Providing Declarative Semantics for \textit{HH} Extended Constraint Logic Programs

Miguel García-Díaz  
miguel@sip.ucm.es

Susana Nieva  
nieva@sip.ucm.es

Dpto. Sistemas Informáticos y Programación  
Facultad de Informática  
Universidad Complutense de Madrid, Spain

\textbf{ABSTRACT}

This paper is focused on a double extension of traditional Logic Programming which enhances it following two different approaches. On one hand, extending Horn logic to hereditary Harrop formulas (\textit{HH}), in order to improve the expressive power; on the other, incorporating constraints, in order to increase the efficiency. For this combination, called \textit{HH(C)}, an operational semantics exists, but no declarative semantic for it has been defined so far.

One of the main features of (Constraint) Logic Programming is that the algorithmic behavior of (constraint) logic programs and its mathematical interpretations agree with each other, in the sense that the declarative meaning of a program can be interpreted operationally as a goal-oriented search for solutions. Both operational (algorithmic) and declarative (mathematical) semantics for programs are useful and widely developed in the frame of Logic Programming as well as in its extension, Constraint Logic Programming.

For these reasons, \textit{HH(C)} was in need of a more mathematical interpretation of programs. In this paper some fixed point semantics for \textit{HH(C)} are presented. Taking as a starting point the techniques used by Miller to interpret the fragment of \textit{HH} that incorporates intuitionistic implication in goals, we have formulated two novel extensions capable of dealing with the whole \textit{HH} logic, including universal quantifiers, as well as with the presence of constraints. Those semantics are proved to be sound and complete w.r.t. the operational semantics of \textit{HH(C)}.

\textbf{Categories and Subject Descriptors}

F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages—denotational semantics; D.3.1 [Programming Languages]: Formal Definitions and Theory—semantics; D.3.2 [Programming Languages]: Lan-

\footnote{The authors have been supported by the Spanish National Project TIC 2002-01167 \textit{MELODIAS}}

\textbf{Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. PPDP’04, August 24–26, 2004, Verona, Italy. Copyright 2004 ACM 1-58113-819-9/04/0008 ...$5.00.}

\textbf{General Terms}

Languages, theory.

\textbf{Keywords}

Hereditary Harrop formulas, constraint systems, sequent calculi, fixed point constructions.

\section{1. INTRODUCTION}

One of the main features of Logic Programming (\textit{LP}) is that, in a logic program, the operational interpretation and the mathematical (declarative) meaning agree with each other, in the sense that the declarative meaning of a program can be interpreted operationally as a goal-oriented search for solutions. In [14] the notion of abstract logic programming language is formulated as a formalization of this idea. There, the declarative meaning of a program is identified with the set of goals that can be proved from it by means of uniform proofs in a deduction system. Several logic extensions of traditional \textit{LP}, enhancing the weak expressive power of logic programs based on Horn clauses, have been proved to be abstract logic programming languages ([14, 15]). This is also the case of the language \textit{HH(C)}, on which the present paper focuses. It was introduced in [11] as a combination of the logic of Hereditary Harrop formulas (\textit{HH}) and Constraint Logic Programming (\textit{CLP}), obtaining a scheme \textit{HH(X)} that may be particularized with any constraint system \textit{C}, providing for an instance \textit{HH(C)}. This language is not only an extension of traditional \textit{LP} (based on Horn logic) improving its expressivity, but also incorporating the efficiency advantages of \textit{CLP} [8]. \textit{HH} extends Horn logic allowing disjunctions, intuitionistic implications and universal quantifiers in goals. These constructions are essential for capturing module structure, hypothetical queries and data abstraction. On the other hand, the purpose of the incorporation of the \textit{CLP} approach is to overcome the inherent limitations in dealing efficiently with elements of domains different from Herbrand terms. Satisfiability of constraints of particular domains may be checked in an efficient way, apart from the logic.
For example, in [7] a constraint solver for an interesting and useful instance of our scheme with a constraint system which combines real numbers with Herbrand terms is described.

\( HH(C) \) is proved to be an abstract logic programming language through the definition of a proof system that exclusively generates uniform proofs. This system, called \( UC \), combines inference rules from intuitionistic sequent calculus with the entailment relation of the constraint system. In addition, in [11] a goal solving procedure for the scheme \( HH(C) \) was presented and it is proved to be sound and complete w.r.t. the intuitionistic deduction system \( UC \).

That goal solving procedure could be regarded as an operational semantics of \( HH(C) \). Although an operational interpretation is needed in order to specify programs that can be executed with certain efficiency, a clear declarative semantics would indeed simplify the programmer’s work. The use of provability as declarative interpretation is still too close to the operational behavior. The deduction system, which is a syntactic tool, should be supported by model-theoretic semantics involving more abstract elements.

The aim of the present work is to settle the absence of a more declarative semantics for \( HH(C) \). The attempts to provide declarative semantics for \( LP \) languages based on mathematical foundations are extensive and fruitful (see e.g. [12, 2, 3]). This is also the case of \( CLP \) [9, 6]. In both, \( LP \) and \( CLP \), most of the studies are based on fixed point theories. The semantics we define here are inspired by the fixed point semantics for a fragment of \( HH \) described in [13].

Our purpose is to find a model such that for any program \( \Delta \), finite set of constraints \( \Gamma \) and goal \( G \), \( G \) can be proved, from \( \Delta \) and \( \Gamma \), in the deduction system \( UC \), if and only if, if \( G \) is satisfied in that model. However, in order to build such a model it is important to realize that, during the search of a proof of a goal from a program \( \Delta \) and a set of constraints \( \Gamma \), both \( \Delta \) and \( \Gamma \) may grow. Having this condition in mind, we have introduced a new notion of interpretation. A interpretation \( I \) will be a function that associates to every pair \((\Delta, \Gamma)\) a set of “true” atoms, in such a way that, if \( \Delta \) or \( \Gamma \) are augmented, the set of true atoms that \( I \) associates to the augmented pair cannot decrease. The model we are looking for will be the least fixed point of a continuous operator that transforms such kind of interpretations.

The main difference between the two semantics we provide is the way in which constraint interpretation is dealt with. For the first one, the denotation of constraints is given in terms of the entailment relation of the constraint system. For the second, a class of constraint systems is considered for which the logical inference based on classical model theory can be used to interpret constraints. The fixed point semantics is reformulated incorporating the logical inference for constraints, instead of the entailment relation. That way, the sets of constraints are replaced by their denotations and the interpretations are applied to pairs \((\Delta, \nu)\), where \( \nu \) is an assignment of the variables into the constraint domain.

The rest of this paper is organized as follows: Section 2 gathers the syntactic aspects of \( HH(C) \), such as the syntax of the constraints, programs and goals, as well as the definition of the deduction relations \( \vdash_C \) and \( \vdash_{UC} \). Some examples of the use of \( HH(C) \) as logic programming language are also shown. Section 3 contains the main new results of the current work. Two fixed point semantics for \( HH(C) \) are presented and they are proved to be sound and complete w.r.t. provability in \( UC \). In order to improve the readability of the paper, some proofs have been omitted or sketched. In Section 4 related works are commented and we summarize future research lines.

2. SYNTAX OF \( HH(C) \)

\( HH(C) \) can be regarded as a constraint logic programming language, not founded in Horn logic, as usual, but in the extended logic of hereditary Harrop formulas [11]. As most \( CLP \) languages, it is in fact a parameterized scheme that can be instantiated by particular constraint systems. The requirements imposed to such generic constraint systems are gathered below.

2.1 Constraint Systems

Given a signature \( \Sigma \) containing constants, function symbols and predicate symbols including the equality predicate \( \approx \), a constraint system \( C \) over \( \Sigma \) is a pair \((L_C, \vdash_C)\), where \( L_C \) is the set of formulas that play the role of constraints, and \( \vdash_C \subseteq \mathcal{P}(L_C) \times L_C \) is the entailment or deduction relation between sets of constraints \( \Gamma \) and constraints \( C \). \( C \) must fulfill the following conditions:

- \( L_C \) is set of first-order formulas built up using the signature \( \Sigma \), which must specifically include \( \top \) (true), \( \bot \) (false), and the equations \( t \approx t' \) for any \( \Sigma \text{-terms} t \text{ and } t' \).
- \( L_C \) is closed under \( \land, \Rightarrow, \exists, \forall \) and the application of substitutions of terms for variables.
- \( \vdash_C \) is compact, i.e., \( \Gamma \vdash_C C \) iff \( \Gamma_0 \vdash_C C \) for some finite \( \Gamma_0 \subseteq \Gamma \). \( \vdash_C \) is also generic, i.e., \( \Gamma \vdash_C C \) implies \( \Gamma \vdash_C C \sigma \) for any substitution \( \sigma \).
- All the inference rules related to \( \land, \Rightarrow, \exists, \forall \) and \( \approx \) valid in the intuitionistic fragment of first-order logic are also valid in \( \vdash_C \).

The preceeding conditions are minimal requirements for a \( C \) to be a constraint system, but in many useful cases \( C \) satisfies additional properties. For instance, if \( Ax_{CF\Gamma} \) is Smolka and Treinen’s axiomatization of the domain of feature trees [16], the constraint system \( CF\Gamma \) can be defined considering the whole set of first-order formulas as constraints, and \( \Gamma \vdash_{CF\Gamma} C \) iff \( \Gamma \cup Ax_{CF\Gamma} \vdash C \), where \( \vdash \) is the entailment relation of classical first-order logic with equality. Another example is the constraint system \( RH \), that can be defined analogously to \( CF\Gamma \), but using \( Ax_R \), the Tarski’s axiomatization of the closed field of real numbers [17]. See also the system \( RH \), that combines the field of real numbers with finite trees, defined in [7].

Hereafter, we will consider a fixed signature \( \Sigma \) and a constraint system \( C \) over \( \Sigma \). \( \Gamma \) will stand for finite sets of constraints. A set of constraints \( \Gamma \) is said to be \( C \text{-satisfiable} \) if \( \emptyset \vdash_C \exists \Gamma \), where \( \exists \) denotes existential closure and \( \Gamma \) the conjunction of constraints in \( \Gamma \).

Constraints can be found embedded in goals and clauses as described in the following subsection.

\[\text{Here and in the rest of the paper, given a set } S, \mathcal{P}(S) \text{ denotes its power set.}\]

\[\text{\( \Gamma \sigma \) is the result of applying the substitution } \sigma \text{ to each formula in } \Gamma, \text{ avoiding the capture of variables.}\]
2.2 Clauses and goals

Let the set of program predicate symbols \( \Pi_P \) be a set of predicate symbols such that \( \Sigma \cap \Pi_P = \emptyset \). In the rest of the paper \( \Sigma \) and \( \Pi_P \) are assumed fixed. Let \( At \) be the set of atomic formulas over \( \Pi_P \) and \( \Sigma \)-terms. The set \( G \) of goals \( G \), and the set \( D \) of clauses \( D \) over \( \Sigma \) and \( \Pi_P \) are defined by the mutually-recursive rules below:

\[
G ::= A \mid C \mid G_1 \land G_2 \mid G_1 \lor G_2 \mid D \Rightarrow G \mid C \Rightarrow G \mid \exists x G \mid \forall x G,
\]

\[
D ::= A \mid G \Rightarrow A \mid D_1 \land D_2 \mid \forall x D,
\]

where \( A \in At \), \( C \in \text{LC} \).

Definition 1. A program, noted \( \Delta \), over \( \Sigma \) and \( \Pi_P \) is a finite subset of \( D \).

Let \( W \) be the set of programs over \( \Sigma \) and \( \Pi_P \).

The following definition will be useful in order to simplify the usage of program clauses.

Definition 2. Given a set of clauses \( S \), the elaboration of \( S \) is the set of clauses \( \text{elab}(S) \) where \( \text{elab}(D) \) is defined by the following rules:

\[
\text{elab}(A) \overset{\text{def}}{=} \{ \top \Rightarrow A \}.
\]

\[
\text{elab}(D_1 \land D_2) \overset{\text{def}}{=} \text{elab}(D_1) \cup \text{elab}(D_2).
\]

\[
\text{elab}(G \Rightarrow A) \overset{\text{def}}{=} \{ G \Rightarrow A \}.
\]

\[
\text{elab}(\exists x D) \overset{\text{def}}{=} \{ \exists x D' \mid D' \in \text{elab}(D) \}.
\]

An elaborated clause is a clause of the form \( \forall x \exists \overline{x} (G \Rightarrow A) \).

In order to simplify the notation, in this paper we will identify a program with its elaboration. And we will write \( \Delta \), to refer to \( \text{elab}(\Delta) \). In this way, programs can be understood as sets of elaborated clauses.

A variant of \( \forall \overline{x} (G \Rightarrow A) \) is a clause \( \forall \overline{x} \exists \overline{y} (G \Rightarrow A) \), where no \( y \in \overline{y} \) occurs in \( G \Rightarrow A \). \( F[\overline{y}/\overline{x}] \) is the result of applying to \( F \) the substitution that replaces \( x_i \) by \( y_i \) for each \( x_i \in \overline{x} \).

2.3 The proof system

We follow the ideas of Miller et al. [14], in which logic programming languages are identified with those such that non-uniform proofs of goals in a deduction system can be discarded. Those languages are called abstract logic programming languages. The goal solving procedure of any of these languages and its respective deduction system agree, and in both (goal solving and deduction system) the search of a proof for a goal is directed by the structure of such goal. For the case of \( HH(C) \), the calculus that guarantees uniform proofs, called \( UC \), was defined in [11], and it is briefly described now.

\( UC \) is a sequent calculus which combines intuitionistic rules for the logic connectives with the entailment relation \( \vdash_C \). For any program \( \Delta \), finite set of constraints \( \Gamma \), and goal \( G \), \( \Delta, \Gamma \vdash_{UC} G \) means that there is a proof for the sequent \( \Delta, \Gamma \vdash G \) using, in a bottom-up fashion, the rules of the calculus \( UC \) that appear below. So \( UC \)-proofs will be regarded as trees.

\( \forall \overline{x} \overline{y} \) is an abbreviation for \( \forall x_1 \ldots \forall x_n \), and analogously for \( \exists \overline{x} \).

U-C-Rules

Rules for constraints and atomic goals:

\[
\frac{\Delta, \Gamma \vdash_C C}{\Delta, \Gamma \vdash \exists \overline{x} (A \approx A' \land G)} \quad (CR)
\]

\[
\frac{\Delta, \Gamma \vdash \exists \overline{x} (A \approx A' \land G)}{\Delta, \Gamma \vdash A} \quad (\text{Clause})
\]

where \( \exists \overline{x} (G \Rightarrow A') \) is a variant of some clause in \( \Delta \); the variables of \( \overline{x} \) do not occur free in the lower sequent; \( A \equiv P(t_1, \ldots, t_n) \), \( A' \equiv P(s_1, \ldots, s_n) \), and \( A \approx A' \) denotes the conjunction \( t_i \approx s_i \land \cdots \land t_n \approx s_n \).

Rules introducing connectives:

\[
\frac{\Delta, \Gamma \vdash G_1 \land G_2}{\Delta, \Gamma \vdash G_1 \land G_2} \quad (\land_R)
\]

\[
\frac{\Delta, \Gamma \vdash G_1 \land G_2}{\Delta, \Gamma \vdash G} \quad (\land_L)
\]

\[
\frac{\Delta, \Gamma \vdash G}{\Delta, \Gamma \vdash C \Rightarrow G} \quad (\Rightarrow_R)
\]

\[
\frac{\Delta, \Gamma \vdash C \Rightarrow G}{\Delta, \Gamma \vdash \exists \overline{x} G} \quad (\exists_R)
\]

\[
\frac{\Delta, \Gamma \vdash \exists \overline{x} G}{\Delta, \Gamma \vdash \forall \overline{x} G} \quad (\forall_R)
\]

In rules (\exists_R) and (\forall_R) the variable \( y \) does not occur free in any formula of the lower sequent, and \( i \in \{1, 2\} \) in rule (\forall_R).

When \( \Delta, C \vdash_{UC} G \) holds, \( C \) is said to be a correct answer constraint for \( G \) from \( \Delta \). In [11] a goal solving procedure for \( HH(C) \) is introduced and proved to be sound and complete w.r.t the deducibility \( \vdash_{UC} \).

2.4 Examples

One of the outstanding features of the logic programming language \( HH(C) \) is its high expressive power. In order to illustrate it, a couple of examples is presented here, for the instance \( HH(\mathbb{R}) \).

Example 1. Taking \( \Pi_P = \{ \text{triangle}, \text{isosceles} \} \), let us consider the program \( \Delta_1 \), in a prolog-like notation, enriched with constraints and the logic connectives \( \land \) and \( \Rightarrow \):

\[
\text{triangle}(A, B, C) \Leftarrow A > 0, B > 0, C > 0, A < C + B, B < A + C, C < A + B.
\]

The predicate \( \text{triangle}(A, B, C) \) becomes true when it is possible to build a triangle with sides of lengths \( A, B \) and \( C \). Let \( \Delta_2 \) be the program:

\[
\text{isosceles}(A, A, C) \Leftarrow \text{triangle}(A, A, C).
\]

\[
\text{isosceles}(A, B, A) \Leftarrow \text{triangle}(A, B, A), A \neq B.
\]

\[
\text{isosceles}(A, B, B) \Leftarrow \text{triangle}(A, B, B), A \neq B.
\]

Suppose that, from \( \Delta_1 \), we want to know which conditions over a variable \( y \) guarantee that, for any \( x > 1 \), it is possible to build an isosceles triangle with sides \( x, y, \delta \). \( \Delta_1 \) must import the clauses of \( \Delta_2 \), and the goal which captures that query is:

\[
G \equiv (\Delta_2 \Rightarrow \forall x (x > 1 \Rightarrow \text{isosceles}(x, x, y))).
\]

where \( \Delta_2 \) means here the conjunction of its clauses. Similarly as in [13], the first implication of \( G \) leads up to use program \( \Delta_2 \) as a module. In fact, the clauses of \( \Delta_2 \) will be locally added to \( \Delta_1 \) when solving \( G \) from \( \Delta_1 \). Notice that such goal cannot be written in \( CLP \) languages based on Horn clauses, because the connectives \( \Rightarrow \) and \( \forall \) would not be allowed in goals. Given the program \( \Delta_1 \) and the goal \( G \), according to the proof system \( UC \), \( C \equiv 0 < y \land y \leq 2 \) is a correct answer constraint for \( G \) from \( \Delta_1 \).
Example 2. Consider the program in Figure 1, borrowed from [11]. It is a reversible program to compute Fibonacci numbers. For instance, both the goals \( \text{fib}(9, x) \) or \( \text{fib}(x, 55) \) can be solved, obtaining the constraint answers \( x \approx 55 \) and \( n \approx 9 \), respectively. Reversibility is also obtained in a pure CLP version, but with a program that runs in exponential time and that recalculates Fibonacci numbers. The HH(R) version is more efficient since none Fibonacci number must be recalculated, and goals of the form \( \text{fib}(n, x) \), \( n \) given, run in linear time.

The goal \( \text{getfib}(n, x, m) \) computes the \( n \)-th Fibonacci number in \( x \), assuming that the Fibonacci numbers \( \text{fib}_i \), with \( 0 \leq i \leq m \), are stored in the local program as atoms for \text{memfib}. During the computation, atoms \text{memfib} for \( \text{fib}_i \), with \( m < i \leq n \), are locally memorized.

Other examples can be found in [11, 10, 7]. The ones in [10] belong to the higher-order version of HH(C), and those in [7] to the instance HH(RH).

3. SEMANTICS FOR HH(C)

The goal solving procedure defined in [11] may be regarded as an operational semantics for HH(C). However, from the theoretical point of view, the programming language HH(C) presented lacks a model semantics. The only meanings that we may associate to programs, so far, are sets of proofs.

In this section, alternative semantics based on fixed point constructions —widely utilized in LP and CLP— are introduced.

3.1 A fixed point semantics

For the traditional LP language, given a program \( P \) there is a continuous operator \( T_P \) transforming interpretations (sets of atoms) such that \( G \) can be proved from \( P \), if and only if, \( G \) is true in the least fixed point of \( T_P \) [19]. As analyzed in [13], for the fragment of HH that includes implications in goals, the situation is more complex, since while building a proof for a goal \( G \) the program \( \Delta \) may be augmented. Therefore programs play the role of contexts, and interpretations become monotonous functions mapping each program into a set of sets. Instead of a family \( \{ I_{\Delta} \}_{\Delta \in \mathcal{D}_G} \) of continuous operators, there is a unique operator \( I \), and the main result is that \( G \) can be proved from \( \Delta \), if and only if, \( G \) is true in the least fixed point of \( I \) at the context \( \Delta \). New difficulties arise extending this approach to HH(C), since the universal quantifier, as well as constraints, are allowed in goals, and then embedded into programs. When proving a goal \( G \) from a program \( \Delta \) there is also the presence of a set of constraints \( \Gamma \); both \( \Delta \) and \( \Gamma \) may result augmented, therefore the notion of context is extended to pairs \( (\Delta, \Gamma) \). So an interpretation of \( \Delta \) and \( \Gamma \) should depend on interpretations of \( (\Delta', \Gamma') \) with \( \Delta' \subseteq \Delta \), \( \Gamma' \subseteq \Gamma \).

3.1.1 Interpretations and forcing relation

We have extended the model theory presented in [13] in order to interpret full HH(C). The semantics there defined is based on a forcing relation between programs and goals that represents whether an interpretation makes true a goal in the context of a program. For the reasons explained before, in our language contexts must be extended to be pairs \( (\Delta, \Gamma) \), and interpretations are defined as monotonous functions able to interpret every pair \( (\Delta, \Gamma) \).

Let us assume that \( \Sigma, \Pi_P, \) a \( \Sigma \)-constraint system \( C \) and a set \( \Pi_P \) of program predicates have been chosen.

Definition 3. An interpretation \( I \) is a monotonous function \( I : W \times \mathcal{P}(\mathcal{C}_L) \longrightarrow \mathcal{P}(A_R) \), i.e., for any \( \Delta_1, \Delta_2 \) and \( \Gamma_1, \Gamma_2 \) such that \( \Delta_1 \subseteq \Delta_2 \) and \( \Gamma_1 \subseteq \Gamma_2 \), \( I(\Delta_1, \Gamma_1) \subseteq I(\Delta_2, \Gamma_2) \) holds. Let \( I \) denote the set of interpretations.

A continuous operator transforming such interpretations will be defined and proved that for any \( \Delta, \Gamma \) and \( G, \Delta; \Gamma \vdash_C G \) if and only if \( G \) is forced by the least fixed point of this operator at the context \( (\Delta, \Gamma) \).

The definition of such operator is founded on previous concepts and results, that are formulated now.

Definition 4. For any \( I_1, I_2 \in I \), \( I_1 \subseteq I_2 \) if for each \( \Delta \) and \( \Gamma \), \( I_1(\Delta, \Gamma) \subseteq I_2(\Delta, \Gamma) \) holds.

It is straightforward to check that \( (I, \subseteq) \) is a poset, i.e. \( \subseteq \) is a partial order. In addition, \( (I, \supseteq) \) is a complete lattice. It is easy to prove that, for any \( S \subseteq I \), the least upper bound and the greatest lower bound of \( S \), denoted by \( \bigcup S \) and \( \bigcap S \) respectively, exist, and they are characterized by the following equations:

\[
(\bigcup S)(\Delta, \Gamma) = \bigcup_{I \in S} I(\Delta, \Gamma) \text{ for any } \Delta \text{ and } \Gamma,

(\bigcap S)(\Delta, \Gamma) = \bigcap_{I \in S} I(\Delta, \Gamma) \text{ for any } \Delta \text{ and } \Gamma.
\]

As a particular case, \((I, \subseteq)\) has an infimum \( \bigcap I \), denoted by \( I_0 \), which is the constant function \( \emptyset \). Moreover, for any chain of interpretations \( \{I_i\}_{i \geq 0} \), such that \( I_0 \subseteq I_1 \subseteq I_2 \subseteq \ldots \subseteq I_{n} \), \( I_n(\Delta, \Gamma) \) is the least upper bound of \( \bigcup_{i \geq 0} I_i(\Delta, \Gamma) \).

The following definition formalizes the notion of a goal \( G \) being “true” for an interpretation \( I \) in a context \( (\Delta, \Gamma) \).

Definition 5. Given \( I \in I, \Delta, \Gamma \), \( G, \Delta \) and \( \Gamma, G \) is said to be forced by \( I, \Delta \) and \( \Gamma \), written \( I, \Delta, \Gamma \vdash G \), where \( \vdash \) is the relation recursively defined by the rules below:

\[
\begin{align*}
& I, \Delta, \Gamma \vdash C \overset{\text{def}}{\Rightarrow} \Delta \vdash_\mathcal{C} C, \\
& I, \Delta, \Gamma \vdash A \overset{\text{def}}{\Rightarrow} I(\Delta, \Gamma), \\
& I, \Delta, \Gamma \vdash G_1 \land G_2 \overset{\text{def}}{\Rightarrow} I, \Delta, \Gamma \vdash G_i \text{ for each } i \in \{1, 2\}, \\
& I, \Delta, \Gamma \vdash G_1 \lor G_2 \overset{\text{def}}{\Rightarrow} I, \Delta, \Gamma \vdash G_i \text{ for some } i \in \{1, 2\}, \\
& I, \Delta, \Gamma \vdash D \overset{\text{def}}{\Rightarrow} G \text{ if } D \text{ is a variable } y \text{ such that:} \\
& \quad - y \text{ does not occur free in } \Delta, \Gamma, \exists G, \\
& \quad - \Delta, \Gamma \vdash_\mathcal{C} \exists y G', \\
& \quad - I, \Delta, \Gamma \cup \{C\} \vdash G[y/x]. \\
& I, \Delta, \Gamma \vdash \forall x G \overset{\text{def}}{\Rightarrow} \text{ there is a variable } y \text{ such that:} \\
& \quad - y \text{ does not occur free in } \Delta, \Gamma, \forall x G, \\
& \quad - I, \Delta, \Gamma \vdash G[y/x].
\end{align*}
\]

Now, we are ready to define the operator over interpretations whose least fixed point supplies the expected version of truth.
Theorem 5. The operator $T$ has a least fixed point, which is $\bigcup_{i \geq 0} T^i(I_\bot)$.

Proof. The claim is an immediate consequence of Lemmas 3, 4 and the Knaster-Tarski fixed point theorem [18].

From now on, $lf(T)$ denotes the least fixed point of $T$.

Example 3. Let $\Delta$ be the program in Example 2. Figure 2 shows some of the goals that are forced by the first interpretations $T^i(I_\bot)$ in the contexts $\langle \Delta, \Gamma \rangle$, where $\Gamma = \{ z_1 \approx 1, z_2 \approx 1, x \approx z_1 + z_2 \}$, $\Delta' = \Delta \cup \{ mf(0, 1), mf(1, 1) \}$ and $\Delta'' = \Delta \cup \{ mf(0, 1), mf(1, 1), mf(2, z_1 + z_2) \}$.

The chart shows the main steps leading to $T^4(I_\bot)$, $\Delta, \Gamma \vdash \text{fib}(2, x)$, $\text{memfib}(2, x)$, $\text{getfib}(0, 1)$.

This is difficult to see, if such forcing relations are verified from left to right. For instance,

$$T(I_\bot), \Delta', \Gamma \vdash \text{mf}(0, z_1)$$

is checked in one step, since $\text{mf}(0, 1) \in \Delta'$ and $\Gamma \vdash z \approx 1$.

The forcing relations in each column justify those in the next. In order to verify that an existential quantification is forced, it is required to introduce new fresh variables. However, for the sake of readability, equivalent and more simple expressions have been used instead.

Bear in mind that only some of the formulas forced are gathered in the table, as it is indicated by the ellipses. In particular, since interpretations are monotonous, any formula present in a position of such table is automatically present in the whole rectangle that has that position as top-left corner.

3.1.2 Soundness and Completeness

The following theorem establishes the full connection between the fixed point semantics presented and the calculus $\mathcal{CL}$. The definitions below correspond to technicalities that will be used in the proof of such soundness and completeness result.

Let $\mathcal{L} = \langle \Delta, \Gamma, G \rangle \in \mathcal{W} \times \mathcal{P}(\mathcal{L}_C) \times G \mid \text{lf}(T), \Delta, \Gamma \vdash G \rangle$. The function $\text{ord} : \mathcal{S} \longrightarrow \mathbb{N}$ is defined as follows. Given any $\langle \Delta, \Gamma, G \rangle \in \mathcal{S}$, Lemma 2 guarantees that the set of natural numbers $k$ such that $T^k(I_\bot), \Delta, \Gamma \vdash G$ is nonempty. Therefore, it is possible to define $\text{ord}(\Delta, \Gamma, G)$ as the least element of such set. Let us consider the partial order $(\mathcal{S}, \prec)$ defined as follows. Given any $\langle \Delta_1, \Gamma_1, G_1 \rangle, \langle \Delta_2, \Gamma_2, G_2 \rangle \in \mathcal{S}$, $\langle \Delta_1, \Gamma_1, G_1 \rangle \prec \langle \Delta_2, \Gamma_2, G_2 \rangle$ if

- $\text{ord}(\Delta_1, \Gamma_1, G_1) < \text{ord}(\Delta_2, \Gamma_2, G_2)$,

or

- $\text{ord}(\Delta_1, \Gamma_1, G_1) = \text{ord}(\Delta_2, \Gamma_2, G_2)$ and $G_1$ is a strict subformula of a goal $G'_2$, where $G'_2$ is obtained through a renaming of the free variables in $G_2$. 

$\text{fib}(N, X) :- \text{memfib}(0, 1) \Rightarrow (\text{memfib}(1, 1) \Rightarrow \text{getfib}(N, X, 1))$.

$\text{getfib}(N, X, M) :- 0 \leq N, N \leq M, \text{memfib}(N, X)$.

$\text{getfib}(N, X, M) :- N > M, \text{memfib}(M-1, F1), \text{memfib}(M, F2)$,

$(\text{memfib}(M + 1, F1 + F2) \Rightarrow \text{getfib}(N, X, M + 1))$.

Figure 1: Definition of a predicate that calculates Fibonacci numbers.
Such partial order is well-founded, because $(\mathbb{N}, <)$ is also well-founded and formulas are finite sequences of symbols.

**Theorem 6.** For any $\Delta, \Gamma$ and $G$, 
\[
\text{lfp}(T), \Delta, \Gamma \models G \iff \Delta; \Gamma \vdash_{\text{UC}} G.
\]

**Proof.** Since this is one of the main results presented, the whole proof is included.

\(\Leftarrow\) Let $h$ be the height of a UC-proof for $\Delta; \Gamma \vdash_{\text{UC}} G$. The claim is proved inductively on $h$.

- **Base case:** $h = 1$. The only possibility is that $G \equiv C \in \mathcal{L}_c$. Then $\Delta; \Gamma \vdash_{\text{UC}} C$ implies that $\Gamma \vdash C$, and therefore $\text{lfp}(T), \Delta, \Gamma \models C$ holds.

- **Inductive case.** Suppose that $\Delta; \Gamma \models G$ has a proof of height $h$. Let us prove $\text{lfp}(T), \Delta, \Gamma \models G$ by case analysis on the UC-rule employed in the bottom of such proof.

  (\text{Clause}) \quad \text{There must exist } \forall x_1, \ldots, \forall x_n(G \Rightarrow A') \in \Delta, \text{ such that the variables } x_1, \ldots, x_n \text{ do not occur free in } \Delta, \Gamma, A, \text{ and that the sequent } \Delta; \Gamma \models \exists x_1 \ldots \exists x_n(A \approx A' \land G) \text{ has a proof of height } h - 1. \text{ By induction hypothesis, } \text{lfp}(T), \Delta, \Gamma \models \exists x_1 \ldots \exists x_n(A \approx A' \land G). \text{ Using the definition of the operator } T, \text{ the latter implies } A \in (T(\text{lfp}(T)))(\Delta, \Gamma), \text{ which is equivalent to } T(\text{lfp}(T)), \Delta, \Gamma \models A. \text{ But since } T(\text{lfp}(T)) = \text{lfp}(T), \text{ the proof is complete.}

(\exists R) \quad \text{There must exist goals } G_1, G_2 \text{ such that } G \equiv G_1 \lor G_2 \text{ and the sequent } \Delta, \Gamma \models G_i \text{ has a proof of height less than } h \text{ for each } i \in \{1, 2\}. \text{ By induction hypothesis, } \text{lfp}(T), \Delta, \Gamma \models G_i \text{ is assumed for } i \in \{1, 2\} \text{ and, as a consequence, } \text{lfp}(T), \Delta, \Gamma \models G.

(\Rightarrow R) \quad \text{There must exist a clause } D \text{ and a goal } G' \text{ such that } G \equiv D \Rightarrow G' \text{ and the sequent } \Delta, D; \Gamma \models G' \text{ has a proof of height } h - 1. \text{ By induction hypothesis, } \text{lfp}(T), \Delta \cup \{D\}, \Gamma \models G'. \text{ Therefore, } \text{lfp}(T), \Delta, \Gamma \models D \Rightarrow G'.

(\Rightarrow C R) \quad \text{There must exist a } C \in \mathcal{L}_c \text{ and a goal } G' \text{ such that } G \equiv C \Rightarrow G' \text{ and the sequent } \Delta; \Gamma, C \models G' \text{ has a proof of height } h - 1. \text{ By induction hypothesis, } \text{lfp}(T), \Delta \cup \{C\} \models G'. \text{ Therefore, } \text{lfp}(T), \Delta, \Gamma \models C \Rightarrow G'.

(\exists R) \quad G \text{ must be of the form } \exists x G', \text{ and there must exist a constraint } C \text{ and a variable } y \text{ variable not occurring free in } \Delta, \Gamma, \exists x G', \text{ such that } \Delta, \Gamma, C \models G'[y/x] \text{ has a proof of height } h - 1 \text{ and } \Gamma \vdash_e \exists y C. \text{ By induction hypothesis, } \text{lfp}(T), \Delta, \Gamma \cup \{C\} \models G'[y/x], \text{ and therefore } \text{lfp}(T), \Delta, \Gamma \models \exists x G'.

(\forall R) \quad G \text{ must be of the form } \forall x G', \text{ and there must exist a variable } y \text{ not occurring free in } \Delta, \Gamma, \forall x G' \text{ such that } \Delta; \Gamma \models G'[y/x] \text{ has a proof of height } h - 1. \text{ By induction hypothesis, } \text{lfp}(T), \Delta, \Gamma \models G'[y/x] \text{ and, as a consequence, } \text{lfp}(T), \Delta, \Gamma \models \forall x G'.

\(\Rightarrow\) By induction on the structural order $(\mathcal{S}, \subset)$. Let us take $(\Delta, \Gamma, C) \in \mathcal{S}$ and assume that, for any other $(\Delta', \Gamma', G') \in \mathcal{S}$, $(\Delta, \Gamma, G') < (\Delta, \Gamma, G)$ implies that $(\Delta', \Gamma', G') \not< (\Delta, \Gamma, G)$ implies that $(\Delta; \Gamma; \vdash_{\text{UC}} G')$. Then, let us conclude $(\Delta; \Gamma; \vdash_{\text{UC}} G)$ by case analysis on the structure of $G$.

- $G \equiv C \in \mathcal{L}_c$. Then $(\Delta, \Gamma, C) \in \mathcal{S}$ implies that $\Gamma \vdash C$, and therefore $(\Delta; \Gamma; \vdash_{\text{UC}} C$ by $(C_n)$.

- $G \equiv A$. In this case, $(\Delta, \Gamma, A) \in \mathcal{S}$ implies that $\text{lfp}(T), \Delta, \Gamma \models A$. Let $k = \text{ord}(\Delta, \Gamma, A))$, and hence $T^k(\mathcal{L}_c), \Delta, \Gamma \models A$, which is equivalent to $A \in (T^k(\mathcal{L}_c))(\Delta, \Gamma)$. This implies that there is $\forall \exists(G \Rightarrow A') \in \Delta$ such that the variables $\exists G do not occur free in $\Delta, \Gamma, A$, and $T^{k-1}(\mathcal{L}_c), \Delta, \Gamma \models \forall \exists(G \Rightarrow A' \land G)$. In this reason, $(\Delta, \Gamma, \forall \exists(G \Rightarrow A' \land G)) < (\Delta, \Gamma, A)$, so the induction hypothesis can be applied, obtaining that $(\Delta; \Gamma; \vdash_{\text{UC}} \forall \exists(G \Rightarrow A' \land G)$. Using the rule (Clause) with the elaboration $\forall \exists(G \Rightarrow A')$, it follows that $(\Delta; \Gamma; \vdash_{\text{UC}} A$.

- $G \equiv G_1 \lor G_2$. Then $(\Delta, \Gamma, G_1) \in \mathcal{S}$ implies that $\text{lfp}(T), \Delta, \Gamma \models G_1$ and $\text{lfp}(T), \Delta, \Gamma \models G_2$. Clearly, $\text{ord}(\Delta, \Gamma, G_1) = \text{ord}(\Delta, \Gamma, G_2)$ and $G_1, G_2$ are strict subformulas of $G$, hence $(\Delta, \Gamma, G_1) \not< (\Delta, \Gamma, G_2)$. Then, by the induction hypothesis, $(\Delta; \Gamma; \vdash_{\text{UC}} G_1, i = 1, 2$. So $(\Delta; \Gamma; \vdash_{\text{UC}} G$, applying the rule $(\exists R)$.

- $G \equiv G_1 \lor G_2$. Then $(\Delta, \Gamma, G) \in \mathcal{S}$ implies that there is $i \in \{1, 2\}$ such that $G_1 \not< \text{lfp}(T), \Delta, \Gamma \models G_i$. Clearly, $\text{ord}(\Delta, \Gamma, G) = \text{ord}(\Delta, \Gamma, G_1)$ and $G_1$ is a strict subformula of $G$ and, as a consequence, $(\Delta, \Gamma, G_1) \not< (\Delta, \Gamma, G)$. Therefore, by the induction hypothesis we obtain $(\Delta; \Gamma; \vdash_{\text{UC}} G_1$ for some $i \in \{1, 2\}$. Thanks to the rule $(\forall R)$, it follows that $(\Delta; \Gamma; \vdash_{\text{UC}} G$.

- $G \equiv D \Rightarrow G'$. Then $(\Delta, \Gamma, G) \in \mathcal{S}$ implies that $\text{lfp}(T), \Delta \cup \{D\}, \Gamma \models G'$. Clearly, $\text{ord}(\Delta, \Gamma, G) = \text{ord}(\Delta \cup \{D\}, \Gamma, G')$ and $G'$ is a strict subformula of $G$, so $(\Delta \cup \{D\}, \Gamma, G') < (\Delta, \Gamma, G)$. Therefore, by the induction hypothesis, $(\Delta, D; \Gamma; \vdash_{\text{UC}}$
Therefore, the claim has been proved. \( \square \)

This fixed point semantics supplies a framework in which properties of programs can be easily analyzed. For instance, the behaviour two programs can be compared using the properties of programs can be easily analyzed. For instance, 

\[ G \equiv C \Rightarrow G' \]  

Then \( \langle \Delta, \Gamma, G \rangle \in S \) implies that \( \text{lp}(T, \Delta, \Gamma \cup \{C\}) \equiv G' \). Clearly, \( \text{ord}(\langle \Delta, \Gamma, G \rangle) = \text{ord}(\langle \Delta, \Gamma \cup \{C\}, G' \rangle) \) and \( G' \) is a strict subformula of \( G \), so \( \langle \Delta, \Gamma \cup \{C\}, G' \rangle < \langle \Delta, \Gamma, G \rangle \). Then, by the induction hypothesis, \( \Delta ; \Gamma ; C \vdash_{uc} G' \), and \( \Delta ; \Gamma \vdash_{uc} G \) due to the rule \( (\Rightarrow \ast_{\gamma}) \).

\( G \equiv \exists x G' \). Then \( \langle \Delta, \Gamma, G \rangle \in S \) implies that there is a constraint \( C \) and a variable \( y \) such that:

* \( y \) does not occur free in \( \Delta, \Gamma, \exists x G' \).
* \( \text{lp}(T, \Delta, \Gamma \cup \{C'\}) \equiv G'[y/x] \).

\( \text{ord}(\langle \Delta, \Gamma, G \rangle) = \text{ord}(\langle \Delta, \Gamma \cup \{C'\}, G'[y/x] \rangle) \) by definition, and \( G'[y/x] \equiv G' \) is a renaming of a strict subformula of \( G \), so \( \langle \Delta, \Gamma \cup \{C'\}, G'[y/x] \rangle < \langle \Delta, \Gamma, G \rangle \). Therefore, \( \Delta ; \Gamma ; C' \vdash_{uc} G'[y/x] \) by the induction hypothesis. Hence \( \Delta ; \Gamma \vdash_{uc} G \), by using the rule \( (\Rightarrow \ast_{\gamma}) \).

3.1.3 Models

At this stage, \( \text{lp}(T) \) has already been proved to be a sound and complete semantics with respect to \( uc \) in a sense. However, instead of having a unique model, it would be desirable to provide for a more general notion of model such that \( \Delta ; \Gamma \vdash_{uc} G \) iff \( G \) is true in the context \( \langle \Delta, \Gamma \rangle \) for every model. Such notion of model is provided below, together with the expected results.

**Definition 7.** Given an elaborated clause \( D \equiv \forall \mathbf{A} \rightarrow A \), an interpretation \( I \) is a model of \( D \), denoted by \( I \models D \), if for any \( \Delta, \Gamma \) and \( A' \in A \) such that \( D \) is a variant of a clause in \( \Delta \) and no variable \( x \in \mathbf{A} \) occurs free in \( \Delta, \Gamma, A' \), if \( I, \Delta, \Gamma \models \mathbf{A} \approx A' \) then \( A' \in I(\Delta, \Gamma) \).

Intuitively, an interpretation \( I \) is model of a clause \( D \) if, whenever \( D \) is available, \( I \) gathers all the atoms possibly inferred by using the clause \( D \).

**Definition 8.** An interpretation \( I \) is said to be a model if \( I \models D \) holds for every elaborated clause \( D \).

**Lemma 7.** For any interpretation \( I \in I \), \( I \) is a model \( \iff T(I) \subseteq I \).

**Proof.** \( T(I) \subseteq I \iff \) for any \( \Delta \) and \( \Gamma \), \( T(I)(\Delta, \Gamma) \subseteq I(\Delta, \Gamma) \iff \) for any \( \Delta, \Gamma, A \) and any variant \( \forall \mathbf{A} \rightarrow A' \) of a clause in \( \Delta \) such that the variables \( \mathbf{A} \) do not occur free in \( \Delta, \Gamma, A \), if \( I, \Delta, \Gamma \models \mathbf{A} \approx A' \) then \( A' \in I(\Delta, \Gamma) \). However, by Definition 8, this is equivalent to say that \( I \models D \) for every elaborated clause \( D \), i.e., \( I \) is a model. \( \square \)

**Lemma 8.** For any \( I \in I \), if \( T(I) \not\subseteq I \) then \( \text{lp}(T) \not\subseteq I \).

**Proof.** Since \( I \subseteq I \), by the monotonicity of \( T \), \( T^n(I) \subseteq T^{n+1}(I) \) holds. Therefore \( \bigcup_{i \geq 0} T^i(I) \subseteq \bigcup_{i \geq 0} T^{i+1}(I) \), which means that \( \text{lp}(T) \subseteq \bigcup_{i \geq 0} T^i(I) \). Since by hypothesis \( T(I) \subseteq I \), again by monotonicity of \( T \) it follows that \( T^{i+1}(I) \subseteq T^i(I) \) for any \( i \geq 0 \), i.e. we have the chain \( I \supseteq T(I) \supseteq T^2(I) \supseteq \ldots \). Therefore, \( \bigcup_{i \geq 0} T^i(I) = I \), and thus the proof for \( \text{lp}(T) \subseteq I \) is complete. \( \square \)

Finally, the expected result can be stated:

**Theorem 9.** For any \( \Gamma, \Delta, \Gamma \models G \), 

\[ \Delta ; \Gamma \vdash_{uc} G \iff I, \Delta, \Gamma \models G \text{ holds for every model } I \]

**Proof.** \( I, \Delta, \Gamma \models G \) for every model \( I \iff I, \Delta, \Gamma \not\models G \) for every \( I \) such that \( T(I) \not\subseteq I \), thanks to Lemma 7 \( \iff \text{lp}(T, \Delta, \Gamma \models G \text{ from Lemmas 8 and 1} \iff I, \Delta, \Gamma \vdash_{uc} G \), by virtue of Theorem 6. \( \square \)

3.2 Incorporating semantic structures to interpret constraints

We have just described a fixed point semantics for \( HH(C) \). In it, the constraint system has been used as a black box, through the entailment relation \( \vdash_{uc} \), which is a syntactic tool. See, for example, the cases \( C \) and \( \exists x G \) of Definition 5. This semantics is defined for any general constraint system \( C \). The conditions imposed in Section 2 are meant as minimal requirements for a \( C \) to be a constraint system, but in many useful cases \( C \) satisfies additional properties, as it was mentioned in Subsection 2.1. For instance, the entailment relation \( \vdash \) referred in it is known to be sound and complete
w.r.t. the standard semantic relation $|=_{s}$ of first-order logic with equality, hence, for the instance $HH(\mathcal{R})$, requirements like $\Gamma \vdash_{K} C$ can be directly replaced by $\Gamma \cup Ax_{K} \vdash_{s} C$, in the definition of the forcing relation.

As in the frame of CLP, we are interested in finding general conditions for the constraint systems that would guarantee the existence of semantics for constraints based on a model theory, in order to incorporate it into the fixed point semantics of logic programs.

More precisely, the semantics of constraint logic programs are usually based on the assumption that: the domain of computation (model), which is the structure used to interpret the constraints; the solver, which checks whether constraints are $\mathcal{C}$-satisfiable; and the constraint theory, that describes the logical semantics of the constraints, agree. See [9] for details.

From now on we will focus on constraint systems $\mathcal{C}$ for which an additional condition is required: there is a standard structure $\mathcal{A}_{c}$ such that $\vdash_{\mathcal{C}}$ and $\mathcal{A}_{c}$ agree in a sense, similar to that of [9], that will be specified soon. Additional notation involving standard structures is now introduced for that purpose.

Given a standard structure $\mathcal{A}_{c}$ over a signature $\Sigma$, which interprets the symbols of $\Sigma$, and with domain $\mathcal{A}_{c}$, an assignment (for $\mathcal{A}_{c}$) is a function $\nu : V \rightarrow \mathcal{A}_{c}$, where $V$ is a set of variables. $\nu = dom(\nu)$ is said to be the domain of $\nu$.

$\text{Assig}$ is the set of assignments. Given $\nu \in \text{Assig}$, a variable $y \notin \text{dom}(\nu)$ and $a \in \mathcal{A}_{c}$, $\nu[y \leftarrow a]$ is the assignment with domain $\text{dom}(\nu) \cup \{y\}$ such that

$$
\nu[y \leftarrow a](x) = \begin{cases} a, & \text{if } x = y \\ \nu(x), & \text{otherwise}, \end{cases}
$$

and it is said to be an extension of $\nu$ to $y$.

Given a first-order formula $F$ over $\Sigma$, $\{F\}_{c}^{\nu} \in \{0, 1\}$ is the classical truth value of the formula $F$ in the model $\mathcal{A}_{c}$ under the assignment $\nu$.

**Definition 9.** Let $\mathcal{C}$ be a constraint system and $\mathcal{A}_{c}$ be a standard structure over $\Sigma.$ $\mathcal{A}_{c}$ and $\vdash_{\mathcal{C}}$ agree if for any $\Gamma$, $\nu$ and $C$, $\Gamma \vdash_{\mathcal{C}} C$ if and only if $\nu \models [C]_{\mathcal{A}_{c}}$.

Intuitively, this means that the entailment in the constraint system can be identified with (the universal closure of) the implication in that specific structure.

Constraints will be interpreted by the sets of assignments (for such $\mathcal{A}_{c}$) that make them true. Formally, given a constraint $C$, the set $[C]$ is defined as follows:

$$
[C] = \{ \nu \in \text{Assig} \mid \text{dom}(\nu) \supseteq \text{free}(C) \text{ and } [C]_{\nu} = \text{true}\}.
$$

Such definition is extended to finite sets of constraints in the natural way, i.e. $[\Gamma] = \big[\big[\Gamma\big]\big]$. Furthermore, in some cases it will be necessary that the domains of such assignments include specific sets of variables, and in this reason we define:

$$
[\Gamma]_{V} = \{ \nu \in [\Gamma] \mid \text{dom}(\nu) \supseteq V \},
$$

where $V$ is any set of variables.

Notice that if $\mathcal{A}_{c}$ and $\vdash_{\mathcal{C}}$ agree, then $\Gamma \vdash_{\mathcal{C}} C \iff [\Gamma]_{\text{free}(C)} \subseteq [C]$, and that $\Gamma$ is $\mathcal{C}$-satisfiable iff $[\Gamma] \neq \emptyset$.

For instance, $\mathcal{A}_{R}$ be the $\Sigma$-structure whose domain is $\mathcal{R}$ and that interprets constants for real numbers and arithmetic symbols in the natural way. Then $\mathcal{A}_{R}$ and $\mathcal{R}$ agree.

**Example 5.** Consider $\mathcal{C} = \mathcal{R}$ and the $\Sigma$-structure above. If $\mathcal{C} \models x * x + y * y \approx 1$, then $[C] = \{ \nu : (x, y) \rightarrow \mathbb{R}^{2} \mid (x^2 + y^2) = 1 \}$. Once each variable is associated to a coordinates axis, this can be assimilated to the set $\{ (x, y) \in \mathbb{R}^{2} \mid x^2 + y^2 = 1 \}$, the circle of radius 1 centered in the origin of the real plane. Thus, the syntactic object $x * x + y * y \approx 1$ is replaced in the forcing relation by such circle, which is its intended meaning in $\mathcal{A}_{R}$.

The new semantics we will present combines a notion of forcing similar to that used in Subsection 3.1 with the classical structure considered for $\mathcal{C}$. New definitions are needed for the concepts of forcing and interpretation.

The notion of $\mathcal{C}$-interpretation of the algebraic semantics provided in [9] associates sets of expressions of the form $p(a_1, \ldots, a_n)$ to programs, where each $a_i$ belongs to the domain of the structure, and $p$ is a program predicate symbol. The approach followed in this paper is close to that, because our $\mathcal{C}$-interpretations associate $(\Delta, \nu)$ atoms with free variables to pairs, which can be assigned to elements of the domain of the structure via $\nu$.

### 3.2.1 $\mathcal{C}$-interpretations and guided forcing relations

As in the case of $\equiv$, we are looking for a model $\mathcal{I}_{c}$ and a relation $\equiv_{c}$ such that $\Delta; \Gamma \vdash_{\mathcal{C}} G$ if $\mathcal{I}_{c}, \Delta, [\Gamma] \equiv_{c} G$. Some technicalities, needed in the proof of such result, will be promptly presented. In fact, the equivalence between $\equiv_{c}$ and $\vdash_{\mathcal{C}}$ is proved connecting $\equiv_{c}$ with $\equiv$, and using the equivalence between $\equiv$ and $\vdash_{\mathcal{C}}$. But such connection is established defining two guided versions of these forcing relations, denoted by $\equiv_{c}^{\tau}$ and $\equiv_{c}^{\tau}$, respectively, where $\tau$ is an index that play the role of guide.

The introduction of those forcing relations demands the definition and manipulation of several notions of interpretations and fixed point operators. The Figure 3 may help to identify the notation and to understand the connection between the different induced semantics.

The index $\tau$ of the guided versions is closely related to the structure of goals. The formal definition is the following:

**Definition 10.** The set of structural trees $T$, with elements $\tau$, is recursively defined by the rule:

$$
\tau ::= \text{est} \mid \text{and}(\tau_1, \tau_2) \mid \text{or}(i, \tau) \mid \text{imp}(\tau) \mid \text{imp}(\tau) \mid c(n, \tau) \mid \text{exists}(\tau) \mid \text{forall}(\tau),
$$

where $i \in \{1, 2\}$ and $n \in \mathbb{N}$.

$T^{\mathcal{C}} \subseteq T$ is the set of trees with the form $c(n, \tau)$, where $\tau \in T$ and $n \in \mathbb{N}$.

A new notion of interpretation is provided, because now context are not pairs $(\Delta, \Gamma)$, but pairs $(\Delta, \nu)$. On the other

<table>
<thead>
<tr>
<th>Interpretations</th>
<th>$\mathcal{I}_{c}$-interpretations</th>
</tr>
</thead>
<tbody>
<tr>
<td>non guided</td>
<td>$(\mathcal{I}, \equiv)$</td>
</tr>
<tr>
<td></td>
<td>$\uparrow$</td>
</tr>
<tr>
<td></td>
<td>Corollary 11</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$</td>
</tr>
<tr>
<td></td>
<td>Definition 14</td>
</tr>
<tr>
<td>guided</td>
<td>$(\mathcal{I}<em>{c}, \equiv</em>{c})$</td>
</tr>
</tbody>
</table>

**Figure 3:** Different fixed point semantics

For instance, $\mathcal{A}_{R}$ be the $\Sigma$-structure whose domain is $\mathcal{R}$ and that interprets constants for real numbers and arithmetic symbols in the natural way. Then $\mathcal{A}_{R}$ and $\mathcal{R}$ agree.
hand, another remarkable difference arises: in the range of the interpretations, each atom is tagged with a tree of $\mathcal{T}^C$.

**Definition 11.** A $C$-interpretation $I^C$ is a function $I^C : W \times \text{Assig} \to \mathcal{P}(A \times \mathcal{T}^C)$ that is monotonous, i.e. for any $\Delta_1, \Delta_2$ and $\nu_1, \nu_2$, if $\Delta_1 \subseteq \Delta_2$ and $\nu_1 = \nu_2|_{\text{dom}(\nu_2)}$ then $I^C(\Delta_1, \nu_1) \subseteq I^C(\Delta_2, \nu_2)$. Moreover, $\langle A, cl(n, \tau) \rangle \in I^C(\Delta, \nu)$ implies that free($\Delta \cup \{A\}$) $\subseteq \text{dom}(\nu)$. Let $\mathcal{T}^C$ be the set of these $C$-interpretations.

A preorder $\sqsubseteq$ can be defined for $\mathcal{T}^C$ similar to such of $I$, $(\mathcal{T}^C, \sqsubseteq)$ is a complete lattice and his infimum, denoted $I_{\sqsubseteq}^C$ is the constant function $\emptyset$.

The guided forcing relation $\models^C_\tau$ is defined from the concept of $C$-interpretation.

**Definition 12.** Given $G$, $I^C \in \mathcal{T}^C$, $\Delta$ and $\nu$ such that free($\Delta \cup \{G\}$) $\subseteq \text{dom}(\nu)$, $G$ is said to be forced by $I^C$, $\Delta$ and $\nu$ with the guide $\tau$, written $I^C, \Delta, \nu \models^C_\tau G$, where $\models^C_\tau$ is the relation recursively defined by the rules below.

- $I^C, \Delta, \nu \models^C_\tau \text{const} \iff \nu \in [C]$.
- $I^C, \Delta, \nu \models^C_{cl(n, \tau)} \iff \langle A, cl(n, \tau) \rangle \in I^C(\Delta, \nu)$.
- For each $i \in \{1, 2\}$, $I^C, \Delta, \nu \models^C_{cl(n, r_1, r_2)} G_1 \land G_2 \iff I^C, \Delta, \nu \models^C_{r_2} G_1$ and $I^C, \Delta, \nu \models^C_{r_1} G_2$.
- $I^C, \Delta, \nu \models^C_{\text{imp}(\tau)} D \iff G \models^C D, \Delta \cup \{D\}, \nu \models^C G$.
- $I^C, \Delta, \nu \models^C G \iff \exists x \in [C]$ or $I^C, \Delta, \nu \not\models^C G$.
- $I^C, \Delta, \nu \models^C_{\exists x(G)} \iff \exists x \models^C G[y/x]$.
- $I^C, \Delta, \nu \models^C_{\forall y(G)} \iff \forall y \not\models^C G[y/x]$.

From this definition it is followed that the label $\tau$ is narrowly connected with the structure of the goal and with the clauses used to prove it.

Intuitively, the subscript $\tau$ in $\models^C_\tau$ plays the role of guide, fixing the choice for the case when the goal is a disjunction or an atom.

This notion must be extended to sets of assignments, which henceforth are denoted by $\Theta$, as follows.

**Definition 13.** Given $I^C, \Delta, G, \tau$ and a set $\Theta$ of assignments, $I^C, \Delta, \Theta \models^C_\tau G$ if $I^C, \Delta, \nu \models^C_\tau G$ for each $\nu \in \Theta$.

Now the non guided forcing relation $\models^C$ can be defined:

**Definition 14.** Given $\Delta, I^C$ and $\Theta \subseteq \text{Assig}$, a goal $G$ is said to be forced by $I^C$, $\Delta$ and $\Theta$, written $I^C, \Delta, \Theta \models^C G$, if there exists $\tau$ such that $I^C, \Delta, \Theta \models^C_\tau G$.

The particular model we are looking for, establishing the connection between $\models^C$ and $\models^C_\tau$, will be defined as the least fixed point of the operator defined over $C$-interpretations defined below. In this definition, and in the rest of the paper, let us assume that, any program $\Delta$ has its clauses ordered by an enumeration, and when a clause is added to $\Delta$, it will be the last one in the order. We will frequently refer to the $n^{th}$ clause of $\Delta$ according to that enumeration.

**Definition 15.** The operator $T^C : \mathcal{T}^C \to \mathcal{T}^C$ transforms $C$-interpretations as follows. For any $I^C \in \mathcal{T}^C$, $\Delta$, $\nu$, $\tau$ and $A$ such that free($\Delta \cup \{A\}$) $\subseteq \text{dom}(\nu)$, $\langle A, cl(n, \tau) \rangle \in I^C(\Delta, \nu)$ if

- $\forall \theta \models^C \Delta, \nu \models^C_\tau G$ is a variant $D$ of the $n^{th}$ clause of $\Delta$ and, for each $x \in \mathcal{T}$, $x \notin \text{dom}(\nu)$.
- $I^C, \Delta, \nu \models^C_\tau \exists x \in [C]$.

Notice that the subscript $\tau$ fixes which clause may be used to prove that an atom is forced. Therefore, in order to check $I^C, \Delta, \nu \models^C_\tau G$ for some $I^C, \Delta, \nu, \tau$ and $G$, no choice is possible, since all of them are gathered in $\tau$.

The operator $T^C$ is proved to be monotonous and continuous. Such proofs are analogous to those for $T$ (Lemmas 3 and 4). Therefore, $T^C$ has a least fixed point, which is $\bigcup_{\tau \in \mathcal{C}} T^C(I^C(\Delta, \nu))$, and we denote it by $T^C(I^C(\Delta, \nu))$.

The following examples intend to motivate and illustrate the behavior of the operator $T^C$, as well as the meaning of programs w.r.t. $\models^C_\tau$.

**Example 6.** This is a very simple example showing the necessity of $\tau$ in the definition of $T^C$ and $\models^C_\tau$. Choosing the constraint system $\mathcal{R}$ and the structure $\mathcal{A}_R$, let us consider the program $\Delta$ in Example 4 and the goal $G \equiv \forall y p(y)$. Notice that $\Delta, \emptyset \not\models^C_\tau G$. Let us suppose that no $\tau$ was used, and so interpretations map pairs $(\Delta, \nu)$ to sets of atoms. Such hypothetical interpretations and the corresponding operator will be overlined. Let $r \in \mathbb{R}$, then $T^C(I^C(\Delta, [y \to r]), G)$ would contain the atom $p(y)$ if $r \geq 0$, thanks to the first clause. But using to the second one, it would be also happen when $r < 0$. Therefore, if the forcing for a goal $\forall y G$ is defined only in terms of the forcing for $G$, it seems impossible to avoid that $T^C(I^C(\Delta, \emptyset)) \not\models^C_\tau \forall y p(y)$. The intuitionism imposes that, in order to force $\forall y p(y)$ from $\Delta$ and any $\Gamma$, the atom $p(y)$ must be forced by all the assignments $[y \to r]\in \mathcal{R}$ and using the same clause. Therefore, it is necessary to store information regarding how atoms have been forced. That is why atoms $\mathcal{A}$ have been replaced by pairs $\langle A, cl(n, \tau) \rangle$. For this example, let $\tau_i = \text{forall}(cl(i, \text{const}))$ for $i \in \{1, 2\}$. It is easy to check that $(p(y), \tau_1) \in T^C(I^C(\Delta, [y \to r]))$ if $r \geq 0$, and $(p(y), \tau_2) \in T^C(I^C(\Delta, [y \to r]))$ if $r < 0$, but that does not lead to the fact that, for some $\tau$, $T^C(I^C(\Delta, \emptyset)) \not\models^C_\tau \forall y p(y)$.

**Example 7.** Let us consider the program $\Delta$ of the instance $HH(R)$:

- $\text{circle}(X, Y) \iff X \times X + Y \times Y \times 1.$
- $\text{parab}(X, Y) \iff X > 0, Y > 0, Y \times Y < X.$
- $\text{sector}(X, Y) \iff \text{circle}(X, Y), \text{parab}(X, Y), X > 0.5.$

Let us try to find for which $\tau$ and assignments $[x \to r]$, $r \in \mathbb{R}$, $\models^C_\tau \exists x \models^C_\tau \forall y \models^C_\tau ((0.1 < y < y < 0.2) \Rightarrow \text{sector}(x, y))$. Let $\tau = \tau_0$. That happens iff $\tau_0 = \text{forall}(\tau_1)$ and $\exists x \models^C_\tau \forall y \models^C_\tau \text{sector}(x, y)$ for each $\tau_0 \in \mathbb{R}$.

Let $\text{forall}(\tau_1)$ and $\text{sector}(x, y)$ for each $\tau_0 \in (0.1, 0.2)$.

$\models^C_\tau \exists x \models^C_\tau \forall y \models^C_\tau \text{sector}(x, y)$ for each $\tau_0 \in (0.1, 0.2)$.
\[ \Leftrightarrow \tau_i = \text{exists} ( \text{exists} ( \tau_i )) \] and for each \( s_0 \in (0.1, 0.2) \) there are \( s_1, s_2 \in \mathbb{R} \) such that

\[ \text{lfp}(T^c) \Delta, [x \leftarrow y, \gamma = s_0, x_1 \leftarrow s_1, y_1 \leftarrow s_2 \rangle \models \text{lfp}_y (x) \approx x_1 \wedge y \approx y_1 \wedge \text{circle}(x_1, y_1) \wedge \text{parab}(x_1, y_1) \wedge x_1 > 0.5. \]

This can be easily simplified into: for each \( s_0 \in (0.1, 0.2) \)

\[ \text{lfp}(T^c) \Delta, [x \leftarrow r, y = s_0, x_1 \leftarrow s_1, y_1 \leftarrow s_2 \rangle \models \text{circle}(x, y) \wedge \text{parab}(x, y) \wedge x > 0.5. \]

If this process is carried through, the only suitable \( \tau \) is obtained, together with the following condition obtained over \( r \): for each \( s_0 \in (0.1, 0.2), r > 0.5, s_0^2 < r \) and \( r^2 + s_0^2 < 1. \) So, if

\[ \Theta = \{ [x \leftarrow r] | \forall s(0.1 < s < 0.2 \Rightarrow (r > 0.5 \wedge s^2 > r \wedge r^2 + s^2 < 1)) \}, \]

\[ \text{lfp}(T^c)(I_{L^c}), \Delta, \Theta \models \forall y((0.1 < y < 0.2) \Rightarrow \text{sector}(x, y)) \]

is satisfied. In fact, \( \Theta \) is the largest set of assignments for which this holds.

Remember that we are interested in having two guided forcing relations because, once a connection between them has been established, another connection is derived between the non guided versions. The guided version of \( \models \) is now defined, as in the previous cases, for a new notion of interpretation:

**Definition 16.** A guided interpretation \( I_T \) is a function \( I_T : W \times P(C) \rightarrow P(At \times T^c) \) that is monotonous, i.e. for any \( \Delta_1, \Delta_2 \) and \( \Gamma_1, \Gamma_2 \), if \( \Delta_1 \subseteq \Delta_2 \) and \( \Gamma_1 \subseteq \Gamma_2 \), then \( I_T(\Delta_1, \Gamma_1) \subseteq I_T(\Delta_2, \Gamma_2) \). Let \( I_T \) be the set of guided interpretations.

A partial order \( \subseteq \) can be defined for \( I_T \), similarly to that for \( I \). (\( I_T, \subseteq \)) is a complete lattice and has an infimum, denoted \( I_{L^c} \), the constant function \( \emptyset \).

**Definition 17.** Given \( I_T \in I_T, \Delta, \Gamma \) and \( \tau \), a goal \( G \) is forced by \( I_T, \Delta \) and \( \Gamma \) with the guide \( \tau \), which is written \( I_T, \Delta, \Gamma \models \tau \models G \), where \( \models \tau \) is the relation recursively defined depending on the structure of \( G \), as follows:

- \( I_T, \Delta, \Gamma \models \text{forall}_{x} \exists y \neg \exists z \quad C \quad \text{if} \quad \models \text{xG} \)
- \( I_T, \Delta, \Gamma \models \text{forall}_{x} \exists y \neg \exists z \quad C \quad \text{if} \quad \models \text{xG} \)
- \( I_T, \Delta, \Gamma \models \text{forall}_{x} \exists y \neg \exists z \quad C \quad \text{if} \quad \models \text{xG} \)
- \( I_T, \Delta, \Gamma \models \text{forall}_{x} \exists y \neg \exists z \quad C \quad \text{if} \quad \models \text{xG} \)

Now the corresponding operator, whose least fixed point will help us to establish the equivalence between \( \models \tau \) and \( \models \tau \), is defined.

**Definition 18.** The operator \( T_T : I_T \rightarrow I_T \) transforms interpretations as follows. For any \( I_T \in I_T, \Delta, \Gamma \) and \( A \in At, (\langle A, c(n, \tau) \rangle) \in T_T(\Delta, \Gamma) \) if there is a variant \( \forall \bar{\tau}(G \Rightarrow \bar{\tau}^i) \) of the \( \tau^i \) clause of \( \Delta \) such that the variables \( \bar{\tau}^i \) do not occur free in \( \Delta, \Gamma, A, \) and \( I_T, \Delta, \Gamma \models \text{exists}(A \Rightarrow \bar{\tau}^i \tau^j \Gamma). \)

The operator \( T_T \) is proved to be monotonic and continuous. The proofs are analogous to those for \( T \) (Lemmas 3 and 4). Therefore, \( T_T \) has a least fixed point, which is \( \bigcup_{\tau \subseteq 0} (T_T)^{\tau}(I_{L_T}, \tau) \), and we denote it by \( \text{lfp}(T_T). \)

The following lemma and corollary justify why the interpretations \( I_T \) were said to be a guided version of those in \( I. \)

**Lemma 10.** Given \( \Delta, \Gamma, G \) and \( n > 0 \)

\[ T^n(I_{L_T}, \Delta, \Gamma) \models G \quad \Leftrightarrow \quad \text{there exists } \tau \text{ such that } (T_T)^n(I_{L_T}, \Delta, \Gamma) \models \tau. \]

**Proof.** The proof is inductive on the order relation between pairs \( \langle m, G \rangle \) defined below, where \( m \geq 0 \).

\[ (m_1, G_1) < (m_2, G_2) \text{ if and only if } m_1 < m_2 \text{ or } (m_1 = m_2 \text{ and } G_1 \text{ is a strict subformula of } G_2 \text{ up to renaming of free variables). Therefore, assuming the claim for every pair } \langle n', G' \rangle < (n, G), \text{ it must be proved for } (n, G), \text{ by case analysis on the structure of } G. \]

**Example 8.** In the Example 1,

\[ T_T^2(I_{L_T}, \Delta, \{ C \}) \models G \]

where \( \tau = \text{impc} \left( \text{forall} \left( \text{imp}(c(2, x, y) \text{and} \text{forall} \left( c(1, x, y) \right)) \right) \right). \]

**Corollary 11.** Given \( \Delta, \Gamma, G \) and \( n > 0 \)

\[ \text{lfp}(T_T)(I_{L_T}, \Delta, \Gamma) \models G \quad \Leftrightarrow \quad \text{there exists } \tau \text{ such that } \text{lfp}(T_T)(I_{L_T}, \Delta, \Gamma) \models \tau. \]

**Proof.** Thanks to Lemma 2,

\[ \text{lfp}(T_T)(I_{L_T}, \Delta, \Gamma) \models G \quad \Leftrightarrow \quad \text{there is } k > 0 \text{ such that } T^k(I_{L_T}, \Delta, \Gamma) \models G. \]

The counterpart of such lemma for the operator \( T_T \) also holds, therefore, for any \( \tau \),

\[ \text{lfp}(T_T)(I_{L_T}, \Delta, \Gamma) \models G \quad \Leftrightarrow \quad \text{there is } k > 0 \text{ such that } (T_T)^k(I_{L_T}, \Delta, \Gamma) \models G. \]

Thus, the claim is a consequence of Lemma 10.

### 3.2.2 Connecting the forcing relations

Our next task is to establish the connection between the guided semantics, and finally the non guided ones.

The following proposition states on the implications of the particular equivalence between \( \models \tau \) and \( \models \tau \).

**Proposition 12.** Given \( \Delta, \Gamma, G, \tau \),

\[ \text{lfp}(T_T)(I_{L_T}, \Delta, \Gamma) \models G \quad \Leftrightarrow \quad \text{lfp}(T^c)(\Gamma) \text{free}(\Delta \cup \{ G \}) \models \tau. \]

**Proof.** Let \( V = \text{free}(\Delta \cup \{ G \}) \) and \( \nu \in \text{free}(V) \), and let us prove \( \text{lfp}(T^c)(I_{L_T}, \Delta, \nu) \models \tau \) by induction on the structure of \( \tau \):

- \( \tau \equiv \text{const} \) and \( G \equiv C \).
- \( \text{lfp}(T_T)(I_{L_T}, \Delta, \Gamma) \models \text{const} \quad \Leftrightarrow \quad \text{lfp}(T^c)(I_{L_T}, \Delta, \Gamma) \models \text{const} \).
- \( \text{lfp}(T_T)(I_{L_T}, \Delta, \Gamma) \models \text{forall} \tau \quad \Leftrightarrow \quad \text{lfp}(T_T)(I_{L_T}, \Delta, \Gamma) \models \text{forall} \tau \).

Notice that \( \text{free}(C') \subseteq \text{dom}(\nu) \).
There are two cases:

i) $\nu \in [C']$. Then $\nu \in [\Gamma \cup \{C'\}]_{\text{free}(\Delta \cup \{G\})}$, so $\text{lfp}(T^c), \Delta, \nu \equiv^c G'$ and so $\text{lfp}(T^c), \Delta, \nu \equiv^c G$.

ii) $\nu \notin [C']$. Then, it is immediately true that $\text{lfp}(T^c), \Delta, \nu \equiv^c \text{imp}(r) \Rightarrow G'$.

- $\tau \equiv \forall r \forall x G$. Let $\forall r \forall x G_0, \text{lfp}(T), \Delta, \Gamma \equiv \forall r \forall x G_0 \iff y \notin \Delta, \Gamma$ nor $G_0, \text{lfp}(T), \Delta, \Gamma \equiv y \forall x G_0[y/x]$. Then, by induction hypothesis, $\text{lfp}(T^c), \Delta, \Gamma \equiv \forall x G_0[y/x]$ for any such $\nu'$. Thus $\text{lfp}(T^c), \Delta, \nu' \equiv^c \text{imp}(r) \forall x G_0$.

The rest of the cases can be proved in the same way.

In order to prove the remaining implication, some particular constraints $c(\Delta, \tau, G)$ are introduced. $c$ is defined as a partial function such that for any $\Delta, \tau$ and $G$, $c(\Delta, \tau, G)$ is not defined if there is not an accordance between $\tau$ and the structure of $G$. In fact, if $\text{lfp}(T^c), \Delta, \nu \equiv^c G$ for some $\nu$, then $c(\Delta, \tau, G)$ is defined, and it is the same constraint that, together with $\Delta$, forces $G$ guided by $\tau$, when there is any.

**Definition 19.** Given $\Delta, G$ and $\tau$, the partial function $c : W \times T \times G \rightarrow L_\tau$ is recursively defined by the rules below.

- $c(\Delta, \text{est}, G) = C$.
- $c(\Delta, \text{est}(n, \tau), G) = c(\Delta, \tau', \exists x (A \equiv A' \wedge G'))$, where $\forall x G' \Rightarrow A'$ is a variant of the $n$th clause of $\Delta$ and for each $x \in \tau$, $x \notin \text{free}(A), \text{free}(\Delta)$.
- $c(\Delta, \text{and}(\tau_1, \tau_2), G \wedge G_2) = c(\Delta, \tau_1, G_1) \wedge c(\Delta, \tau_2, G_2)$.
- $c(\Delta, \text{or}(i, \tau), G \vee G_2) = c(\Delta, \tau', G_1)$.
- $c(\Delta, \text{imp}(\tau), D' \Rightarrow G') = c(\Delta \cup \{D'\}, \tau', G')$.
- $c(\Delta, \text{imp}(\tau), C' \Rightarrow G') = C' \Rightarrow c(\Delta, \tau', G')$.
- $c(\Delta, \exists x G') = \exists y c(\Delta, \tau', G'[y/x])$ where $y \notin \text{free}(\Delta), \text{free}(\exists x G')$.
- $c(\Delta, \forall x G') = \forall y c(\Delta, \tau', G'[y/x])$ where $y \notin \text{free}(\Delta), \text{free}(\forall x G')$.

Notice that $c$ is in fact a partial function. For example, $c(\Delta, \text{and}(\tau_1, \tau_2), G \wedge G_2)$ is defined exactly when both $c(\Delta, \tau_1, G_1)$ and $c(\Delta, \tau_2, G_2)$ are defined. It is straightforward to check that $\text{free}(c(\Delta, \tau, G)) \subseteq \text{free}(\Delta \cup \{G\})$.

**Example 9.** Let $\Delta, \tau$ and $G$ be those in Example 7. Then $c(\Delta, \tau, G)$ is defined and $[c(\Delta, \tau, G)] = [\forall s(0.1 < s < 0.2 \Rightarrow (s * s < s \wedge x * x + s * s < 1.1))]$.

The following lemmas correspond to the technicalities we have announced in order to prove that, in the sense of Proposition 12, $\equiv^c$ implies $\equiv_r$. Their proofs are rather mechanical and therefore omitted or summarized.

The lemma below states the essential property of the constraint $c(\Delta, \tau, G)$ w.r.t. the semantics $\equiv^c$.

**Lemma 13.** Given $\Delta, \tau, G, \nu$ such that $\text{free}(\Delta \cup \{G\}) \subseteq \text{dom}(\nu)$, $\text{lfp}(T^c), \Delta, \nu \equiv^c G \iff (c(\Delta, \tau, G)$ is defined and $\nu \in [c(\Delta, \tau, G)]$).

**Proof.** The proof is inductive on the structure of $\nu$. \[\Box\]

Now we establish the connection between $c(\Delta, \tau, G)$ and the semantics $\equiv_r$.

**Lemma 14.** Given $\Delta, \tau, G$ and $\nu$, $\text{lfp}(T)$, $\Delta, \Gamma \equiv_r G \iff (c(\Delta, \tau, G)$ is defined and $\Gamma \equiv_c c(\Delta, \tau, G)$).

Finally, we are ready to prove the counterpart of Proposition 12.

**Proposition 15.** Given $\Delta, \Gamma \equiv_r G$ and $\tau$, $\text{lfp}(T^c), \Delta, \Gamma \equiv^c G \iff \text{lfp}(T), \Delta, \Gamma \equiv_r G$.

**Proof.** Let $V = \text{free}(\Delta \cup \{G\})$ and let us assume that $\text{lfp}(T^c), \Delta, \Gamma \equiv^c G$. $\Gamma$ is $\equiv_r$ implies that $\Gamma \equiv_r G$ is not empty, so thanks to Lemma 13 we have that $\Gamma \equiv_r G$ is defined and $\Gamma \equiv_r G \subseteq [c(\Delta, \tau, G)]$. However, we want to obtain that $\Gamma \equiv^c G \subseteq [c(\Delta, \tau, G)]$. So in order to prove it, let $\nu \in \Gamma \equiv^c G$, and let $\nu'$ be any extension of $\nu$ such that $\forall \nu \subseteq \text{dom}(\nu')$. Then, $\nu' \in \Gamma \equiv_r G$ thanks to $\nu$, and since $\nu' \equiv_r G \subseteq [c(\Delta, \tau, G)]$ also holds. Hence, $\Gamma \equiv_r G$ has been proved, which implies that $\Gamma \equiv_r c(\Delta, \tau, G)$. Finally, from Lemma 14, $\text{lfp}(T), \Delta, \Gamma \equiv_r G$ is obtained. \[\Box\]

The main theorem below, which establishes the relation between the non guided semantics, is a consequence of the previous results.

**Theorem 16.** For any $\Delta, \Gamma \equiv_r G$ and $\tau$,

$\text{lfp}(T), \Delta, \Gamma \equiv_r G \iff \text{lfp}(T^c), \Delta, \Gamma \equiv^c G \iff \Delta; \Gamma \vdash_{uc} G$.

**Proof.** $\text{lfp}(T), \Delta, \Gamma \equiv_r G$ there exists some $\tau$ such that $\text{lfp}(T), \Delta, \Gamma \equiv_r G$ by Corollary 11 $\exists \tau$ such that $\text{lfp}(T^c), \Delta, \Gamma \equiv^c G$ by Propositions 12 and 15 $\iff \text{lfp}(T), \Delta, \Gamma \equiv_{uc} G$ by definition of $\equiv^c \iff \Delta; \Gamma \vdash_{uc} G$ by Theorem 6. \[\Box\]

**4. CONCLUSIONS**

In previous papers [11, 10] combinations of HH and CLP were proposed, producing first and higher order schemes $HH(C)$ parametric w.r.t. the constraint system. These amalgamated languages gather the expressivity and the efficiency advantages of $HH$ and CLP, respectively. A proof system that merges inference rules from intuitionistic sequent calculus with the entailment relation of a constraint system was defined. This proof system guarantees uniform proofs, which are the basis of abstract logic programming languages [14]. A goal solving procedure that is sound and complete w.r.t. the proof system was also presented. Such procedure could be seen as an operational semantics of $HH(C)$, however the absence of a more declarative semantics for this new language was evident. In this paper we have defined semantics for $HH(C)$ based on fixed point constructions as is usually done in the LP and CLP fields [12, 2, 3, 9, 6].

As far as we know, our work is the first attempt to give declarative semantics to an amalgamated logic that combines the Hereditary Harrop fragment of intuitionistic first-order logic with a constraint system. Due to the embedding of implications and universal quantifiers inside goals (and so
inside programs), finding a fixed point semantics becomes a hard task, further obstructed by the presence of constraints.

In [13] a model theory is presented for an extension of Horn clauses including implications in goals based on a fixed point construction, and it is proved that the operational meaning of implication is sound and complete w.r.t. this semantics. Our approach is close to this framework, but it incorporates the semantics of universal quantifiers and constraints in goals. The universal quantifier is also handled in [4], but the presence of universal constraints involves further difficulties that we have solved.

A semantics for the fragment of λ-prolog — that is based on the higher-order logic \(HH\) without constraints —, in which classical and intuitionistic theories coincide, is presented in [20]. But this is not the case if implications and universal quantifiers are considered.

Referring to CLP, most of the defined semantics use different fixed point constructions. For instance in [9] fixed point semantics constitutes a bridge between operational and algebraic semantics. This is also our aim. But notice that in traditional CLP the programs are limited to be Horn clauses with constraints. So in the frame of constraint systems which are complete w.r.t. a theory, programs (with embedded constraints) may be interpreted using classical logical inference. However, this is not the case in our language. A classical theory can be considered for the constraint system, but anyway the intuitionism remains, even in the interpretation of pure programs.

The deduction system, which is a syntactic tool, should be supported by model-theoretic semantics involving more abstract elements.

We are still interested in finding a pure model-theoretic semantics, not so directly connected with the operational one, in which models should provide for meanings of constraints, programs and goals in a homogeneous way, and \(C\) is a correct answer constraint for \(\Delta\) and \(G\), if and only if, every model satisfying \(\Delta\) and \(C\) satisfies \(G\). However, bearing in mind that \(HC\) is not a traditional sequent calculus (due to the presence of constraints), its correspondence with a classical or intuitionistic inference relation (\(\models\)) cannot be direct, so the definition of an specific model-theoretic semantics merging the intuitionistic behavior of \(HH\) and the interpretation of constraints becomes a hard task. We are researching for more abstract model theories based on indexed categories or uniform algebras [5, 1], that could provide for such pure model-theoretic semantics.

Acknowledgements We strongly appreciate the comments and suggestions from James Lipton concerning the current work.

5. REFERENCES