fifth Italian Conference on Logic Programming*, pages 399–425, 1990.

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

58. K. Marriott and H. Søndergaard. Precise and Efficient Groundness Analysis for Logic Programs. *ACM Letters on Programming Languages and Systems* 2(1–4):181–196, 1993.

59. K. Marriott, H. Søndergaard, and N. D. Jones. Denotational Abstract Interpretation of Logic Programs. *ACM Transactions on Programming Languages and Systems* 16(3):607–648.

60. A. Melton, D.A. Schmidt, and G.E. Strecker. Galois Connections and Computer Science Applications. In D. Pitt et al., editor, *Category Theory and Computer Programming*, volume 240 of *Lecture Notes in Computer Science*, pages 299–312. Springer-Verlag, Berlin, 1986.

61. J. Morgado. A Characterization of the Closure Operators by means of one Axiom. *PortugaliæMathematica*, 21(3):155–156, 1962.

62. Oystein Ore. Combinations of Closure Relations. *Annals of Mathematics*, 44(3):514–533, 1943.

63. P. Panangaden, V. A. Saraswat, P. Scott, and R. Seely. A Hyperdoctrinal view of Concurrent Constraint Programming. In J. deBakker and G. Roszenberg and W. deRoever eds. *Proc. of the REX Workshop*, volume 666 of *Lecture Notes in Computer Science*, pages 457–476. Springer-Verlag, Berlin, 1992.

64. 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*, pages 333–353. ACM, 1991.

65. D. Scott. Domains for Denotational Semantics. In M. Nielsen and E. M. Schmidt, editors, *Proc. Ninth International Colloquium on Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 577–613. Springer-Verlag, Berlin, 1982.

66. M.B. Smyth. Topology. In S. Abramsky, Dov M. Gabbay and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, Background Mathematical Structures, pages 641–761. Oxford Science Publications, 1992.

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

68. Jichang Tan and I-Peng Lin. Compiling Dataflow Analysis of Logic Programs. In *ACM Programming Language Design and Implementation*, volume 27 of *SIGPLAN Notices*, pages 106–115. ACM Press, 1992.

69. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5:285–309, 1955.

70. J. W. Thatcher, E. G. Wagner, and J. B. Wright. More on advice on structuring compilers and proving them correct. *Theoretical Computer Science*, 15:223–249, 1981.

71. M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

72. A. van Gelder. Deriving Constraints Among Argument Sizes in Logic Programs. In *Proc. of the eleventh ACM Conference on Principles of Database Systems*, pages 47–60. ACM, 1990.

73. K. Verschaetse and D. De Schreye. Derivation of Linear Size Relations by abstract interpretation. In M. Bruynooghe and M. Wirsing, editors, *Fourth International Symposium on Programming Language Implementation and Logic Programming, Proc. of PLILP'92*, volume 631 of *Lecture Notes in Computer Science*, pages 296–310. Springer-Verlag, Berlin, 1992.

74. M. Ward. The Closure Operators of a Lattice. *Annals of Mathematics*, 43(2):191–196, 1942.

75. E. Zaffanella, R. Giacobazzi, and G. Levi. Abstracting Synchronization in Concurrent Constraint Programming. In M. Hermenegildo and J. Penjam, editors, *Proc. Sixth Int'l Symp. on Programming Language Implementation and Logic Programming*, PLILP'94, volume 844 of *Lecture Notes in Computer Science*, pages 57–72. Springer-Verlag, Berlin, 1994.

A. APPENDIX: PROOFS OF SELECTED RESULTS

Proposition 3.1.

Closed semirings are continuous.

PROOF. [sketch] It is easy to prove, from R_4, that if $\{a_i\}_{i\in I}$ is a possibly infinite family of objects in $\mathcal{C}$, and $a \in \mathcal{C}$ then, if $a \oplus a_i = a$ for all $i \in I$, then $a \oplus (\sum_{i\in I} a_i) = a$. Therefore, by idempotence, a closed semiring is always continuous or finitary (this claim has been recently proved in [51], Proposition 13).

Proposition 3.2.

$\mathcal{C}$ is partially ordered by $\trianglelefteq$, and forms a complete lattice.

PROOF. [sketch] Since $(\mathcal{C}, \otimes, \oplus, \mathbf{1}, \mathbf{0})$ is a closed semiring, $\oplus$ is associative, commutative and idempotent, whence it is easy to show that $\mathcal{C}$ is partially ordered by $\trianglelefteq$. For every $c \in \mathcal{C}$, $c \oplus \mathbf{0} = \mathbf{0} \oplus c = c$, so $\mathbf{0} \trianglelefteq c$, i.e., $\mathbf{0}$ is the least element of the partially ordered set $\langle \mathcal{C}, \trianglelefteq \rangle$. Consider any family $X = \{c_i\}_{i\in\mathcal{I}} \subseteq \mathcal{C}$. By Definition 3, $\mathcal{C}$ is closed under finite and infinite applications of $\oplus$, whence $\sum X \in \mathcal{C}$. From associativity, commutativity, and idempotence of $\oplus$ we have, for any $i \in \mathcal{I}$, $c_i \oplus (\sum X) = c_0 \oplus \ldots \oplus c_i \oplus c_i \oplus \ldots = \sum X$, whence $c_i \trianglelefteq (\sum X)$ for all $c_i \in X$, i.e., $\sum X$ is an upper bound of X. From continuity, $\sum X$ is also the least upper bound of X. It follows that $\mathcal{C}$ is a $\oplus$-semilattice with a minimal element $\mathbf{0}$. Thus $\langle \mathcal{C}, \trianglelefteq \rangle$ is a complete lattice.

Theorem 3.1.

Let $\mathcal{A}$ be an arbitrary constraint system. For any $c, c' \in \mathcal{C}$, $x \in V$, $X \subseteq V$ and $t, t', t'' \in \tau$ such that x ind t, the following properties hold:

P1: $\exists_X \exists_X c = \exists_X c$;
P2: $c \trianglelefteq c' \quad \Rightarrow \quad \exists_X c \trianglelefteq \exists_X c'$;
P3: $\forall c, c' \in \mathcal{C}: \quad c' \trianglelefteq \exists_X c \;\Leftrightarrow\; \exists_X c' \trianglelefteq \exists_X c$;
P4: $\forall c, c' \in \mathcal{C}: \quad c \trianglelefteq c' \;\wedge\; c' \trianglelefteq \exists_X c \;\Rightarrow\; \exists_X c = \exists_X c'$;
P5: $\exists_{\{x\}} c = c$ *iff* $\exists_{\{x\}} \tilde{c} = c$ *for some* $\tilde{c} \in \mathcal{C}$;
P6: $\exists_{\{x\}} c = c$ *if* x *ind* c;
P7: $d_{t,t'} = \exists_{\{x\}}(d_{t,x} \otimes d_{x,t'})$ *where* x *ind* t, t',
P8: $c \trianglelefteq c' \quad \Rightarrow \quad \partial^t_x c \trianglelefteq \partial^t_x c'$;
P9: $\partial^t_x \exists_{\{x\}} c = \exists_{\{x\}} c$;
P10: $\partial^t_x\ c = c$ *iff* $\partial^t_x \tilde{c} = c$ *for some* $\tilde{c} \in \mathcal{C}$;
P11: $\exists_X \mathbf{1} = \mathbf{1}$; $\exists_X c = \mathbf{0}$ *iff* $c = \mathbf{0}$;
P12: $\exists_{\{x\}} d_{x,t} = \mathbf{1}$;
P13: $(d_{t,t'} \otimes d_{t',t''}) \oplus d_{t,t''} = d_{t,t''}$ *(transitivity).*

PROOF. Let $c, c' \in \mathcal{C}$, $x \in V$, $X \subseteq V$ and $t, t', t'' \in \tau$. It is straightforward to prove the componentwise monotonicity of $\otimes$ by the axioms.

P1, P2: Straightforward by definition.

P3: By distributivity of $\exists$ on $\oplus$ and idempotence: $c' \oplus \exists_X c = \exists_X c \quad \Rightarrow \exists_X c' \oplus \exists_X c = \exists_X c$. The other implication follows by C_2.

P4: By monotonicity of $\exists$ and from the previous property we obtain: $\exists_X c \trianglelefteq \exists_X c'$ and $\exists_X c' \trianglelefteq \exists_X c$ respectively.

P5: By idempotence if x is bound in c then $\exists_{\{x\}}c = c$. Notice that the set of fixed-points of $\exists_X$ is the range of $\exists_X$ itself. The converse is straightforward.

P6: Assume x *ind* t for some $t \in \tau$. This implies $\exists_{\{x\}}c = \exists_{\{x\}}\partial_x^t c$. From the definition of ∂_x^t and Axiom C_4, this is equal to $\exists_{\{x\}}(d_{x,t} \otimes c)$, which is nothing but $\partial_x^t c$. Since x *ind* t, we have $\partial_x^t c = c$, which proves the result.

P7: Assume $t, t' \in \tau$ and x *ind* t, t'. By Axiom D_3: $\exists_{\{x\}}(d_{t,x} \otimes d_{x,t'}) = d_{[t/x]x,t'} = d_{t,t'}$.

P8: From the monotonicity of $\exists$ and $\otimes$.

P9: Assume x *ind* t, then by definition $\partial_x^t \exists_{\{x\}}c = \exists_{\{x\}}(d_{x,t} \otimes \exists_{\{x\}}c)$. From Axiom C_3 this is equal to $\exists_{\{x\}}d_{x,t} \otimes \exists_{\{x\}}c$. From Property P12 (proved below), this in turn is equal to $\exists_{\{x\}}c$.

P10: By P9, $\partial_x^t\ \partial_x^t c = \partial_x^t(\exists_{\{x\}}(d_{x,t} \otimes c)) = \exists_{\{x\}}(d_{x,t} \otimes c) = \partial_x^t c$.

P11: Both follow from Axioms C_1 and C_2.

P12: By Axiom D_1, for any $t' \in \tau$: $\exists_{\{x\}}d_{x,t} = \exists_{\{x\}}(d_{x,t} \otimes d_{t',t'}) = d_{[t/x]t',[t/x]t'} = \mathbf{1}$;

P13: Assume x *ind* t, t', t''.

$$\begin{aligned}
d_{t,t''} &= \exists_{\{x\}}(d_{t,x} \otimes d_{x,t''}) && [\ \text{P7}\ ] \\
&= \partial_x^{t'}\exists_{\{x\}}(d_{t,x} \otimes d_{x,t''}) && [\ x \notin FV(\exists_{\{x\}}(d_{t,x} \otimes d_{x,t''})) \text{ and P9}\ ] \\
&\trianglerighteq \partial_x^{t'}(d_{t,x} \otimes d_{x,t''}) \ = \ d_{t,t'} \otimes d_{t',t''} && [\ \partial\text{-monotonicity (P8) and Axiom } D_4\ ]
\end{aligned}$$

Lemma 3.1.

For any constraint system $\mathcal{A}$, if c and c' are $\mathcal{A}$-constraints and X is a set of variables such that X ind c, then $\exists_X(c \otimes c') = c \otimes \exists_X(c')$.

Proof. By Theorem 3.1, $\exists_X c = c$, whence $\exists_X(c \otimes c') = \exists_X(\exists_X c \otimes c')$. From Axiom C_3, this is equal to $\exists_X c \otimes \exists_X c'$, which is equal to $c \otimes \exists_X c'$ since $\exists_X c = c$ by Theorem 3.1.

Lemma 3.2.

For any constraints c and c' in a constraint system $\mathcal{A}$, $c \otimes \exists_{\{x\}}c' = \exists_{\{y\}}(c \otimes \tilde{c}')$, where y ind c, c'; $y \neq x$ and $\tilde{c}' = \partial_x^y c'$.

Proof. Suppose that y *ind* c, y *ind* c', $y \neq x$, and $\tilde{c}' = \partial_x^y c'$. Since y *ind* c, we have $\exists_{\{y\}}(c \otimes \tilde{c}') = c \otimes \exists_{\{y\}}\partial_x^y c'$. Since $y \neq x$, Axiom C_4 implies $\exists_{\{y\}}\partial_x^y c' = \exists_{\{x\}}\exists_{\{y\}}(d_{x,y} \otimes c')$. Since y *ind* c', this is equal to $\exists_{\{x\}}(\exists_{\{y\}}d_{x,y} \otimes c')$. From Property P12, this in turn is equal to $\exists_{\{x\}}c'$. It follows that $c \otimes \exists_{\{x\}}c' = c \otimes \exists_{\{y\}}\partial_x^y c' = \exists_{\{y\}}(c \otimes \tilde{c}')$.

Lemma 4.1.

$\mathbf{1} \parallel p(\bar{t}) \rightsquigarrow_P^* c \parallel \varepsilon$ iff $d_{\bar{x},\bar{t}} \parallel p(\bar{x}) \rightsquigarrow_P^* c' \parallel \varepsilon$ and $\exists_{\{\bar{x}\}}c' = c$; for $\bar{x}$ not used in the derivation for c.

PROOF. $\mathbf{1} \parallel p(\bar{t}) \rightsquigarrow_P^* c \parallel \varepsilon$ if and only if for some clause $C \equiv p(\bar{t}_0) \ :- \ c_0 \parallel B_1, ..., B_n$ in P,

$$\mathbf{1} \parallel p(\bar{t}) \rightsquigarrow_P d_{\bar{t},\bar{t}_0} \otimes c_0 \parallel B_1, ..., B_n \rightsquigarrow_P^* c \parallel \varepsilon.$$

This is true if and only if $\bar{x}$ *ind* C and

$$\mathbf{1} \parallel p(\bar{t}) \rightsquigarrow_P \exists_{\{\bar{x}\}}(d_{\bar{t},\bar{x}} \otimes d_{\bar{x},\bar{t}_0}) \otimes c_0 \parallel B_1, ..., B_n \rightsquigarrow_P^* c \parallel \varepsilon,$$

i.e., if and only if $\bar{x}$ *ind* C and

$$\mathbf{1} \parallel p(\bar{t}) \rightsquigarrow_P \exists_{\{\bar{x}\}}(d_{\bar{t},\bar{x}} \otimes d_{\bar{x},\bar{t}_0} \otimes c_0) \parallel B_1, ..., B_n \rightsquigarrow_P^* c \parallel \varepsilon.$$

Since $\bar{x}$ is not used in the derivation of c: $\bar{x}$ *ind* c, this is true if and only if $d_{\bar{t},\bar{x}} \parallel p(\bar{x}) \rightsquigarrow_P^* c' \parallel \varepsilon$ and $\exists_{\{x\}} c' = c$. The result follows.

Lemma 4.2.

Let $G = c_0 \parallel p_1(\bar{t}_1), ..., p_n(\bar{t}_n)$ be an $\mathcal{A}$-goal and $P \in CLP(\mathcal{A})$. $\mathcal{J}_P(G) = c$ iff there exist $p_i(\bar{x}_i) \ :- \ c_i \in \mathcal{O}(P)$, such that $\bar{x}_i$ ind G and $\bar{x}_i \cap \bar{x}_j = \emptyset$ for $1 \le i, j \le n$, $i \ne j$; and $c = \exists(c_0 \otimes d_{\bar{x}_1,\bar{t}_1} \otimes c_1 ... \otimes d_{\bar{x}_n,\bar{t}_n} \otimes c_n)_{var(G)}$.

PROOF.
Let $P \in CLP(\mathcal{A})$. Let G be a goal. We prove that:

$$\left\{ \exists(c)_{var(G)} \;\middle|\; G \rightsquigarrow_P^* c \parallel \varepsilon \right\} =$$

$$\left\{ \exists(c)_{var(G)} \;\middle|\; \begin{array}{l} G = c_0 \parallel p_1(t_1), ..., p_m(t_m) \\ \forall i = 1..m: \ 1 \parallel p_i(\bar{x}_i) \rightsquigarrow_P^* c_i \parallel \varepsilon \\ \bar{x}_i \ ind \ G \ \ and \ \ \bar{x}_i \cap \bar{x}_j = \emptyset \ for \ j = 1...n \ i \ne j \\ c = c_0 \otimes d_{\bar{x}_1,\bar{t}_1} \otimes \exists(c_1)_{x_1} ... \otimes d_{\bar{x}_m,\bar{t}_m} \otimes \exists(c_m)_{x_m} \end{array} \right\}.$$

($\subseteq$) The proof is by induction on the length n of the derivation. In the base case, assume $G = \mathbf{1} \parallel p(t)$ and $G \rightsquigarrow_P c \parallel \varepsilon$. By definition, this holds iff $p(x) \ :- \ c' \in P$ and $c = d_{t,x} \otimes c'$. By Lemma 3.1 and because $x \cup var(c')$ *ind* t we have $\exists(d_{t,x} \otimes c')_{var(t)} = \exists(d_{t,x} \otimes \exists(c')_x)_{var(t)}$. Let $\dot{V} = var(d_{t,x} \otimes c')$, then

$$\begin{array}{rll} \exists_{\dot{V} \setminus var(t)}(d_{t,x} \otimes c') & = \exists_{\dot{V} \setminus (var(t) \cup x)} \exists_x (d_{t,x} \otimes c') & [\text{ Axiom } C_4 \ ] \\ & = \exists_x (d_{t,x} \otimes \exists_{\dot{V} \setminus (var(t) \cup x)} c') & [\text{ Lemma 3.1 }] \\ & = \exists_x (d_{t,x} \otimes \exists(c')_x) & [\ FV(d_{t,x} \otimes \exists(c')_x) = var(t) \cup x \ ] \\ & = \exists(d_{t,x} \otimes \exists(c')_x)_{var(t)}. & [\ x \subseteq \dot{V} \setminus var(t) \ ] \end{array}$$

In the inductive case, let $G = c_0 \parallel p_1(t_1), ..., p_m(t_m)$ such that $G \rightsquigarrow_P^n c \parallel \varepsilon$. Consider a clause $p_1(x_1) \ :- \ c_1 \parallel b_1(r_1), ..., b_k(r_k)$ in P and assume:

$$G \rightsquigarrow_P c_0 \otimes d_{t_1,x_1} \otimes \exists(c_1)_{vc} \parallel b_1(r_1), ..., b_k(r_k), p_2(t_2), ..., p_m(t_m) \rightsquigarrow_P^{n-1} c \parallel \varepsilon$$

where $vc = x_1 \bigcup_{i=1}^{k} var(r_i)$. By the inductive hypothesis, for $i, l = 1..k$ and $j, w = 2..m$, we can define $b_i(y_i) \ :- \ c'_i$ and $p_j(x_j) \ :- \ c_j$ such that x_j, y_i *ind* G; $y_i \ne x_j$; for $i \ne l$, $j \ne w$: $y_i \ne y_l$ and $x_j \ne x_w$; $1 \parallel b_i(y_i) \rightsquigarrow_P^* \tilde{c}'_i \parallel \varepsilon$, $1 \parallel p_j(x_j) \rightsquigarrow_P^* \tilde{c}_j \parallel \varepsilon$, $c'_i = \exists(\tilde{c}'_i)_{y_i}$, $c_j = \exists(\tilde{c}_j)_{x_j}$ (i.e., $FV(c'_i) \subseteq y_i$ and $FV(c_j) \subseteq x_j$); and, by Lemma 3.1:

$$\exists(c)_{var(G')} = c_0 \otimes d_{t_1,x_1} \otimes \exists(c_1)_{vc} \otimes \exists \left(\begin{array}{c} d_{r_1,y_1} \otimes c'_1 \otimes ... \otimes d_{r_k,y_k} \otimes c'_k \otimes \\ d_{t_2,x_2} \otimes c_1 \otimes ... \otimes d_{t_m,x_m} \otimes c_m \end{array} \right)_{var(G')}$$

where $G' = c_0 \otimes d_{t_1,x_1} \otimes \exists(c_1)_{vc} \parallel b_1(r_1), ..., b_k(r_k), p_2(t_2), ..., p_m(t_m)$. By definition:

$$\mathbf{1} \parallel p_1(x_1) \rightsquigarrow^*_P c_1 \otimes d_{r_1,y_1} \otimes \tilde{c}'_1 \otimes ... \otimes d_{r_k,y_k} \otimes \tilde{c}'_k \parallel \varepsilon.$$

Let $c'' = c_1 \otimes d_{r_1,y_1} \otimes \tilde{c}'_1 \otimes ... \otimes d_{r_k,y_k} \otimes \tilde{c}'_k$. Consider now the constraint:

$$c_0 \otimes \exists(d_{t_1,x_1} \otimes \exists(c'')_{x_1} \otimes d_{t_2,x_2} \otimes c_1 \otimes ... \otimes d_{t_m,x_m} \otimes c_m)_{var(G)}.$$

Because for $j = 2..m$: $FV(c_j) \subseteq x_j$; x_j *ind* t_j; x_1 *ind* t_1 and $FV(\exists(c'')_{x_1}) \subseteq x_1$, the constraint above is equivalent to

$$c_0 \otimes \exists_{\{x_1...x_m\}}(d_{t_1,x_1} \otimes \exists(c'')_{x_1} \otimes d_{t_2,x_2} \otimes c_1 \otimes ... \otimes d_{t_m,x_m} \otimes c_m). \qquad (*)$$

Let $\dot{V} = var(c)$. Since $\dot{V} \setminus var(G) = (\dot{V} \setminus var(G')) \cup vc$ where, with an abuse of notation, we denote by $var(G')$ the set of variables $vc \cup var(c_0) \cup var(t_1) \bigcup_{i=2}^{m} var(t_i)$, and for $i, l = 1..k$ and $j, w = 2..m$: $FV(c'_i) \subseteq y_i$ and $FV(c_j) \subseteq x_j$; x_j, y_i *ind* G; $y_i \neq x_j$; for $i \neq l$, $j \neq w$: $y_i \neq y_l$ and $x_j \neq x_w$. From Lemma 3.1 we have:

$$\begin{aligned}
&\exists(c)_{var(G)} = \exists_{vc}(\exists(c)_{var(G')}) \\
&= \exists_{vc}\left(c_0 \otimes d_{t_1,x_1} \otimes \exists(c_1)_{vc} \otimes \exists_{y_1...y_k,x_2...x_m}\left(\begin{array}{c} d_{r_1,y_1} \otimes c'_1 \otimes \cdots \otimes d_{r_k,y_k} \otimes c'_k \\ d_{t_2,x_2} \otimes c_1 \otimes \cdots \otimes d_{t_m,x_m} \otimes c_m \end{array}\right)\right) \\
&= c_0 \otimes \exists_{x_1..x_m}\left(\begin{array}{c} d_{t_1,x_1} \otimes \exists_{y_1...y_k,vc\setminus x_1}\left(\begin{array}{c} \exists(c_1)_{vc} \otimes d_{r_1,y_1} \otimes c'_1 \\ \otimes ... \otimes d_{r_k,y_k} \otimes c'_k \end{array}\right) \\ \otimes d_{t_2,x_2} \otimes c_1 \otimes ... \otimes d_{t_m,x_m} \otimes c_m \end{array}\right) \\
&= c_0 \otimes \exists_{x_1...x_m}(d_{t_1,x_1} \otimes \exists(c'')_{x_1} \otimes d_{t_2,x_2} \otimes c_1 \otimes \cdots \otimes d_{t_m,x_m} \otimes c_m)
\end{aligned}$$

which is equivalent to $(*)$.

($\supseteq$) The proof follows by observing that, because of Lemma 3.2, we can always replace hidden variables with fresh variables in arbitrary, but finite, conjunctions of (complete) constraints (such as those computed for each atomic goal $\mathbf{1} \parallel p_i(\bar{x}_i)$). Assume $G = c_0 \parallel p_1(\bar{t}_1), ..., p_m(\bar{t}_m)$ and for each $i = 1, ..., m$: $1 \parallel p_i(\bar{x}_i) \rightsquigarrow^*_P c_i \parallel \varepsilon$, where $\bar{x}_i$ *ind* G; $\bar{x}_i \cap \bar{x}_j = \emptyset$ for $j = 1, ..., n$ such that $i \neq j$. Let

$$c = c_0 \otimes d_{\bar{x}_1,\bar{t}_1} \otimes \exists(c_1)_{\bar{x}_1} ... \otimes d_{\bar{x}_m,\bar{t}_m} \otimes \exists(c_m)_{\bar{x}_m}.$$

Notice that the computed answer constraints c_i for $i = 1, ..., m$ are finite constraints. Moreover, since we assume V to be infinite, for each $i = 1, ..., m$, there are (fresh) variables $nv_i \subseteq V$ such that $nv_i \cap var(G) = \emptyset$, nv_i *ind* $c_1, ..., c_m$ and $nv_i \cap nv_j = \emptyset$ for $j = 1, ..., m$. Thus, by Lemma 3.2 if $v_i = var(c_i) \setminus \bar{x}_i$ and $\tilde{c}_i = \partial^{nv_i}_{v_i} c_i$:

$$c = \exists_{nv_1..nv_m}(c_0 \otimes d_{\bar{x}_1,\bar{t}_1} \otimes \tilde{c}_1 ... \otimes d_{\bar{x}_m,\bar{t}_m} \otimes \tilde{c}_m).$$

It is straightforward to associate a successful derivation for G with renaming variables $nv_1, ..., nv_m$ such that $\exists(c)_{var(G)} = \exists_{\bar{x}_1..\bar{x}_m} \exists_{nv_1..nv_m} c$.

Lemma A.1. Let $\mathcal{A}$ be a constraint system, $C = p(\bar{x}) :- c \parallel p_1(\bar{t}_1), ..., p_n(\bar{t}_n)$ be an $\mathcal{A}$-clause and I be an $\mathcal{A}$-interpretation. For $i = 1, ..., n$ let $p_i(\bar{x}'_i) :- c'_i$ and $p_i(\bar{x}''_i) :- c''_i$ be variants of constrained atoms in I that have been renamed apart from C and from each other. Then: $\exists(c \otimes d_{\bar{t}_1,\bar{x}'_1} \otimes c'_1 \otimes ... \otimes d_{\bar{t}_n,\bar{x}'_1} \otimes c'_n)_{\bar{x}} = \exists(c \otimes d_{\bar{t}_1,\bar{x}''_1} \otimes c''_1 \otimes ... \otimes d_{\bar{t}_n,\bar{x}''_1} \otimes c''_n)_{\bar{x}}$.

Proof. Assume, for $i = 1, ..., n$: $p_i(\bar{x}'_i)$:− c'_i and $p_i(\bar{x}''_i)$:− c''_i be renamings of some $p_i(\bar{x}_i)$:− c_i, such that $\bar{x}'_i$, $\bar{x}''_i$, and $\bar{x}''_i$ do not share any variables with each other or with C. By definition, for $i = 1, ..., n$: $c''_i = \exists_{\bar{x}_i}(d_{\bar{x}'_i,\bar{x}_i} \otimes c_i)$ and $c'_i = \exists_{\bar{x}_i}(d_{\bar{x}''_i,\bar{x}_i} \otimes c_i)$. Let $\tilde{c}' = \exists(c \otimes d_{\bar{t}_1,\bar{x}'_1} \otimes c'_1 \otimes \cdots \otimes d_{\bar{t}_n,\bar{x}'_n} \otimes c'_n)_{\bar{x}}$ and $\dot{V} = var(c \otimes d_{\bar{t}_1,\bar{x}'_1} \otimes c'_1 \otimes \cdots \otimes d_{\bar{t}_n,\bar{x}'_n} \otimes c'_n)$. By applying Theorem 3.1 we can hide variables $\bar{x}'_1..\bar{x}'_n$ in $\tilde{c}'$, namely:

$$\begin{aligned}
\tilde{c}' &= \exists_{\dot{V}\setminus\{\bar{x},\bar{x}'_1..\bar{x}'_n\}}\exists_{\bar{x}'_1..\bar{x}'_n}\left(\begin{array}{c} c \otimes d_{\bar{t}_1,\bar{x}'_1} \otimes \exists_{\bar{x}_1}(d_{\bar{x}'_1,\bar{x}_1} \otimes c_1)\otimes \\ \ldots \otimes d_{\bar{t}_n,\bar{x}'_n} \otimes \exists_{\bar{x}_n}(d_{\bar{x}'_n,\bar{x}_n} \otimes c_n) \end{array}\right) \\
&\quad [\text{ Axiom } C_4 \text{ and definition }] \\
&= \exists_{\dot{V}\setminus\{\bar{x},\bar{x}'_1..\bar{x}'_n\}}\left(\begin{array}{c} c \otimes \exists_{\bar{x}'_1,\bar{x}_1}(d_{\bar{t}_1,\bar{x}'_1} \otimes d_{\bar{x}'_1,\bar{x}_1} \otimes c_1)\otimes \\ \ldots \otimes \exists_{\bar{x}'_n,\bar{x}_n}(d_{\bar{t}_n,\bar{x}'_1} \otimes d_{\bar{x}'_n,\bar{x}_n} \otimes c_n) \end{array}\right) \\
&\quad [\text{ independence }] \\
&= \exists_{\dot{V}\setminus\{\bar{x},\bar{x}'_1..\bar{x}'_n\}}(c \otimes \exists_{\bar{x}_1}(d_{\bar{t}_1,\bar{x}_1} \otimes c_1) \otimes \ldots \otimes \exists_{\bar{x}_n}(d_{\bar{t}_n,\bar{x}_n} \otimes c_n)) \\
&\quad [\text{ Theorem 3.1 }] \\
&= \exists_{(\dot{V}\setminus\{\bar{x},\bar{x}'_1..\bar{x}'_n\})\cup\{\bar{x}_1..\bar{x}_n\}}(c \otimes d_{\bar{t}_1,\bar{x}_1} \otimes c_1 \otimes \ldots \otimes d_{\bar{t}_n,\bar{x}_n} \otimes c_n) \\
&\quad [\text{ independence }]
\end{aligned}$$

Since $\bar{x}'_1..\bar{x}'_n$ are independent for $(c \otimes d_{\bar{t}_1,\bar{x}_1} \otimes c_1 \otimes \ldots \otimes d_{\bar{t}_n,\bar{x}_n} \otimes c_n)$, we have

$$\tilde{c}' = \exists(c \otimes d_{\bar{t}_1,\bar{x}_1} \otimes c_1 \otimes \ldots \otimes d_{\bar{t}_n,\bar{x}_n} \otimes c_n)_{\bar{x}}.$$

The same argument can be applied to prove that $\tilde{c}'' = \exists(c \otimes d_{\bar{t}_1,\bar{x}_1} \otimes c_1 \otimes \ldots \otimes d_{\bar{t}_n,\bar{x}_n} \otimes c_n)_{\bar{x}}$.

Lemma 4.3.

Let $\mathcal{A}$ be a constraint system and $P \in CLP(\mathcal{A})$. For any $I \in \wp(\mathcal{B}^{\mathcal{A}})$: $(T_P(I^\flat))^\flat = (T_P(I))^\flat$.

Proof. The lemma follows by $\exists$/meet-distributivity. Because P is a finite set, it is equivalent to prove that for any $C \in P$: $(T_{\{C\}}(I^\flat))^\flat = (T_{\{C\}}(I))^\flat$. Let $C = p(\bar{t})$:− $c \,[\!]\, p_1(\bar{t}_1), ..., p_n(\bar{t}_n)$ and I be an interpretation. Assume $p(\bar{x})$:− $\exists(\tilde{c})_{\bar{x}} \in (T_{\{C\}}(I^\flat))^\flat$. By definition, $p_i(\bar{x}_i)$:− $c_i \ll_{C,\bar{x}_1,...,\bar{x}_{i-1}} I^\flat$; $\tilde{c} = d_{\bar{x},\bar{t}} \otimes c \otimes c'_1 \otimes \cdots \otimes c'_n$, where $c'_i = d_{\bar{x}_i,\bar{t}_i} \otimes c_i$; and $\bar{x}$ *ind* $C, c_1, ..., c_n$. By definition, for each $i = 1, ..., n$ there exists a set of indices W_i such that $p_i(\bar{y}_k)$:− $\ddot{c}_k \ll_{x_i} I$ for any $k \in W_i$, and $c_i = \sum_{k \in W_i} \partial^{x_i}_{y_k} \ddot{c}_k$. Therefore, for each $i = 1, ..., n$ we can choose $k_i \in W_i$ such that

$$p_i(\bar{y}_{k_i}) \ :- \ \ddot{c}_{k_i} \ll_{\bar{x},\bar{x}_i,C,\bar{y}_{k_1},...,\bar{y}_{k_{i-1}}} I.$$

Thus, by T_P definition, for each $k_1 \in W_1$, ..., $k_n \in W_n$:

$$p(\bar{x}) \ :- \ \exists(d_{\bar{x},\bar{t}} \otimes c \otimes c'_1 \otimes \cdots \otimes c'_n)_{\bar{x}} \in T_{\{C\}}(I),$$

where $c'_i = d_{\bar{y}_{k_i},\bar{t}_i} \otimes \ddot{c}_{k_i}$, for $i = 1, ..., n$.
The thesis follows by $\exists$ and $\otimes$ distributivity: $\exists(\tilde{c})_{\bar{x}}$ is equal to

$$\exists(d_{\bar{x},\bar{t}} \otimes c \otimes \sum_{k_1 \in W_1} (d_{\bar{x}_1,\bar{t}_1} \otimes \exists_{\{y_{k_1}\}}(d_{\bar{y}_{k_1},\bar{x}_1} \otimes \ddot{c}_{k_1})) \otimes \cdots \otimes$$

$$\sum_{k_n \in W_n} (d_{\bar{x}_n,\bar{t}_n} \otimes \exists_{\bar{y}_{k_n}}(d_{\bar{y}_{k_n},\bar{x}_n} \otimes \ddot{c}_{k_n})))_{\bar{x}}$$

$$= \exists(d_{\bar{x},\bar{t}} \otimes c \otimes \sum_{k_1 \in W_1} (d_{\bar{y}_{k_1},\bar{t}_1} \otimes \ddot{c}_{k_1}) \otimes \cdots \otimes \sum_{k_n \in W_n} (d_{\bar{y}_{k_n},\bar{t}_n} \otimes \ddot{c}_{k_n}))_{\bar{x}}$$
$$= \sum_{k_1 \in W_1} \cdots \sum_{k_n \in W_n} \exists \left(d_{\bar{x},\bar{t}} \otimes c \otimes (d_{\bar{y}_{k_1},\bar{t}_1} \otimes \ddot{c}_{k_1}) \otimes \ldots \otimes (d_{\bar{y}_{k_n},\bar{t}_n} \otimes \ddot{c}_{k_n}) \right)_{\bar{x}}.$$

Proposition 4.2.

Let $\mathcal{A}$ be a constraint system and $P \in CLP(\mathcal{A})$. T_P is a continuous function on the complete lattice $\langle \wp(\mathcal{B}^{\mathcal{A}}), \subseteq \rangle$ and $T_P^\flat$ is continuous on the complete lattice $\langle \wp^\flat(\mathcal{B}^{\mathcal{A}}), \sqsubseteq \rangle$.

Proof. The proof of continuity of T_P follows the standard lines (e.g. see [2]). The continuity of $T_P^\flat$ is then a straightforward consequence of the continuity of T_P and Lemma 4.3.

Theorem 4.1.

Let $\mathcal{A}$ be a constraint system with dimension ω, and $P \in CLP(\mathcal{A})$, then $\mathcal{F}(P) = \mathcal{O}(P)/\!\sim$ and $\mathcal{F}^\flat(P) = (\mathcal{O}(P)/\!\sim)^\flat$.

Proof. We consider the condensed case only, the other case is similar. The proof is by induction: for each $n \in \mathcal{N}$, we show that if $p(\bar{x})\ :-\ c$ is any element of $T_P^\flat \uparrow n(\emptyset)$ then $c = \sum\{\exists(c')_{\{\bar{x}\}} \mid \mathbf{1} \parallel p(\bar{x}) \leadsto_P^* c' \parallel \varepsilon\}$. The base case is straightforward by the definition of $\leadsto$. For the inductive case, consider a predicate p in P defined by clauses $C_1, \ldots, C_k$, with $C_j =$ '$p(\bar{t}_j)\ :-\ c_{0_j} \parallel p_{1_j}(\bar{t}_{1_j}), ..., p_{m_j}(\bar{t}_{m_j})$,' $1 \le j \le k$. Let $p(\bar{x})\ :-\ c \in T_P^\flat \uparrow n(\emptyset)$, then by definition: $c = \sum_{j=1}^{k} \exists(c_j)_{\bar{x}}$ where $\bar{x}$ *ind* C_j for each $j = 1, ..., k$:

$$c_j = d_{\bar{x},\bar{t}_j} \otimes c_{0_j} \otimes d_{\bar{x}_{1_j},\bar{t}_{1_j}} \otimes c_{1_j} \otimes \ldots \otimes d_{\bar{x}_{m_j},\bar{t}_{m_j}} \otimes c_{m_j}$$

and where $p_{i_j}(\bar{x}_{i_j})\ :-\ c_{i_j} \in T_P^\flat \uparrow (n-1)(\emptyset)$ $(FV(c_{i_j}) \subseteq \bar{x}_{i_j})$; $\bar{x}_{i_j}$ *ind* C_j; $\bar{x}$, $\bar{x}_{i_j}$ and$\bar{x}_{l_h}$ are mutually variable-disjoint for each $i, l = 1, .., m$, $j, h = 1, .., k$ such that $i \neq l$ and $j \neq h$.

By the inductive hypothesis: for each $i = 1, .., m$ and $j = 1, .., k$:

$$c_{i_j} = \sum\{\exists(\dot{c})_{\bar{x}_{i_j}} \mid \mathbf{1} \parallel p_{i_j}(\bar{x}_{i_j}) \leadsto_P^* \dot{c} \parallel \varepsilon\}.$$

Thus, by distributivity of $\otimes$ over $\oplus$: $c_j = \sum D_j$ where

$$D_j = \left\{ c_j \;\middle|\; \begin{array}{l} c_j = d_{\bar{x},\bar{t}_j} \otimes c_{0_j} \otimes d_{\bar{x}_{1_j},\bar{t}_{1_j}} \otimes \exists(\dot{c}_{1_j})_{\bar{x}_{1_j}} \otimes \ldots \otimes d_{\bar{x}_{m_j},\bar{t}_{m_j}} \otimes \exists(\dot{c}_{m_j})_{\bar{x}_{m_j}} \\ \mathbf{1} \parallel p_{1_j}(\bar{x}_{1_j}) \leadsto_P^* \dot{c}_{1_j} \parallel \varepsilon,\ \ldots\ , \mathbf{1} \parallel p_{m_j}(\bar{x}_{m_j}) \leadsto_P^* \dot{c}_{m_j} \parallel \varepsilon \end{array} \right\}.$$

Let $G_j = d_{\bar{x},\bar{t}_j} \otimes c_{0_j} \parallel p_{1_j}(\bar{t}_{1_j}), ..., p_{m_j}(\bar{t}_{m_j})$. Because for $j, h = 1, .., k$, $i, l = 1, .., m$: x *ind* C_j and $x_{i_j} \neq x_{l_h}$ for each $i \neq l$ and $j \neq h$; we have

$$\sum_{j=1}^{k} \exists(\textstyle\sum D_j)_{\bar{x}} = \sum_{j=1}^{k} (\textstyle\sum\{\exists(\exists(c_j)_{var(G_j)})_{\bar{x}} \mid G_j \leadsto_P^* c_j \parallel \varepsilon\}) \quad [\text{ by Lemma 4.2 }]$$
$$= \textstyle\sum\{\exists(c)_{\bar{x}} \mid \mathbf{1} \parallel p(\bar{x}) \leadsto_P^* c \parallel \varepsilon\}. \quad [\ \bar{x} \subseteq var(G_j)\ ]$$

It is sufficient now to prove that $\mathbf{1} \parallel p(\bar{x}) \leadsto_P^n c \parallel \varepsilon$ implies that there exists $\tilde{c} \in \mathcal{C}$ and $p(\bar{x})\ :-\ \exists(\dot{c})_{\bar{x}} \in \mathcal{F}^\flat(P)$ such that $\exists(\dot{c})_{\bar{x}} = \exists(c)_{\bar{x}} \oplus \tilde{c}$. We prove this by induction: the base case is straightforward by the definition of $T_P^\flat$. For the inductive case, assume that if $\mathbf{1} \parallel p(\bar{x}) \rightsquigarrow_P^n c \parallel \varepsilon$ then there exists $\tilde{c} \in \mathcal{C}$ and $p(\bar{x}) \ :- \ \exists(\dot{c})_{\bar{x}} \in \mathcal{F}^\flat(P)$ such that $\exists(\dot{c})_{\bar{x}} = \exists(c)_{\bar{x}} \oplus \tilde{c}$. Consider:

$$\mathbf{1} \parallel p(\bar{x}) \ \rightsquigarrow_P c' \parallel \ p_1(\bar{t}_1), ..., p_m(\bar{t}_m) \ \rightsquigarrow_P^n \ c \parallel \varepsilon$$

where $p(\bar{t}) \ :- \ c_0 \parallel p_1(\bar{t}_1), ..., p_m(\bar{t}_m)$ is some renamed apart clause in P and $c' = \mathbf{1} \otimes d_{\bar{x},\bar{t}} \otimes c_0 = d_{\bar{x},\bar{t}} \otimes c_0$. Consider the atomic goals: $\mathbf{1} \parallel p_i(\bar{x}_i)$ for $i = 1, ..., m$. By Lemma 4.2:

$$\mathbf{1} \parallel p_i(\bar{x}_i) \rightsquigarrow_P^k \ddot{c}_i \parallel \varepsilon$$

where $k \leq n$ and $c = d_{\bar{x},\bar{t}} \otimes c_0 \otimes d_{\bar{x}_1,\bar{t}_1} \otimes \exists(\ddot{c}_1)_{\bar{x}_1} \otimes ... \otimes d_{\bar{x}_m,\bar{t}_m} \otimes \exists(\ddot{c}_m)_{\bar{x}_m}$. By the inductive hypothesis, for each $i = 1, ..., m$ there exists $p_i(\bar{x}_i) \ :- \ \exists(\dot{c}_i)_{\bar{x}_i} \in \mathcal{F}^\flat(P)$ and $\tilde{c}_i \in \mathcal{C}$ such that $\exists(\dot{c}_i)_{\bar{x}_i} = \exists(\ddot{c}_i)_{\bar{x}_i} \oplus \tilde{c}_i$.

The definition of $T_P^\flat$ implies that

$$p(\bar{x}) \ :- \ \exists(d_{\bar{x},\bar{t}} \otimes c_0 \otimes d_{\bar{x}_1,\bar{t}_1} \otimes \exists(\dot{c}_1)_{\bar{x}_1} \otimes ... \otimes d_{\bar{x}_m,\bar{t}_m} \otimes \exists(\dot{c}_m)_{\bar{x}_m})_{\bar{x}}$$

is in $\mathcal{F}^\flat(P)$. Therefore

$$p(\bar{x}) \ :- \ \exists(d_{\bar{x},\bar{t}} \otimes c_0 \otimes d_{\bar{x}_1,\bar{t}_1} \otimes (\exists(\ddot{c}_1)_{\bar{x}_1} \oplus \tilde{c}_1) \otimes \ldots \otimes d_{\bar{x}_m,\bar{t}_m} \otimes (\exists(\ddot{c}_m)_{\bar{x}_m} \oplus \tilde{c}_1))$$

is in $\mathcal{F}^\flat(P)$. The theorem is proved because $\otimes$ is associative, distributes on $\oplus$, and $\oplus$ is associative and commutative.

Proposition 5.1.

Let $\mathcal{A}$ and $\mathcal{A}'$ be constraint systems as specified above. Let also $c \in \mathcal{C}$, $x \in V$ and $t \in \tau$ such that x ind t. If $\alpha_\kappa : \mathcal{A} \xrightarrow{s} \mathcal{A}'$ then $\alpha(\partial_x^t c) \trianglelefteq' \partial'^{\kappa(t)}_{\kappa(x)} \alpha(c)$.

Proof. Assume the hypothesis.

$$\begin{array}{rcl} \alpha(\partial_x^t c) & = & \alpha(\exists_x(d_{x,t} \otimes c)) \\ & \trianglelefteq' & \exists'_{\kappa(x)} \alpha(d_{x,t} \otimes c) \\ & \trianglelefteq' & \exists'_{\kappa(x)} (\alpha(d_{x,t}) \otimes' \alpha(c)) \\ & \trianglelefteq' & \exists'_{\kappa(x)} (d'_{\kappa(x),\kappa(t)} \otimes' \alpha(c)) = \partial'^{\kappa(t)}_{\kappa(x)} \alpha(c). \end{array}$$

Proposition 5.3.

Let $\mathcal{A}$ and $\mathcal{A}'$ be constraint systems with universes $\mathcal{C}$ and $\mathcal{C}'$ respectively. If $\mathcal{A}'$ is correct with respect to $\mathcal{A}$ by means of a semimorphism α, there exists a mapping $\gamma : \mathcal{C}' \to \mathcal{C}$ such that (α, γ) is a Galois insertion of $\langle \mathcal{C}', \trianglelefteq' \rangle$ into $\langle \mathcal{C}, \trianglelefteq \rangle$.

Proof. Assume the hypothesis. Define $\gamma(c') = \sum\{c \mid \alpha(c) \trianglelefteq' c'\}$. Let $c'_1 \trianglelefteq' c'_2$. Then, from the definition of γ and the monotonicity of α, $\sum\{c \mid \alpha(c) \trianglelefteq' c'_1\} \oplus \sum\{c \mid \alpha(c) \trianglelefteq' c'_2\} = \gamma(c'_2)$, i.e., γ is monotonic. Let $c' \in \mathcal{C}'$ and $c \in \mathcal{C}$. From the definition of γ, we have $\alpha(\gamma(c')) = \alpha(\sum\{c \mid \alpha(c) \trianglelefteq' c'\})$. From the additivity of α this is equal to $\sum\{\alpha(c) \mid \alpha(c) \trianglelefteq' c'\}$, and this in turn is equal to c' from the surjectivity of α. Thus, $\alpha(\gamma(c')) = c'$.
From the monotonicity of α we have $c \trianglelefteq \sum\{\tilde{c} \mid \alpha(\tilde{c}) \trianglelefteq' \alpha(c)\}$. It follows, from the definition of γ, that $c \trianglelefteq \gamma(\alpha(c))$.

Theorem 5.1.

Let $P \in CLP(\mathcal{A})$ and $P' \in CLP(\mathcal{A}')$ be the corresponding program on $\mathcal{A}'$. If $\mathcal{A}'$ is correct with respect to $\mathcal{A}$, there exists $\beta : \wp^\flat(\mathcal{B}^{\mathcal{A}}) \to \wp^\flat(\mathcal{B}^{\mathcal{A}'})$ such that $\beta(\mathcal{F}(P)) \sqsubseteq' \mathcal{F}(P')$ and $\beta(\mathcal{F}^\flat(P)) \sqsubseteq' \mathcal{F}^\flat(P')$.

Proof. As before we prove the condensed case as the other is similar. Let $\mathcal{A}'$ be a correct constraint system with respect to $\mathcal{A}$, and α_κ be the corresponding semimorphism. Let $a \ll I$ denote a variant of an object $a \in I$ that has been renamed apart from all elements of I. The mapping $\beta : \wp^\flat(\mathcal{B}^{\mathcal{A}}) \to \wp^\flat(\mathcal{B}^{\mathcal{A}'})$ defined as

$$\beta(I) = \left\{ \ [p(\kappa(\bar{x})) \ :- \ \alpha(c)]_\sim \ \middle| \ p(\bar{x}) \ :- \ c \ll I \ \right\}$$

is continuous by definition. As observed in [5], by β continuity, the proof can be reduced to show that $\beta(T_P^\flat(I)) \sqsubseteq' T_P^\flat(\beta(I))$ for all $I \in \wp^\flat(\mathcal{B}^{\mathcal{A}})$. Let $I \in \wp^\flat(\mathcal{B}^{\mathcal{A}})$, and $\{C_1^p, ..., C_m^p\}$ be the set of clauses in P defining p, where $C_j^p =$ '$p(\bar{t}_j) \ :- \ c_j \ [\![\ q_{1_j}(\bar{t}_{1_j}), \ldots q_{n_j}(\bar{t}_{n_j})$', $1 \le j \le m$. Let $\{C_1^{p\prime}, ..., C_m^{p\prime}\}$ be the corresponding set of clauses on $\mathcal{A}'$ in the program P', and

$$\langle q_{1_j}(\bar{x}_{1_j}) \ :- \ \tilde{c}_{1_j}, \ldots, q_{n_j}(\bar{x}_{n_j}) \ :- \ \tilde{c}_{n_j} \rangle \ll_{C_j^p} I$$

and $c_{i_j} = d_{\bar{x}_{i_j}, \bar{t}_{i_j}} \otimes \tilde{c}_{i_j}$, $1 \le i \le n$. Then, $[p(\kappa(\bar{x})) \ :- \ c]_\sim \in \beta(T_P^\flat(I))$ where

$$c = \alpha \left(\sum_{j=1}^{m} \exists (d_{\bar{x}, \bar{t}_j} \otimes c_j \otimes c_{1_j} \otimes \cdots \otimes c_{n_j})_{\bar{x}} \right)$$

From the definition of semimorphism, we have

$$c \trianglelefteq \left({\sum_{j=1}^{m}}' \exists' (\alpha(d_{\bar{x}, \bar{t}_j}) \otimes' \alpha(c_j) \otimes' \alpha(c_{1_j}) \otimes' \cdots \otimes' \alpha(c_{n_j}))_{\kappa(\bar{x})} \right)$$

Let $\langle q_{1_j}(\kappa(\bar{x}_{1_j})) \ :- \ \tilde{c}'_{1_j}, \ldots, q_{n_j}(\kappa(\bar{x}_{n_j})) \ :- \ \tilde{c}'_{n_j} \rangle \ll_{C_j^{p\prime}} \beta(I)$, and for $1 \le i \le n$ $c'_{i_j} = \alpha(d_{\bar{x}_{i_j}, \bar{t}_{i_j}}) \otimes' \tilde{c}'_{i_j}$. It follows that

$$c \trianglelefteq \left({\sum_{j=1}^{m}}' \exists' (\alpha(d_{\bar{x}, \bar{t}_j}) \otimes' \alpha(c_j) \otimes' c'_{1_j} \otimes' \cdots \otimes' c'_{n_j})_{\kappa(\bar{x})} \right).$$

By the definition of a semimorphism, for any two terms t_1, t_2: $\alpha(d_{t_1, t_2}) \trianglelefteq d'_{\kappa(t_1), \kappa(t_2)}$. Then, by definition, $\beta(T_P^\flat(I)) \sqsubseteq' T_P^\flat(\beta(I))$.

Proposition 5.5.

$Vrel_S$ is a morphism of term systems.

PROOF. Rigid terms are mapped to $\emptyset$. Denote by s' the substitution operation on τ_V. Let $t_1, t_2 \in \tau$ and $x \in V$. If x is not a relevant variable in t_2 then $Vrel_S(s_x(t_1, t_2)) = Vrel_S(t_2)$ and $s'_x(Vrel_S(t_1), Vrel_S(t_2)) = Vrel_S(t_2)$ because $x \notin Vrel_S(t_2)$. Assume x to be a relevant variable in t_2. By definition $Vrel_S(s_x(t_1, t_2)) = Vrel_S(t_1) \cup (Vrel_S(t_2) \setminus \{x\})$. The thesis follows from the definition of s', namely: $s'_x(Vrel_S(t_1), Vrel_S(t_2)) = (Vrel_S(t_2) \setminus \{x\}) \cup Vrel_S(t_1)$.

Theorem 5.2.

α is an additive semimorphism from the constraint system $\mathcal{H}$ to Prop.

PROOF. (outline)
We prove that α is well defined for the simpler case of the "size" norm (for more details see [34]; a similar condition is also proved in [18]). Let $c = \cup\{c_i \mid i \in I\}$ and $c' = \cup\{c'_i \mid i \in I'\}$ be equivalent satisfiable constraints. Suppose that $\bigvee_{i \in I} \alpha(c_i)$ is not equivalent to $\bigvee_{i \in I'} \alpha(c'_i)$. Then there must be a truth assignment r for which there exists $i \in I$ such that for each $j \in I'$: $\alpha(c_i)(r) = true$ but $\alpha(c'_j)(r) = false$. Now $\alpha(c_i)$ and $\alpha(c'_j)$ are both conjunctions of formulae of the form $X \leftrightarrow Y$ for $X, Y \subseteq V$, since the existentially quantified variables can be eliminated by replacing the constraint with the disjunction of all the constraints obtained by replacing the variables with the combinations of all the possible values *true* and *false* [21, 55]. Let X_r, Y_r be a partition of V such that $r(X_r) = true$ and $r(Y_r) = false$ (obviously, r cannot bind all the variables to *true*—otherwise both the constraints should be *true*). For each $j \in I'$, each of the conjunctive subformula of $\alpha(c'_j)$ contains $X_j \leftrightarrow Y_j$ for some X_j and Y_j such that $X_j \subseteq X_r$ and $Y_j \cap Y_r \neq \emptyset$. This is a contradiction because there exists $j' \in I'$ such that if θ_{X_j} is a (grounding) solution for $c'_{j'}$ on the variables X_j: $y \in Y_j$ is ground in $(c'_{j'})\theta_{X_j}$ iff y is ground in $(c_i)\theta_{X_j}$. The properties of semimorphism are straightforward by the definition.

Lemma 5.2.

Let ρ be an $\exists$-consistent upper closure operator on the constraint system $\mathcal{A}$ with universe $\mathcal{C}$, term system τ and set of variables V. Then:

1. *for each $c \in \mathcal{C}$, $x \in V$ and $t \in \tau$ such that x ind t: $\rho(\partial_x^t c) = \partial_x^t \rho(\partial_x^t c)$;*
2. *for each $c \in \mathcal{C}$, $X \subseteq V$: $\rho(\exists_X c) = \rho(\exists_X \rho(c))$.*

PROOF. For (1), we have from the definition of ∂_x^t and the $\exists$-consistency of ρ that $\partial_x^t \rho(\partial_x^t c) = \exists_{\{x\}}(d_{x,t} \otimes \exists_{\{x\}} \rho(\exists_{\{x\}}(d_{x,t} \otimes c)))$. From Axiom C_3 and Property P12, this is equal to $\exists_{\{x\}} \rho(\exists_{\{x\}}(d_{x,t} \otimes c)) = \exists_{\{x\}} \rho(\partial_x^t c)$. Since ρ is $\exists$-consistent, this is equal to $\rho(\partial_x^t c)$. The result follows.

The proof of (2) proceeds as follows: Let $c \in \mathcal{C}$ and $X \subseteq V$. From the monotonicity of ρ and $\exists$, we have: $\rho(\exists_X c) \trianglelefteq \rho(\exists_X \rho(c))$. By $\exists$-consistency: $\exists_X \rho(c) \trianglelefteq \exists_X \rho(\exists_X c) = \rho(\exists_X c)$. The result then follows from the idempotence and monotonicity of ρ.

Lemma 5.3.

Let ρ be a consistent upper closure operator on the constraint system $\mathcal{A}$, with universe $\mathcal{C}$, term system τ and set of variables V. Then for each $c \in \mathcal{C}$, $x \in V$ and $t \in \tau$ such that x ind t: $\rho(\partial_x^t c) = \rho(\partial_x^t \rho(c))$.

Proof. Let $c \in \mathcal{C}$, $x \in V$ and $t \in \tau$:

$$
\begin{array}{rcll}
\rho(\partial_x^t \rho(c)) & = & \rho(\exists_{\{x\}}(d_{x,t} \otimes \rho(c))) & [\text{ definition }] \\
& = & \rho(\exists_{\{x\}} \rho(d_{x,t} \otimes \rho(c))) & [\ \rho(\exists_X c) = \rho(\exists_X \rho(c))\ ] \\
& = & \rho(\exists_{\{x\}} \rho(\rho(d_{x,t}) \otimes \rho(c))) & [\text{ by idempotence and } \otimes\text{-quasi morphism }] \\
& = & \rho(\exists_{\{x\}} \rho(d_{x,t} \otimes c)) & [\text{ by } \otimes\text{-quasi morphism }] \\
& = & \rho(\partial_x^t c). & [\ \rho(\exists_X c) = \rho(\exists_X \rho(c)) \text{ and definition }]
\end{array}
$$

Proposition 5.6.

Let $\mathcal{A}$ and $\mathcal{A}^\sharp$ be constraint systems with universes $\mathcal{C}$ and $\mathcal{C}^\sharp$ respectively, such that $\mathcal{A}^\sharp$ is correct with respect to $\mathcal{A}$ by means of a surjective and additive semi-morphism α. Let $\gamma = \lambda c^\sharp. \sum\{c \mid \alpha(c) \trianglelefteq^\sharp c^\sharp\}$ and $\rho = \gamma \circ \alpha$. Then:

1. *$\rho(\mathcal{C})$ is isomorphic to $\mathcal{C}^\sharp$;*
2. *if $\alpha(\exists_X c) = \exists^\sharp_{\kappa(X)} \alpha(\exists_X c)$ for every $X \subseteq V$ and $c \in \mathcal{C}$, then $\gamma \circ \alpha$ is $\exists$-consistent.*

Proof. Assume the hypothesis. Let $\imath : \rho(\mathcal{C}) \to \mathcal{C}^\sharp$ such that $\forall c \in \mathcal{C} :\ \imath(\rho(c)) = \alpha(\rho(c))$. By the hypothesis of Galois insertion: $\alpha(\rho(c)) = \alpha(c)$. $\imath$-surjectivity is straightforward by α-surjectivity. $\imath$-injectivity follows because $\alpha(c) = \alpha(c') \Rightarrow \rho(c) = \rho(c')$. This establishes that $\rho(\mathcal{C})$ is isomorphic to $\mathcal{C}^\sharp$.

For the second part, let $c \in \mathcal{C}$ and X be a set of variables. We prove that $\exists_X \rho(\exists_X c) = \rho(\exists_X c)$. By $\exists$-distributivity: $\exists_X \rho(\exists_X c) = \sum\{\exists_X c' \in \mathcal{C} \mid \alpha(c') \trianglelefteq^\sharp \alpha(\exists_X c)\}$ and $\rho(\exists_X c) = \sum\{c' \in \mathcal{C} \mid \alpha(c') \trianglelefteq^\sharp \alpha(\exists_X c)\}$. We show that $\alpha(\exists_X c') \trianglelefteq^\sharp \alpha(\exists_X c)$ for each constraint c' such that $\alpha(c') \trianglelefteq^\sharp \alpha(\exists_X c)$, i.e., if $c' \in \rho(\exists_X c)$ then $\exists_X c' \in \rho(\exists_X c)$. By the hypothesis we have:

$$
\begin{array}{rcl}
\alpha(\exists_X c') & \trianglelefteq^\sharp & \exists^\sharp_{\kappa(X)} \alpha(c) \\
& \trianglelefteq^\sharp & \exists^\sharp_{\kappa(X)} \alpha(\exists_X c) \\
& = & \alpha(\exists_X c)
\end{array}
$$

Thus, by $\exists$-extensivity (i.e., $c \trianglelefteq \exists_X c$ for each constraint c and set of variables X) we have:

$$\rho(\exists_X c) = \sum \left\{ \exists_X c' \;\middle|\; \alpha(c') \trianglelefteq^\sharp \alpha(\exists_X c) \right\} = \exists_X \rho(\exists_X c).$$

Proposition 5.7.

Let $\mathcal{A}$ and $\mathcal{A}^\sharp$ be constraint systems with universes $\mathcal{C}$ and $\mathcal{C}^\sharp$ respectively, such that $\mathcal{A}^\sharp$ is correct with respect to $\mathcal{A}$ by means of a surjective and additive semi-morphism α. Let $\gamma : \mathcal{C}^\sharp \longrightarrow \mathcal{C}$ be defined as $\gamma(c^\sharp) = \sum\{c \mid \alpha(c) \trianglelefteq^\sharp c^\sharp\}$ and $\rho = \gamma \circ \alpha$. Let $X \subseteq V$ and $c, c_1, c_2 \in \mathcal{C}$. If α_κ is a morphism on constraint systems then:

1. *$\exists_X \rho(c) = \rho(\exists_X c)$*
2. *$\rho(\rho(c_1) \otimes \rho(c_2)) = \rho(c_1 \otimes c_2)$.*

Proof. (1) Let $c \in \mathcal{C}$ and X be a set of variables. By $\exists$-extensivity: $\rho(\exists_X c) \trianglelefteq \exists_X \rho(c)$. By Proposition 5.6 and by the hypothesis, $\rho(\exists_X c) = \sum\{\exists_X c' \mid \exists_{\kappa(X)} \alpha(c') =$

$\exists_{\kappa(X)}\alpha(c)\}$. However, $\exists_X \rho(c) = \sum\{\exists_X c' \mid \alpha(c') = \alpha(c)\} \trianglelefteq \rho(\exists_X c)$, which proves the thesis.

(2) From the monotonicity of ρ and $\otimes$, we have $\rho(c_1 \otimes c_2) \trianglelefteq \rho(\rho(c_1) \otimes \rho(c_2))$. The converse is satisfied by definition:

$$\rho(c_1 \otimes c_2) = \sum\{c \mid \alpha(c) \trianglelefteq^\sharp \alpha(c_1 \otimes c_2)\}$$
$$\rho(\rho(c_1) \otimes \rho(c_2)) = \sum\{c \mid \alpha(c) \trianglelefteq^\sharp \alpha(c' \otimes c''),\ \alpha(c') \trianglelefteq^\sharp \alpha(c_1),\ \alpha(c'') \trianglelefteq^\sharp \alpha(c_2)\}.$$

By hypothesis, α is a morphism. Thus, by transitivity, and $\otimes^\sharp$ monotonicity if $c \in \rho(\rho(c_1) \otimes \rho(c_2))$ then $c \in \left\{ \ c \,\middle|\ \alpha(c) \trianglelefteq^\sharp \alpha(c_1 \otimes c_2) \ \right\}$.

Lemma 5.4.

Let ρ be a consistent upper closure operator on the constraint system $\mathcal{A}$, with universe $\mathcal{C}$, term system τ and set of variables V. Then for each $c \in \rho(\mathcal{C})$, $x \in V$ and $t \in \tau$ such that x ind t: $\tilde{\partial}^t_x c = \rho(\partial^t_x c)$.

Proof. By Lemma 5.2, $\rho(\partial^t_x c) = \rho(\exists_{\{x\}} \rho(d_{x,t} \otimes c))$. From $\otimes$-quasi morphism, this is equal to $\tilde{\exists}_{\{x\}}(\rho(\rho(d_{x,t}) \otimes \rho(c)))$, where by definition $\tilde{\exists}_X = \rho \circ \exists_X$. Since $c \in \rho(\mathcal{C})$ and ρ is a closure operator and therefore idempotent, this is equal to $\tilde{\exists}_{\{x\}}(\rho(d_{x,t})\tilde{\otimes}c)$. The lemma follows.

Theorem 5.3.

Let ρ be a consistent upper closure operator on the constraint system $\mathcal{A}$. $\rho(\mathcal{A})$ is a constraint system.

Proof. Let $c, c_1, c_2 \in \rho(\mathcal{C})$, $C \subseteq \rho(\mathcal{C})$, $X, Y \subseteq V$, $x \in V$ $t, t_1, t_2 \in \tau$ and x *ind* t. In the following we denote $\tilde{\exists}_X = \rho \circ \exists_X$ and $\tilde{\partial}^t_x c = \rho(\exists_{\{x\}}(\rho(\rho(d_{x,t}) \otimes c)))$.

R_1: We prove that $(\rho(\mathcal{C}), \tilde{\otimes}, \tilde{\oplus}, \mathbf{1}, \rho(\mathbf{0}))$ is a closed semiring. By ρ idempotence and $\otimes/\oplus$-quasi morphism: $\rho(\mathbf{0})\tilde{\oplus}c = \rho(\rho(\mathbf{0}) \oplus c) = \rho(\mathbf{0} \oplus c) = c$; $\mathbf{1}\tilde{\otimes}c = \rho(\mathbf{1} \otimes c) = c$; $\rho(\mathbf{0})\tilde{\otimes}c = \rho(\rho(\mathbf{0}) \otimes c) = \rho(\mathbf{0} \otimes c) = \rho(\mathbf{0})$. Distributivity follows by $\otimes$-quasi morphism:

$$\begin{array}{rcl} c\tilde{\otimes}(\tilde{\sum}C) & = & \rho(c \otimes \rho(\sum C)) \\ & = & \rho(\rho(c) \otimes \rho(\sum C)) \\ & = & \rho(\sum\{\rho(c \otimes c') \mid c' \in C\}) \\ & = & \tilde{\sum}\{\rho(c \otimes c') \mid c' \in C\} \\ & = & \tilde{\sum}\{c\tilde{\otimes}c' \mid c' \in C\} \end{array}$$

C_1: By Lemma 5.2 $\rho(\exists_X(\rho(\mathbf{0}))) = \rho(\exists_X(\mathbf{0})) = \rho(\mathbf{0})$;

C_2: $c\tilde{\oplus}\tilde{\exists}_X c = \rho(c \oplus \rho(\exists_X c)) = \rho(c \oplus \exists_X c) = \tilde{\exists}_X \rho(c) = \tilde{\exists}_X c$;

C_3: By definition, $\tilde{\exists}_X(c_1\tilde{\otimes}\tilde{\exists}_X c_2) = \rho(\exists_X \rho(c_1 \otimes \rho(\exists_X c_2)))$. Since ρ is a consistent upper closure operator, it is a $\otimes$-quasi morphism, and further, $c_1 = \rho(c_1)$ since $c_1 \in \rho(\mathcal{C})$; thus, $\rho(c_1 \otimes \rho(\exists_X c_2)) = \rho(\rho(c_1) \otimes \rho(\exists_X c_2)) = \rho(c_1 \otimes \exists_X c_2)$. Thus, we have $\tilde{\exists}_X(c_1\tilde{\otimes}\tilde{\exists}_X c_2) = \rho(\exists_X \rho(c_1 \otimes \exists_X c_2))$. From Lemma 5.2, this is equal to $\rho(\exists_X c_1 \otimes \exists_X c_2) = \tilde{\exists}_X c_1 \tilde{\otimes}\tilde{\exists}_X c_2$.

C_4: By Lemma 5.2: $\tilde{\exists}_X \tilde{\exists}_Y c = \rho(\exists_X \rho(\exists_Y c)) = \rho(\exists_{X \cup Y} c) = \tilde{\exists}_{X \cup Y} c$.

C_5: By definition, $\tilde{\exists}_X(\tilde{\sum}C) = \rho(\exists_X \rho(\sum C))$. From Lemma 5.2 and Axiom C_5 this is equal to $\rho(\sum(\{\exists_X c \mid c \in C\}))$. Since ρ is an upper closure operator it is also a quasi-complete join-morphism, whence this is equal to $\rho(\sum(\rho(\{\exists_X c \mid c \in C\}))) = \rho(\sum\{\rho(\exists_X c) \mid c \in C\})$. This is nothing but $\tilde{\sum}\{\tilde{\exists}_X c \mid c \in C\}$.

D_1: is straightforward.

D_2: is straightforward.

D_3: By Lemmata 5.3 and 5.4:

$$\tilde{\partial}^t_x \rho(d_{t_1,t_2}) = \rho(\partial^t_x \rho(d_{t_1,t_2})) = \rho(\partial^t_x d_{t_1,t_2}) = \rho(d_{[t/x]t_1,[t/x]t_2}).$$

D_4: By Lemmata 5.3 and 5.4, and by $\otimes$-quasi morphism:

$$\begin{array}{rcl} \tilde{\partial}^t_x(c_1 \tilde{\otimes} c_2) & = & \rho(\partial^t_x(\rho(c_1 \otimes c_2))) \\ & = & \rho(\partial^t_x(c_1 \otimes c_2)) \\ & = & \rho(\partial^t_x c_1 \otimes \partial^t_x c_2) \\ & = & \rho(\rho(\partial^t_x c_1) \otimes \rho(\partial^t_x c_2)) \\ & = & \tilde{\partial}^t_x c_1 \tilde{\otimes} \tilde{\partial}^t_x c_2. \end{array}$$

Theorem 5.4.

Let ρ be a consistent upper closure operator for a constraint system $\mathcal{A}$ with universe of constraints $\mathcal{C}$ and let $c_1, c_2 \in \mathcal{C}$. $\rho(c_1) \otimes \rho(c_2) \trianglelefteq \rho(c_1 \otimes c_2)$. If $\mathcal{A}$ is $\otimes$-idempotent and $\mathbf{1}$ is the annihilator for $\oplus$, then $\rho(c_1 \otimes c_2) = \rho(c_1) \otimes \rho(c_2)$.

Proof. Let $c_1, c_2 \in \mathcal{C}$. $\rho(c_1) \otimes \rho(c_2) \trianglelefteq \rho(c_1 \otimes c_2)$ follows by ρ-extensivity. Assume the hypothesis on $\mathcal{A}$. We prove that: $\rho(c_1) \otimes \rho(c_2) = \rho(c_1 \otimes c_2)$. By the hypothesis, for each constraint c, c': $c \oplus (c \otimes c') = c \otimes (1 \oplus c') = c$ (i.e., $c \otimes c' \trianglelefteq c$). Let $c \trianglelefteq \rho(c_1 \otimes c_2)$. By monotonicity: $c \trianglelefteq \rho(c_1)$ and $c \trianglelefteq \rho(c_2)$. Thus: $c \otimes c \trianglelefteq \rho(c_1) \otimes \rho(c_2)$. The thesis follows by $\otimes$-idempotence.

Theorem 6.1.

Let $\mathcal{A}$ be a constraint system with universe $\mathcal{C}$, variables V and term system τ. If ρ is an upper closure operator satisfying any existential property and a (possibly empty) combination of properties P_1—P_3, then $\rho(\mathcal{A})$ is a non-distributive constraint system.

Proof. We prove the non-distributive laws for a generic upper closure operator ρ satisfying either E_1 or E_2. The other claims for any combination of properties P_1—P_3 can be easily derived from them. Let $c, c' \in \rho(\mathcal{C})$, $C \subseteq \rho(\mathcal{C})$, $X, Y\{x\} \subseteq V$ and $t, t_1, t_2 \in \tau$ such that x *ind* t:

R_3: $\rho(\mathbf{0}) \otimes c \trianglerighteq \rho(\mathbf{0} \otimes c) = \rho(\mathbf{0})$.

R_5:

$$\begin{array}{rcl} c \otimes (\tilde{\sum}C) & = & c \otimes \rho(\sum C) \\ & = & \rho(c \otimes \rho(\sum C)) \\ & \trianglerighteq & \rho(c \otimes (\sum C)) \\ & = & \rho(\sum\{\ (c \otimes c') \mid c' \in C\ \}) \\ & = & \tilde{\sum}\{\ (c \otimes c') \mid c' \in C\ \}. \end{array}$$

C_1: The case where ρ satisfies E_1 is proved in Theorem 5.3. Otherwise, it is straightforward to prove the non-distributive version of C_1 by extensivity for any closure operator.

C_3: Assume ρ satisfies E_1:

$$\begin{array}{rcl}
\rho(\exists_X (c \otimes \rho(\exists_X c'))) & = & \rho(\exists_X (c \otimes \exists_X \rho(\exists_X c'))) \\
& = & \rho(\exists_X c \otimes \exists_X \rho(\exists_X c')) \\
& \trianglelefteq & \rho(\exists_X c) \otimes \rho(\exists_X \rho(\exists_X c')) \\
& = & \rho(\exists_X c) \otimes \rho(\exists_X c').
\end{array}$$

The case where ρ satisfies E_2 is straightforward.

C_4: The case where ρ satisfies E_1 is proved in Theorem 5.3. Assume ρ satisfies E_2:

$$\rho(\exists_X (\rho(\exists_Y c))) = \rho(\exists_{X \cup Y} (\rho(c))) = \rho(\exists_{X \cup Y} c)$$

therefore C_4 is always distributive.

C_5: For a generic upper closure operator ρ:

$$\begin{array}{rcl}
\rho(\exists_X (\rho(\sum C))) & \trianglerighteq & \rho(\sum \left\{ \exists_X c' \,\middle|\, c' \in C \right\}) \\
& = & \rho(\sum \left\{ \rho(\exists_X c') \,\middle|\, c' \in C \right\}).
\end{array}$$

The case where ρ satisfies E_1 is proved in Theorem 5.3.

D_3: Straightforward by ρ extensivity.

D_4: It follows by $\otimes$ idempotence and commutativity:

$$\begin{array}{rc}
\rho(\exists_{\{x\}}(\rho(d_{x,t}) \otimes \rho(c))) \otimes \rho(\exists_{\{x\}}(\rho(d_{x,t}) \otimes \rho(c'))) & \trianglerighteq \\
\rho(\exists_{\{x\}}(\rho(d_{x,t}) \otimes \rho(c)) \otimes \exists_{\{x\}}(\rho(d_{x,t}) \otimes \rho(c'))) & \trianglerighteq \\
\rho(\exists_{\{x\}}(\rho(d_{x,t}) \otimes \rho(c) \otimes \rho(d_{x,t}) \otimes \rho(c'))) & = \\
\rho(\exists_{\{x\}}(\rho(d_{x,t}) \otimes \rho(c) \otimes \rho(c'))) &
\end{array}$$

Proposition 6.1.

$(\tau^a, S^a, \mathcal{V})$ is a term system of dimension α, and κ is a morphism from τ into τ^a.

PROOF. We simplify the notation by assuming w.l.o.g. that $\mathcal{V} = V$. Let $t, t' \in \tau$, $a, b \in \tau^a$ such that $\kappa(t) = a$ and $\kappa(t') = b$, then we have:

T_1: $s^a_x(a, x) = \kappa(s_x(t, x)) = \kappa(t) = a$.

T_2: $s^a_x(a, y) = \kappa(s_x(t, y)) = y$.

T_3: $s^a_x(a, s^a_x(y, b)) = \kappa(s_x(t, t_2))$ where $\kappa(t_2) = \kappa(s_x(y, t'))$.
Then: $s^a_x(a, s^a_x(y, b)) = \kappa(s_x(t, s_x(y, t'))) = \kappa(s_x(y, t'))$.

T_4: The proof is analogous to that for T_3.

Proposition 6.2.

$\rho(\mathcal{A})$ is a correct R_5 and D_4 non-distributive constraint system.

PROOF. From Theorem 6.1, it is enough to prove the D_3 distributivity. Let $\rho = \gamma \circ \alpha$, $t, t_1, t_2 \in \tau$ and $x \in V$. The proof follows by $\exists/\alpha$ additivity and from the basic properties of κ:

$$\begin{array}{rl}
\rho(\exists_{\{x\}}(\rho(d_{x,t}) \otimes \rho(d_{t_1,t_2}))) & = \\
\rho(\exists_{\{x\}} \oplus \{d_{x,t_1} \otimes d_{t_2,t_3} \mid \kappa(t) = \kappa(t_1),\ \kappa(t_1) = \kappa(t_2),\ \kappa(t_2) = \kappa(t_3)\}) & = \\
\gamma(\oplus\{\alpha(\exists_{\{x\}} d_{x,t_1} \otimes d_{t_2,t_3}) \mid \kappa(t) = \kappa(t_1),\ \kappa(t_1) = \kappa(t_2),\ \kappa(t_2) = \kappa(t_3)\}) & = \\
\gamma(\oplus\{\alpha(d_{[t_1/x]t_2,[t_1/x]t_3}) \mid \kappa(t) = \kappa(t_1),\ \kappa(t_1) = \kappa(t_2),\ \kappa(t_2) = \kappa(t_3)\}) & = \\
\gamma(d_{\kappa([t/x]t_1),\kappa([t/x]t_2)}) & = \\
\rho(d_{[t/x]t_1,[t/x]t_2}) &
\end{array}$$

Correctness is straightforward since ρ is an upper closure operator.

Proposition 6.5.

$Rel = (\wp(\mathcal{L}), \cap, \cup, \Re^n, \emptyset, \exists_X, [t = t'])_{X \subseteq V_n; t,t' \in \tau_{Exp}}$ is a constraint system.

PROOF. [sketch] Most of this proof follows from the fact that the structure $\mathcal{LR}_n$ discussed in Example 4 is a constraint system (see [34] for details) and from an equivalent result in [41]. It is also straightforward to prove that $(\wp(\mathcal{L}), \cap, \cup, \Re^n, \emptyset)$ satisfies the axioms of closed semiring.