Formalizing a Constraint Deductive Database Language Based on Hereditary Harrop Formulas with Negation

Susana Nieva[1], Jaime Sánchez-Hernández[1], and Fernando Sáenz-Pérez[2,⋆]

[1] Dept. Sistemas Informáticos y Computación, UCM, Spain
[2] Dept. Ingeniería del Software e Inteligencia Artificial, UCM, Spain
{nieva,jaime,fernan}@sip.ucm.es

Abstract. In this paper, we present an extension of the scheme $HH(\mathcal{C})$ (Hereditary Harrop formulas with Constraints) with a suitable formulation of negation in order to obtain a constraint deductive database query language. In addition to constraints, our proposal includes logical connectives (implication and quantifiers) for defining databases and queries, which altogether are unavailable in current database query languages.

We define a proof theoretic semantic framework based on a sequent calculus, that allows to represent the meaning of a database query by means of a derived constraint answer in the sense of *CLP*. We also introduce an appropriate notion of stratification, which provides a starting point for suitable operational semantics dealing with recursion and negation. We formalize a fixed point semantics for stratifiable databases, whose fixpoint operator is applied stratum by stratum. This semantics is proved to be sound and complete with respect to derivability in the sequent calculus, and it provides the required support for actual implementations, as the prototype we have developed already and introduce in this paper.

1 Introduction

The scheme $HH(\mathcal{C})$ (Hereditary Harrop formulas with Constraints) [10] extends *HH* by adding constraints, in a similar way the extension of *LP* (Logic Programming) with constraints gave rise to the *CLP* (Constraint Logic Programming) scheme [9]. In this scheme, a parametric domain of constraints is assumed, so that it is possible to consider different instances (such as arithmetical constraints over real numbers and finite domain constraints). The extension is completely integrated into the language: Constraints are allowed to occur in goals, bodies of clauses and answers.

For example, considering the instance $HH(\mathcal{R})$, i.e., the domain of arithmetic constraints over real numbers, a circle can be defined by its center and radius, using non-linear constraints (in Prolog-like notation):

```
circle(XC,YC,R,X,Y) :- ((X-XC)**2 + (Y-YC)**2) ≤ R**2.
```

⋆ The authors are partially supported by the Spanish projects 'MERIT-FORMS': TIN2005-09207-C03-03, 'PROMESAS-CAM': S-0505/TIC/0407.

J. Garrigue and M. Hermenegildo (Eds.): FLOPS 2008, LNCS 4989, pp. 289–304, 2008.

We can ask, for instance, if any pair (x, y) such that $x^2 + y^2 = 1$ (the circumference centered in the origin and radius 1) is inside the circle with center $(0, 0)$ and radius 2 by means of the goal:

$\forall$ `x` $\forall$ `y ((x**2 + y**2` $\approx$ `1)` $\Rightarrow$ `circle(0,0,2,x,y)).`

In this paper, we investigate the use of $HH(\mathcal{C})$ not as a (general purpose) programming language, but as the basis for database systems with constraints. We argue that, in the same way that Datalog [20] and Datalog with constraints [16] arise for modeling database systems inspired in Prolog and *CLP* respectively, the language $HH(\mathcal{C})$ can offer a suitable starting point for the same purpose.

$HH(\mathcal{C})$ improves the expressivity of traditional deductive database languages because the underlying logic embraces both new connectives and constraints. In particular, implications can be used to write hypothetical queries, universal quantification allows encapsulation, and constraints allow managing infinite data. To the best of our knowledge, former works (e.g., [15,18,17,5]) do not consider all these features altogether.

Let us see an example. Assume an instance in which finite and real constraint domains are combined. We can define the database:

```
flight(mad,par,1.5).    flight(par,ny,10).    flight(lon,ny,9).
travel(X,Y,T) :- flight(X,Y,D), T >= D.
travel(X,Y,T) :- flight(X,Z,T1), travel(Z,Y,T2), T >= T1+T2.
```

The relations `flight` and `travel` represent tuples `<Origin, Destination, FlightTime>` for both direct and linked connections between cities (extensional and intensional database, resp.). The implication

`flight(mad, lon, T)` $\Rightarrow$ `travel(mad, ny,11)`

(a valid goal in our language) represents the query: Assuming that there is a direct connection between Madrid and London, what duration should it have in order to be able to travel from Madrid to New York in 11 hours at most? The answer to this query will be the constraint $11 \geq T + 9$, which is equivalent to $T \leq 2$ in the constraint system.

Another hypothetical query to the previous database can be whether it is possible to travel from Madrid to some place in any time greater than 1.5. The goal formulation $\forall$ `t (t` $>$ `1.5` $\Rightarrow$ $\exists$ `y travel(mad,y,t))` includes also universal quantification, and the corresponding answer is *true*.

However, $HH(\mathcal{C})$ lacks of negation, which is needed to capture set difference in order to be complete with respect to Relational Algebra (RA). As it is well-known, incorporating negation into logic programming languages is a difficult task (see [2] for a survey). Negation in the specific field of deductive database systems has been also widely studied [1,3]. In our language, negation is even more complex due to the presence of implication and universal quantification in goals. Based on an extension of the sequent calculus defined for $HH(\mathcal{C})$ in [10], we provide a proof theoretic meaning of goals (queries) from programs (databases), in such a way that the existence of constraints is exploited to represent answers and

to finitely model infinite databases. This is also the case of constraint databases, but as our core logic is very expressive, the resulting language is richer.

Using the database of the previous example, the query $\neg\exists$ `t flight(X,Y,t)` (or its equivalent $\forall$ `t` $\neg$ `flight(X,Y,t)`), which represents the cities in the database that have no direct flights between them, is not available in extended database languages as domain relational calculus or Datalog with constraints. However, in our system —where formulas are interpreted in the context of the constraint domain of the particular instance— $\forall$ `t` $\neg$ `flight(X,Y,t)` represents a valid goal, and one of its possible answer constraints is: $(\neg(X \approx mad) \vee \neg(Y \approx par)) \wedge (\neg(X \approx par) \vee \neg(Y \approx ny)) \wedge (\neg(X \approx lon) \vee \neg(Y \approx ny))$, which is equivalent to $X \approx mad \wedge Y \approx ny$ in the domain of the cities registered in the current database.

After formalizing $HH(\mathcal{C})$ with negation in Section 2, by means of the proof theoretical meaning, in Section 3 we focus on the problem that arises when dealing with recursion and negation: Termination. We adapt the usual notions of stratified negation to our context in order to establish syntactic conditions that characterize a limited form of negation, for which an operational semantics could be defined. The main results of this paper appear in Section 4, where a fixed point semantics, based on the previous notion of stratification, is defined and proved to be sound and complete with respect to the proof theoretical one (full proofs can be found in `http://gpd.sip.ucm.es/papers/Archivos/nss-tr2008.pdf`). As it is shown in Section 5, this semantics provides support for an implementation.

2 $HH(\mathcal{C})$ with Negation

The original formalisms in which $HH(\mathcal{C})$ is founded [10,7] are not enough expressive to represent set difference, so it is incomplete with respect to RA. We will extend the scheme including negation to obtain a complete *Constraint Deductive Database* (*CDDB*) language w.r.t. RA. Next, we make precise the syntax of the formulas of $HH(\mathcal{C})$ extended with negation, denoted as $HH_\neg(\mathcal{C})$, showing how the usual notions of programs and goals of Logic Programming can be translated into databases and queries, respectively. The evaluation of a query with respect to a deductive database can be seen as the computation of a goal from a set of facts (ground atoms) defining the extensional database, and a set of clauses, defining the intensional database. As it is common in deductive databases, the definition of a predicate, by means of clauses, can be seen in our language as the definition of a view in relational databases.

2.1 Syntax

As usual, formulas will be built up from terms, using predicates and connectives. We consider *defined predicate symbols*, representing the names of database relations, to build atoms, and *non-defined (built-in) predicate symbols*, including at least the equality predicate symbol $\approx$, to build constraints. We will also assume a set of constant and operator symbols in the constraint system, and a set of variables to build terms.

Well formed formulas in $HH_{\neg}(\mathcal{C})$ can be classified into clauses D (defining database relations) and goals (or queries) G. They are recursively defined by the following rules:

$$D ::= A \mid G \Rightarrow A \mid D_1 \wedge D_2 \mid \forall x D$$
$$G ::= A \mid \neg A \mid C \mid G_1 \wedge G_2 \mid G_1 \vee G_2 \mid D \Rightarrow G \mid C \Rightarrow G \mid \exists x G \mid \forall x G$$

A represents an atom, i.e., a formula of the form $p(t_1, \ldots, t_n)$, where p is a defined predicate symbol of arity n, and t_i are terms; C represents a constraint. The incorporation of negated atoms in goals is the surplus to $HH(\mathcal{C})$.

The constraints we consider belong to a generic system $\mathcal{C} = \langle \mathcal{L}_\mathcal{C}, \vdash_\mathcal{C} \rangle$ where $\mathcal{L}_\mathcal{C}$ is the constraint language and $\vdash_\mathcal{C}$ is a binary *entailment relation*. $\Gamma \vdash_\mathcal{C} C$ denotes that the constraint C is inferred in the constraint system $\mathcal{C}$ from the set of constraints Γ. Some minimal conditions are imposed to $\mathcal{C}$ to be a constraint system: $\mathcal{L}_\mathcal{C}$ contains at least every first-order formula built up using $\top$ (*true*), $\bot$ (*false*), built-in predicate symbols, the connectives $\wedge, \neg$, and the existential quantifier $\exists$. Regarding to $\vdash_\mathcal{C}$, it includes the inference rules related to the considered connectives and quantifiers, valid in intuitionistic logic with equality; in addition, it is compact and generic (see [10] for details). The novelty is that $\mathcal{C}$ is required to deal with negation, because the incorporation of $\neg$ to HH is propagated to the constraint system, which has the responsibility of checking the satisfiability of answers in the constraint domain.

We say that a constraint C is $\mathcal{C}$-satisfiable if $\emptyset \vdash_\mathcal{C} \overline{\exists} C$, where $\overline{\exists} C$ stands for the existential closure of C. C and C' are $\mathcal{C}$-equivalent if $C \vdash_\mathcal{C} C'$ and $C' \vdash_\mathcal{C} C$.

For instance, the constraint systems of the examples are assumed to verify the required minimal conditions aforementioned. Moreover, they also include the connective $\vee$, constants to represent numbers and cities, arithmetical operators, and built-in predicates ($\geq, \ldots$).

Programs, denoted by Δ, are sets of clauses and represent databases. Any Δ can always be given as an equivalent set, $elab(\Delta)$, of implicative clauses with atomic heads in the way we precise now. The *elaboration* of a program Δ is the set $elab(\Delta) = \bigcup_{D \in \Delta} elab(D)$, where $elab(D)$ is defined by:

$$elab(A) = \{\top \Rightarrow A\} \qquad elab(D_1 \wedge D_2) = elab(D_1) \cup elab(D_2)$$
$$elab(G \Rightarrow A) = \{G \Rightarrow A\} \qquad elab(\forall x D) = \{\forall x D' \mid D' \in elab(D)\}$$

We will assume that a view defining a predicate is a set of elaborated clauses of the form $\forall x_1 \ldots \forall x_n (G \Rightarrow A)$[1]. In the examples (as before), we will use the common notation A :- G, assuming that capital letters inside A and G represent variables that are implicitly universally quantified, and incorporating the new connectives in goals. A is called the head and G the body of the clause as usual. Negation is not allowed in the head of a clause, but inside its body.

Example 1. Assume a more realistic situation of the flights example in the Introduction, where flight delays may happen:

[1] $\forall x_1 \ldots \forall x_n$ will be abbreviated by $\forall \overline{x}$.

```
deltravel(X,Y,T) :- flight(X,Y,T1), delay(X,Y,T2), T ≥ T1+T2.
deltravel(X,Y,T) :- flight(X,Z,T1), delay(X,Z,T2),
                    deltravel(Z,Y,T3), T ≥ T1+T2+T3.
```

Tuples of `delay` may be in the extensional database or may be assumed when the query is formulated. For instance, the goal:

```
(∀ x delay(par,x,1),delay(mad,par,0.5)) ⇒ deltravel(mad,ny,T)
```

represents the query: What is the time needed to travel from Madrid to New York assuming that for any destination there is a delay of one hour from Paris, and the flight from Madrid to Paris is half an hour delayed? According to its proof theoretic interpretation, in order to solve the goal `deltravel(mad,ny,T)`, the clauses `delay(par,X,1)` and `delay(mad,par,0.5)` will be added locally to the database, and they will not be considered any more once the goal is solved. Similar queries can be generalized as views (defined by clauses). For instance:

```
needtime(C1,C2,D,T) :- ∀ x delay(C1,x,D) ⇒ deltravel(C1,C2,T).
```

Notice that this clause is neither allowed by Prolog with negation nor Datalog.

Since flights may or may not be delayed, a more general view can be defined in order to know the expected time of a trip:

```
trip(X,Y,T) :- nondeltravel(X,Y,T) ; deltravel(X,Y,T).
nondeltravel(X,Y,T) :- ¬ delayed(X,Y), travel(X,Y,T).
delayed(X,Y) :- ∃ t (delay(X,Y,t), ¬ t ≈ 0).
```

2.2 Sequent Calculus

Several kinds of semantics have been defined for $HH(\mathcal{C})$ without negation, including proof theoretic, operational [10] and fixed point semantics [7], as well as for its higher-order version [11]. The simplest way for explaining the meaning of programs and goals in the present framework is by using a proof theoretic semantics. Queries formulated to a database are interpreted by means of the inference system that governs the underlying logic. This proof system, called $\mathcal{UC}$ (Uniform sequent calculus handling Constraints) [10] is a sequent calculus that combines traditional inference rules with the entailment relation $\vdash_{\mathcal{C}}$ of the generic constraint system $\mathcal{C}$. The rules defining derivability in $\mathcal{UC}$ appear in Figure 1. Sequents have the form $\Delta; \Gamma \vdash C$, where programs and sets of constraints are on the left, and goals on the right.

Next, we explain the rules $(\exists_R)$ and $(Clause)$, the others correspond to widespread intuitionistic rules introducing connectives on the right of the sequent (see, e.g., [13]), except (C_R) which deals with goals that are pure constraints. $(\exists_R)$ captures the fact that the witness in the proof of an existentially quantified formula can be represented by a constraint that can be more general than an equality $x \approx t$ simulating a substitution (e.g., $(x * x \approx 2)$ represents the witness $\sqrt{2}$, which cannot be written as a term). $(Clause)$ represents backchaining and allows to prove an atomic goal $A \equiv p(t_1, \ldots, t_n)$, using a program clause whose head $A' \equiv p(t'_1, \ldots, t'_n)$ is not required to unify with A, but rather solving a new existentially quantified goal that, by applying the $(\exists_R)$ rule, will result in

$$\frac{\Gamma \vdash_{\mathcal{C}} C}{\Delta; \Gamma \vdash C}\ (C_R) \qquad \frac{\Delta; \Gamma \vdash \exists x_1 \ldots \exists x_n((A' \approx A) \wedge G)}{\Delta; \Gamma \vdash A}\ (Clause)\ (*), \text{where}$$

$\forall x_1 \ldots \forall x_n(G \Rightarrow A')$ is a variant of a formula of $elab(\Delta)$

$$\frac{\Delta; \Gamma \vdash G_i}{\Delta; \Gamma \vdash G_1 \vee G_2}\ (\vee_R)\ (i = 1, 2) \qquad \frac{\Delta; \Gamma \vdash G_1 \quad \Delta; \Gamma \vdash G_2}{\Delta; \Gamma \vdash G_1 \wedge G_2}\ (\wedge_R)$$

$$\frac{\Delta, D; \Gamma \vdash G}{\Delta; \Gamma \vdash D \Rightarrow G}\ (\Rightarrow_R) \qquad \frac{\Delta; \Gamma, C \vdash G}{\Delta; \Gamma \vdash C \Rightarrow G}\ (\Rightarrow C_R)$$

$$\frac{\Delta; \Gamma, C \vdash G[y/x] \quad \Gamma \vdash_{\mathcal{C}} \exists y C}{\Delta; \Gamma \vdash \exists x G}\ (\exists_R)(**) \qquad \frac{\Delta; \Gamma \vdash G[y/x]}{\Delta; \Gamma \vdash \forall x G}\ (\forall_R)(**)$$

$(*)$ $x_1, \ldots, x_n$ fresh for A

$(**)$ y fresh for the formulas in the conclusion

Fig. 1. Rules of the Sequent Calculus $\mathcal{UC}$

a search for a constraint that implies the equality $A' \approx A$ (that stands for $t'_1 \approx t_1 \wedge \ldots \wedge t'_n \approx t_n$).

$\mathcal{UC}$ provides only uniform proofs in the sense defined by Miller et al. [13], i.e., goal-oriented proofs. The rules are applied backwards and, at any step, the applied rule is that corresponding to the connective of the goal to be proved.

The Meaning of Negated Atoms. Derivability in $\mathcal{UC}$ provides proof theoretic semantics for $HH(\mathcal{C})$. The incorporation of negation makes necessary to extend the notion of derivability, because there is no rule for this connective in $\mathcal{UC}$. Therefore, we extend $\mathcal{UC}$ with a new rule to incorporate derivability of negated atoms. The idea of interpreting the query $\neg A$ from a database Δ, by means of an answer constraint C, is that whenever C' is a possible answer to the query A from Δ, then $C \vdash_{\mathcal{C}} \neg C'$. This is formalized with the "metarule":

$$\frac{\Gamma \vdash_{\mathcal{C}} \neg C \quad \text{for every } \Delta; C \vdash A}{\Delta; \Gamma \vdash \neg A}\ (\neg_R)$$

We say that $(\neg_R)$ is a metarule since its premise considers any derivation $\Delta; C \vdash A$ of the atom A. In practice, there is a derivation of $\neg A$ when the set of answer constraints of A from Δ is finite.

We define the inference system $\mathcal{UC}_\neg$ as $\mathcal{UC}$ plus the rule $(\neg_R)$. The notation $\Delta; \Gamma \vdash_{\mathcal{UC}_\neg} G$ means that the sequent $\Delta; \Gamma \vdash G$ has a proof using the rules of $\mathcal{UC}$ and $(\neg_R)$.

Definition 1. *If $\Delta; C \vdash_{\mathcal{UC}_\neg} G$ then C is called an* answer constraint *to the query G in the database Δ.*

Example 2. Consider the program below defining the inside of a rectangle with left-bottom corner (X_1, Y_1) and right-top corner (X_2, Y_2).

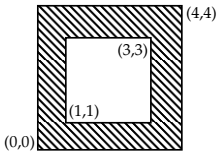
$\Delta = \{$ `rectangle(X1,Y1,X2,Y2,X,Y) :- X≥X1, X≤X2, Y≥Y1, Y≤Y2` $\}$

It is possible to formulate a query to get the difference between two rectangles (the dashed frame in the next figure) by the goal:

```
rectangle(0,0,4,4,X,Y), ¬ rectangle(1,1,3,3,X,Y)
```

obtaining as an answer constraint:

$$\begin{array}{r}C \equiv ((y > 3) \wedge (y \leq 4) \wedge (x \geq 0) \wedge (x \leq 4)) \vee \\ ((y \geq 0) \wedge (y < 1) \wedge (x \geq 0) \wedge (x \leq 4)) \vee \\ ((y \geq 0) \wedge (y \leq 4) \wedge (x > 3) \wedge (x \leq 4)) \vee \\ ((y \geq 0) \wedge (y \leq 4) \wedge (x \geq 0) \wedge (x < 1))\end{array}$$



by the following deduction:

$$\dfrac{\dfrac{\dfrac{C \vdash_{\mathcal{R}} \exists a_1, a_2, b_1, b_2, x_1, y_1 (a_1 \approx 0 \ldots)}{\Delta; C \vdash \exists a_1, a_2, b_1, b_2, x_1, y_1 (a_1 \approx 0 \wedge x_1 \approx x \wedge x_1 \geq a_1 \wedge \\ a_2 \approx 0 \wedge y_1 \approx y \wedge x_1 \leq b_1 \wedge b_1 \approx 4 \wedge y_1 \geq a_2 \wedge b_2 \approx 4 \wedge y_1 \leq b_2)} (C_R)}{\Delta; C \vdash rectangle(0,0,4,4,x,y)} (Clause) \quad \mathbf{D}}{\Delta; C \vdash rectangle(0,0,4,4,x,y) \wedge \neg rectangle(1,1,3,3,x,y)} (\wedge_R)$$

where **D** is the deduction:

$$\dfrac{C \vdash_{\mathcal{R}} \neg(x \geq 1 \wedge y \geq 1 \wedge x \leq 3 \wedge y \leq 3) \quad \dfrac{\ldots}{\Delta;\ x \geq 1 \wedge y \geq 1 \wedge x \leq 3 \wedge y \leq 3 \vdash rectangle(1,1,3,3,x,y)}}{\Delta; C \vdash \neg rectangle(1,1,3,3,x,y)} (\neg_R)$$

In order to define an operational semantics for $HH_{\neg}(\mathcal{C})$, some finiteness conditions must be imposed to make viable the metarule $(\neg_R)$. That is, we have to guarantee to get only a finite number of non-equivalent computed answer constraints for any atom that occurs negated in some goal.

In this way, it is possible to impose the following restriction: A predicate q can not occur negated in the definition of a predicate p if "q depends on p". This restriction establishes a limitation on mutually recursive definitions. But, even in the case of adopting this strong syntactic restriction, completeness w.r.t. RA remains, since RA does not include recursion.

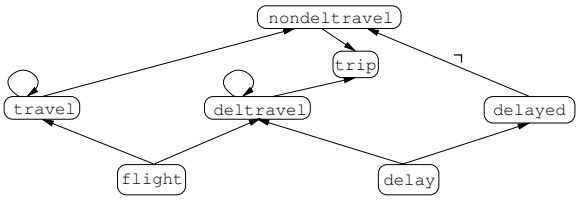
In the next section, we formalize the concept of positive and negative dependencies, and the stratifiable database notion is introduced.

3 Dependency Graphs and Stratified Negation

A well-known problem arises in deductive database languages when negation and recursion are considered altogether. Several approaches have been used to deal with this problem. This is the case of *answer set programming* [6] or the use of *stratified negation* [20]. We have found this second approach more suitable to our scheme, because $HH_{\neg}(\mathcal{C})$ provides constraints as answers, and handles implications, which involves dynamic program increase. Stratification is based on the definition of a *dependency graph* for a program. Given a set of clauses and goals Φ, the corresponding dependency graph DG_{Φ} is a directed graph whose nodes are the defined predicate symbols in Φ, and the edges are determined by the implication symbols of the formulas.

Here, we adapt those notions as a useful starting point of a fixed point semantics for our language. But now, the construction of dependency graphs must consider the fact that implications may occur not only between the head and the body of a clause, but also inside the goals, and therefore in any clause body. This feature will be taken into account in the following way: An implication of the form $F_1 \Rightarrow F_2$ produces edges (or paths) in the graph from the defined predicate symbols inside F_1 to every defined predicate symbol inside F_2. An edge will be negatively labeled when the corresponding atom occurs negated on the left of the implication. Since constraints do not include defined predicate symbols, they cannot produce dependencies. In [14], we defined an algorithm to compute the dependency graph of any set Φ.

Example 3. Consider a database Δ consisting of the predicates defined in previous examples. The dependency graph for Δ is:



The query $G \equiv \exists$ `t (deltravel(X,Y,t)` $\Rightarrow$ `delayed(X,Y))` would introduce the new edge `deltravel` $\rightarrow$ `delayed` into the previous graph.

The dependency graph is used to define stratification in $HH_{\neg}(\mathcal{C})$.

Definition 2. *Given a set of formulas Φ, its corresponding dependency graph DG_Φ, and two predicates p and q, we say:*

- q depends *on p if there is a path from p to q in DG_Φ.*
- q negatively depends *on p if there is a path from p to q in DG_Φ with at least one negatively labeled edge.*

Definition 3. *Let Φ be a set of formulas and $P = \{p_1, \ldots, p_n\}$ the set of defined predicate symbols of Φ. A* stratification *of Φ is any mapping $s : P \rightarrow \{1, \ldots, n\}$ such that $s(p) \leq s(q)$ if q depends on p, and $s(p) < s(q)$ if q negatively depends on p. Φ is stratifiable if there is a stratification for it.*

Example 4. A stratification for the database Δ of Example 3 will collect all the predicates in the stratum 1 except `nondeltravel` and `trip`, which will be in stratum 2. Intuitively, this means that for evaluating `nondeltravel`, the rest of predicates (except `trip`) should be evaluated before (in particular, `delayed`). Adding the query G in the Example 3, $\Delta \cup \{G\}$ remains stratifiable, but adding `trip(mad,lon,T)` $\Rightarrow$ `delay(mad,ny, T)`, results in a non-stratifiable set: This adds the dependency `trip` $\rightarrow$ `delay`, and then, any stratification s must satisfy $s($`trip`$) \leq s($`delay`$) < s($`nondeltravel`$) \leq s($`trip`$)$, that is impossible.

A remarkable point is that we assume, on the follow, the existence of a fixed stratification s for the considered sets $\Delta \cup \{G\}$.

It is useful to have a notion of the stratum of an atom (i.e., the stratum of its predicate symbol), but also to extend this notion to any formula or set of formulas.

Definition 4. *Let F be a goal or a clause. The* stratum of a formula *F, denoted $str(F)$, is recursively defined as:*

$str(p(t_1, \ldots, t_n)) = s(p) \qquad str(\neg A) = 1 + str(A) \qquad str(C) = 1$
$str(F_1 \Box F_2) = max(str(F_1), str(F_2))$, *where* $\Box \in \{\wedge, \vee, \Rightarrow\}$
$str(QxF) = str(F)$, *where* $Q \in \{\exists, \forall\}$

The stratum of a set of formulas Φ *is* $str(\Phi) = max\{str(F) \mid F \in \Phi\}$.

4 Fixed Point Semantics

We have extended the semantics presented in [7] in order to interpret full $HH_{\neg}(\mathcal{C})$. The semantics there defined is based on a *forcing relation* among programs, sets of constraints and goals that states whether an interpretation makes true a goal G in the context $\langle \Delta, \Gamma \rangle$ of a program and a set of constraints. Interpretations were defined as functions able to give meaning to every pair $\langle \Delta, \Gamma \rangle$ as sets of atoms. The interpretation should depend on this context because, when computing implicative goals, Δ or Γ may be augmented.

In order to deal with negation, interpretations and the fixpoint operator will operate over strata. So, contexts will be stratifiable databases (that may augment). An interpretation over a stratum i of a database will be a set of pairs $(A, C) \in At \times \mathcal{SL}_{\mathcal{C}}$ (atom, $\mathcal{C}$-satisfiable constraint), where $str(A) \leq i$.

4.1 Stratified Interpretations and Forcing Relation

Let $\mathcal{W}$ be the set of stratifiable databases Δ (with respect to the same fixed stratification s), At be the set of open atoms, and $\mathcal{SL}_{\mathcal{C}}$ be the set of $\mathcal{C}$-satisfiable constraints modulo $\mathcal{C}$-equivalence.

We will consider functions $I : \mathcal{W} \to \mathcal{P}(At \times \mathcal{SL}_{\mathcal{C}})$. In order to simplify the notation, we write $(A, C) \in At \times \mathcal{SL}_{\mathcal{C}}$, assuming that C denotes any constraint $\mathcal{C}$-equivalent to it. The notation $[I(\Delta)]_i$ represents the following subset of $I(\Delta)$,

$$[I(\Delta)]_i = \{(A, C) \in I(\Delta) \mid str(A) = i\}.$$

Notice that if $str(\Delta) = k$, then $\{[I(\Delta)]_i \mid 1 \leq i \leq k\}$ is a partition of $I(\Delta)$.

Interpretations can be classified on strata. An interpretation gives information up to its corresponding stratum.

Definition 5. *Let $i \geq 1$. An* interpretation I over the stratum i *is a function $I : \mathcal{W} \to \mathcal{P}(At \times \mathcal{SL}_{\mathcal{C}})$, such that for any $\Delta \in \mathcal{W}$, and any $j > i$, $[I(\Delta)]_j = \emptyset$. We denote by $\mathcal{I}_i$ the* set of interpretations over i.

For every $i \geq 1$, an order on $\mathcal{I}_i$ can be defined.

Definition 6. *Let $i \geq 1$ and $I_1, I_2 \in \mathcal{I}_i$. I_1* is less or equal than *I_2* at stratum *i, denoted by $I_1 \sqsubseteq_i I_2$, if for each $\Delta \in \mathcal{W}$ the following conditions are satisfied:*

- $[I_1(\Delta)]_j = [I_2(\Delta)]_j$, *for every* $1 \leq j < i$.
- $[I_1(\Delta)]_i \subseteq [I_2(\Delta)]_i$.

It is straightforward to check that for any $i \geq 1$, $(\mathcal{I}_i, \sqsubseteq_i)$ is a poset.

The idea is that when an interpretation over a stratum i increases, the information of the smaller strata remains invariable. In such a way, if $str(\neg A) = i$, since $str(A) = i - 1$, the truth value of $\neg A$ at the stratum i will remain invariable and monotonicity of the truth relation can be guaranteed even for negative atoms, as we will show.

In addition, the following result holds.

Lemma 1. *For any $i \geq 1$, any chain of interpretations of $(\mathcal{I}_i, \sqsubseteq_i)$, $\{I_n\}_{n\geq 0}$, such that $I_0 \sqsubseteq_i I_1 \sqsubseteq_i I_2 \sqsubseteq_i \ldots$, has a least upper bound $\bigsqcup_{n\geq 0} I_n$, which can be defined as: $(\bigsqcup_{n\geq 0} I_n)(\Delta) = \bigcup_{n\geq 0}\{I_n(\Delta)\}$, for any $\Delta \in \mathcal{W}$.*

Proof. Straightforward using the definition of $(\mathcal{I}_i, \sqsubseteq_i)$. □

The following definition formalizes the notion of a query G being "true" for an interpretation I in the context of a database Δ, if the constraint C is satisfied. As already said, we assume that s is not only a stratification for Δ, but also for $\Delta \cup \{G\}$.

Definition 7. *Let $i \geq 1$. The* forcing relation $\Vdash$ *between pairs I, Δ and pairs (G, C) (where $I \in \mathcal{I}_i$, $str(G) \leq i$, and C is $\mathcal{C}$-satisfiable) is recursively defined by the rules below. When $I, \Delta \Vdash (G, C)$, it is said that (G, C) is* forced *by I, Δ.*

- $I, \Delta \Vdash (C', C) \iff C \vdash_{\mathcal{C}} C'$.
- $I, \Delta \Vdash (A, C) \iff (A, C) \in I(\Delta)$.
- $I, \Delta \Vdash (\neg A, C) \iff$ *for every $(A, C') \in I(\Delta)$, $C \vdash_{\mathcal{C}} \neg C'$ holds. If there is no pair of the form (A, C') in $I(\Delta)$, then $C \equiv \top$.*
- $I, \Delta \Vdash (G_1 \wedge G_2, C) \iff$ *for each $i \in \{1, 2\}$, $I, \Delta \Vdash (G_i, C)$.*
- $I, \Delta \Vdash (G_1 \vee G_2, C) \iff$ *for some $i \in \{1, 2\}$ $I, \Delta \Vdash (G_i, C)$.*
- $I, \Delta \Vdash (D \Rightarrow G, C) \iff I, \Delta \cup \{D\} \Vdash (G, C)$.
- $I, \Delta \Vdash (C' \Rightarrow G, C) \iff I, \Delta \Vdash (G, C \wedge C')$.
- $I, \Delta \Vdash (\exists x G, C) \iff$ *there is C' such that $I, \Delta \Vdash (G[y/x], C')$, where y does not occur free in Δ, $\exists x G$, C, and $C \vdash_{\mathcal{C}} \exists y C'$.*
- $I, \Delta \Vdash (\forall x G, C) \iff I, \Delta \Vdash (G[y/x], C)$ *where y does not occur free in Δ, $\forall x G$, C.*

Those rules are well-defined because if s is a stratification for $\Delta \cup \{G\}$, with $str(G) \leq i$, and G' is a subformula of G, then s is also a stratification for $\Delta \cup \{G'\}$, and $str(G') \leq i$. Notice that, for the particular case $G \equiv D \Rightarrow G'$, s will be also a stratification for $\Delta \cup \{D, G'\}$.

From now on, when we write $I, \Delta \Vdash (G, C)$ we will assume that if $I \in \mathcal{I}_i$, then $str(G) \leq i$ and C is $\mathcal{C}$-satisfiable. The relation $\Vdash$ is not defined otherwise. Formally, $\Vdash$ should be denoted $\Vdash_i$, because there is a forcing relation for each $\mathcal{I}_i$. We avoid the subindex in order to simplify the notation.

The following lemma establishes the monotonicity of the forcing relation.

Lemma 2. *Let $i \geq 1$ and $I_1, I_2 \in \mathcal{I}_i$ such that $I_1 \sqsubseteq_i I_2$. Then, for any $\Delta \in \mathcal{W}$, and $(G, C) \in \mathcal{G} \times \mathcal{SL}_\mathcal{C}$, it holds $I_1, \Delta \Vdash (G, C) \Longrightarrow I_2, \Delta \Vdash (G, C)$.*

Proof. The proof is inductive on the structure of G and it is derived from the definitions of the forcing relation and the order between interpretations. We only show the case of negation.

Assume $I_1, \Delta \Vdash (\neg A, C)$. Then, either $C \vdash_\mathcal{C} \neg C'$ for every C' such that $(A, C') \in I_1(\Delta)$, or there is no such C' and $C \equiv \top$. Since $str(\neg A) \leq i$, obviously $str(A) = j$, for some $j < i$. But then $[I_2(\Delta)]_j = [I_1(\Delta)]_j$, because $I_1 \sqsubseteq_i I_2$, and therefore $I_2, \Delta \Vdash (\neg A, C)$. □

Lemma 3. *Let $i \geq 1$ and let $\{I_n\}_{n \geq 0}$ be a denumerable family of interpretations over the stratum i, such that $I_0 \sqsubseteq_i I_1 \sqsubseteq_i I_2 \sqsubseteq_i \ldots$. Then, for any Δ, G and C, $\bigsqcup_{n \geq 0} I_n, \Delta \Vdash (G, C) \iff$ there exists $k \geq 0$ such that $I_k, \Delta \Vdash (G, C)$.*

Proof. The implication to the left is a consequence of Lemma 2, since $I_k \sqsubseteq_i \bigsqcup_{n \geq 0} I_n$ holds for any k. The converse is proved by induction on the structure of G, using the result of Lemma 1. We show one of the cases.

$(\exists x G')$ $\bigsqcup_{n \geq 0} I_n, \Delta \Vdash (\exists x G', C) \iff$ there is a variable y that does not occur free in Δ, $\exists x G'$, and C, such that $\bigsqcup_{n \geq 0} I_n, \Delta \Vdash (G'[y/x], C')$, and $C \vdash_\mathcal{C} \exists y C'$. By induction hypothesis, it holds $I_k, \Delta \Vdash (G'[y/x], C')$ for some $k \geq 0$. Therefore, there is a $k \geq 0$ such that $I_k, \Delta, \Vdash (\exists x G', C)$. □

Next, a continuous operator for every stratum transforming interpretations is defined. Its least fixed point supplies the expected version of truth at each stratum.

Definition 8. *Let $i \geq 1$ represent a stratum. The* operator $T_i : \mathcal{I}_i \longrightarrow \mathcal{I}_i$ *transforms interpretations over i as follows. For any $I \in \mathcal{I}_i$, $\Delta \in \mathcal{W}$, and $(A, C) \in At \times \mathcal{SL}_\mathcal{C}$, $(A, C) \in T_i(I)(\Delta)$ when:*

- *$(A, C) \in [I(\Delta)]_j$ for some $j < i$ or*
- *$str(A) = i$ and there is a variant $\forall \overline{x}(G \Rightarrow A')$ of a clause in $elab(\Delta)$, such that the variables $\overline{x}$ do not occur free in A, and $I, \Delta \Vdash (\exists \overline{x}(A \approx A' \wedge G), C)$.*

The crucial aspect of T_i is: For a database Δ, T_i incorporates information obtained exclusively from the clauses of Δ, whose heads are atoms of the stratum i, and the information of smaller strata remains invariable. Notice that if $str(A) = i$, then $str(\exists \overline{x}(A \approx A' \wedge G)) \leq i$ and T_i is well-defined.

In order to establish the existence of a fixed point of T_i, it will be proved to be monotonous and continuous.

Lemma 4 (Monotonicity of T_i). *Let $i \geq 1$ and $I_1, I_2 \in \mathcal{I}_i$ such that $I_1 \sqsubseteq_i I_2$. Then, $T_i(I_1) \sqsubseteq_i T_i(I_2)$.*

Proof. Let us consider any Δ and $(A, C) \in T_i(I_1)(\Delta)$. This implies that $str(A) \leq i$. If $str(A) = j < i$, then $(A, C) \in [I_1(\Delta)]_j = [I_2(\Delta)]_j$, because $I_1 \sqsubseteq_i I_2$ and $j < i$. Hence $(A, C) \in T_i(I_2)(\Delta)$, by definition of T_i. If $str(A) = i$, then there is a

variant $\forall\overline{x}(G \Rightarrow A')$ of a clause of Δ, such that the variables $\overline{x}$ do not occur free in A, and $I_1, \Delta \Vdash (\exists\overline{x}(A \approx A' \wedge G), C)$. Using Lemma 2 and the fact that $I_1 \sqsubseteq_i I_2$, we obtain $I_2, \Delta \Vdash (\exists\overline{x}(A \approx A' \wedge G), C)$, which implies $(A, C) \in T_i(I_2)(\Delta)$, by definition of T_i. □

Lemma 5 (Continuity of T_i). *Let $i \geq 1$ and $\{I_n\}_{n\geq 0}$ be a denumerable family of interpretations over i, such that $I_0 \sqsubseteq_i I_1 \sqsubseteq_i I_2 \sqsubseteq_i \ldots$. Then $T_i(\bigsqcup_{n\geq 0} I_n) = \bigsqcup_{n\geq 0} T_i(I_n)$.*

Proof. The inclusion $\supseteq$ is a consequence of the monotonicity of T_i. Let us prove the inclusion $\subseteq$. Consider any Δ and $(A, C) \in T_i(\bigsqcup_{n\geq 0} I_n)(\Delta)$. Then $str(A) \leq i$. If $str(A) = j < i$, $(A, C) \in [T_i(\bigsqcup_{n\geq 0} I_n)(\Delta)]_j = [I_0(\Delta)]_j$, then $(A, C) \in T_i(I_0)(\Delta) \subseteq \bigcup_{n\geq 0} T_i(I_n)(\Delta) = (\bigsqcup_{n\geq 0} T_i(I_n))(\Delta)$. If $str(A) = i$, there is a variant $\forall\overline{x}(G \Rightarrow A')$ of a clause of Δ, such that the variables $\overline{x}$ do not occur free in A, and $\bigsqcup_{n\geq 0} I_n, \Delta \Vdash (\exists\overline{x}(A \approx A' \wedge G), C)$. Thanks to Lemma 3, there exists $k \geq 0$, such that $I_k, \Delta \Vdash (\exists\overline{x}(A \approx A' \wedge G), C)$, and therefore $(A, C) \in T_i(I_k)(\Delta)$. As a consequence, also in this case $T_i(\bigsqcup_{n\geq 0} I_n)(\Delta) \subseteq \bigcup_{n\geq 0} T_i(I_n)(\Delta) = (\bigsqcup_{n\geq 0} T_i(I_n))(\Delta)$. □

Proposition 1. *The operator T_1 has a least fixed point, which is $\bigsqcup_{n\geq 0} T_1^n(I_\perp)$, where the interpretation $I_\perp$ represents the constant function $\emptyset$.*

Proof. By the Knaster-Tarski fixed point theorem [19], using Lemma 5. □

Let fix_1 denote $\bigsqcup_{n\geq 0} T_1^n(I_\perp)$, i.e., the least fixed point at stratum 1.

Consider now the following sequence $\{T_2^n(fix_1)\}_{n\geq 0}$ of interpretations in $(\mathcal{I}_2, \sqsubseteq_2)$. Using the properties of T_i, it is easy to prove by induction on $n \geq 0$ that this sequence is a chain

$$fix_1 \sqsubseteq_2 T_2(fix_1) \sqsubseteq_2 T_2(T_2(fix_1)) \sqsubseteq_2 \ldots, \sqsubseteq_2 T_2^n(fix_1), \ldots$$

As before, in accordance with Lemmas 1 and 5, $\{T_2^n(fix_1)\}_{n\geq 0}$ has a least upper bound, $\bigsqcup_{n\geq 0} T_2^n(fix_1)$, in $(\mathcal{I}_2, \sqsubseteq_2)$ that is a fixed point of T_2, denoted by fix_2. Proceeding successively on the same way, a chain:

$$fix_{i-1} \sqsubseteq_i T_i(fix_{i-1}) \sqsubseteq_i T_i(T_i(fix_{i-1})) \sqsubseteq_i \ldots, \sqsubseteq_i T_i^n(fix_{i-1}), \ldots$$

can be defined for any stratum $i > 1$, and a fixed point of it

$$fix_i = \bigsqcup_{n\geq 0} T_i^n(fix_{i-1})$$

can be found.

In particular, if $str(\Delta) = k$, we simplify fix_k writing fix. Then, $fix(\Delta)$ represents the pairs (A, C) such that A can be deduced from Δ if C is satisfied. Notice that $fix(\Delta)$ is computed by saturating strata sequentially from $fix_1(\Delta)$ up to $fix_k(\Delta)$, using for every i only the clauses of the stratum i.

4.2 Soundness and Completeness

The fixed point semantics defined in [7] for *HH(C)* was proved to be sound and complete with respect to the calculus $\mathcal{UC}$. Our interest now is to prove soundness and completeness of the new fixed point semantics for $HH_\neg(\mathcal{C})$, with respect to the extended calculus $\mathcal{UC}_\neg$. This means that the forcing relation, considering the least fixed point at the last stratum of a database and a query, coincides with derivability in $\mathcal{UC}_\neg$. More precisely, if $str(G) = i$, (G, C) is forced by fix_i in the context of Δ if and only if C is an answer constraint of G from Δ.

Without negation, any database Δ and query G have a stratification with only one stratum. If this is the case, soundness and completeness are similar to those results for *HH(C)*.

Proposition 2. *For every $\Delta \in \mathcal{W}$, and every pair $(G, C) \in \mathcal{G} \times \mathcal{SL}_\mathcal{C}$, if $str(G) = 1$ then: $fix_1, \Delta \Vdash (G, C) \iff \Delta; C \vdash_{\mathcal{UC}_\neg} G$.*

Proof. The proof is an adaptation of those presented in [7] to the definition of the forcing relation defined now for $HH_\neg(\mathcal{C})$. Notice that, since we are assuming that $str(G) = 1$, then the case $G \equiv \neg A$ has not to be considered. □

Now, we consider the general case.

Theorem 1 (Soundness and Completeness). *For every $i \geq 1$, $\Delta \in \mathcal{W}$, and every pair $(G, C) \in \mathcal{G} \times \mathcal{SL}_\mathcal{C}$, if $str(G) \leq i$ then:*

$$fix_i, \Delta \Vdash (G, C) \iff \Delta; C \vdash_{\mathcal{UC}_\neg} G.$$

Proof. By induction on i. Proposition 2 is the proof of the case $i = 1$.

For $i > 1$, assume the induction hypothesis: for every Δ, G, C, with $str(G) \leq i - 1$: $fix_{i-1}, \Delta \Vdash (G, C) \iff \Delta; C \vdash_{\mathcal{UC}_\neg} G$.

The proof is analogous to the base case, except for $\neg A$. Let us analyze this case: $fix_i, \Delta \Vdash (\neg A, C) \iff$ for every C' such that $(A, C') \in fix_i(\Delta)$, it holds $C \vdash_\mathcal{C} \neg C'$, or there is no such C' and $C \equiv \top$. Obviously, $str(\neg A) \leq i - 1$, then the previous sentence is equivalent to say that for every C' such that $fix_{i-i}, \Delta \Vdash (A, C')$, it holds $C \vdash_\mathcal{C} \neg C'$, or there is no such C' and $C \equiv \top$. Applying the induction hypothesis, it is equivalent to say that either for every C' such that $\Delta; C' \vdash_{\mathcal{UC}_\neg} A$ and $C \vdash_\mathcal{C} \neg C'$ holds, or there is not such C' and $C \equiv \top$. This is equivalent to $\Delta; C \vdash_{\mathcal{UC}_\neg} \neg A$.

As a consequence of this theorem: $(A, C) \in fix(\Delta) \iff \Delta; C \vdash_{\mathcal{UC}_\neg} A$. This means that the atoms in the fixed point of a database are those that can be derived by the calculus.

The advantage of this fixed point semantics over the proof theoretic one is that it can be considered as the formal basis of particular implementations of database systems based on $HH_\neg(\mathcal{C})$. The prototype presented in the next section is a proof of it. Notice that the previous formalisms are defined for a generic constraint system $\mathcal{C}$ as a black box for which the existence a solver that checks $\mathcal{C}$-satisfiability has been assumed. The complexity of an implementation will depend on the particular instance domain and solver.

5 Implementing an Instance

In this section, we briefly report a Prolog-implemented prototype of $HH_{\neg}(\mathcal{D})$, where $\mathcal{D}$ is a basic finite domain constraint system with equality and disequality. It has as input a database Δ and, if stratifiable, computes the set $fix(\Delta)$. Δ contains not only the extensional and intensional databases, but also the translation of the user query G into a clause `goal(`$\overline{X}$`) :-` G, where $\overline{X}$ are the free variables of G. So, if the computed $fix(\Delta)$ contains for instance the pair `(goal(`$X_1, \ldots, X_n$`),` $X_1 \approx t_1 \wedge \ldots \wedge X_n \approx t_n$`)`, then $X_1 \approx t_1 \wedge \ldots \wedge X_n \approx t_n$ is an answer to G.

Computing the outcome corresponding to a set Δ follows some stages: 1) Build the dependency graph, 2) Compute a stratification (if there is any), and, if succeeds, 3) Compute $fix(\Delta)$ as a set of pairs (A, C) for the stratification. For the first stage, the algorithm in [14] has been used. For the second one, the dependency graph is used as an input to an algorithm following Definition 4.

A more elaborated computation is needed for the third stage: $fix(\Delta)$ is computed sequentially, from $fix_1(\Delta)$ up to $fix_k(\Delta)$, where $k = str(\Delta)$. When computing the information of stratum i, only pairs (A, C) such that $str(A) = i$ are calculated, and the information of smaller strata remains invariable. Following Definition 8, the successive iterations of the fixpoint operator T_i deliver new pairs which are obtained by considering the pairs deduced in previous iterations in the context of Δ and every ground instance of the clauses defining predicates of stratum i. As T_i is monotonous (Lemma 4) and we deal with a finite domain constraint system, a terminating loop finds all the pairs for a given $fix_i(\Delta)$. The set $fix_i(\Delta)$ is completely evaluated when no pair (A, C) is added after an iteration of T_i. Therefore, the stratum i is *saturated* and the computation of $fix_{i+1}(\Delta)$ begins, applying T_{i+1} to the just calculated $fix_i(\Delta)$.

The forcing relation is implemented by means of the Prolog predicate `force` that makes calls to the constraint solver, which solves constraints for the particular finite domain system.

However, care has to be taken when programming `force` for the case $D \Rightarrow G$. Following Definition 7, G has to be proved in the context of the database augmented with the clause D. This turns out to be more complex to be computed, since the head of the clause added to the current context may belong to stratum j, where $j < i$ and, therefore, $fix_j(\Delta \cup \{D\})$ must be calculated, which we call a subcomputation level (level in short) involving the local clause D. Once G is solved, both D and the deduced pairs are ignored for the rest of the computation.

The pairs proved at each iteration of the fixpoint operator as well as the database clauses are stored as Prolog facts. As there may be facts and clauses belonging to different levels, we identify them with a number for the corresponding level. The main level is identified as 0. For example, when using a clause as `p(a) :- q(a,b) => r(b)` at level l, the goal `q(a,b) => r(b)` must be forced, for which the local clause `q(a,b)` is added at level $l + 1$. Now, the goal `r(b)` will be tried to be forced at level $l + 1$. When this is done, `q(a,b)` is removed from the database and the computation returns to level l, where a pair corresponding to `p(a)` will be added to the current fixed point if `r(b)` was forced in that augmented context.

As an executable example, let us consider again Example 1, where real constraints have been removed. The query: "Assuming a flight from Paris to London, what cities are reachable from Madrid?" can be represented as $G \equiv$ `flight(par,lon) => trip(mad,X)`. The clause `goal(X):-` G is added to the database for solving G. Executing this example delivers the following meaning:

`[...,(goal(X),X` $\approx$ `par),(goal(X),X` $\approx$ `ny),(goal(X),X` $\approx$ `lon),...]`

Therefore, we can conclude that it is possible to travel to Paris, New York and London assuming that flight.

6 Conclusions

We have studied the application of the constraint logic programming scheme $HH(\mathcal{C})$ as a *CDDB* system. With this purpose, this scheme has been extended with a suitable formulation of negation. The result provides a database language more expressive than former ones [15,18,17,5], because $HH_{\neg}(\mathcal{C})$ owns constraints as well as implication and quantifiers altogether.

Two semantics, based on proof theory and fixed point techniques, have been defined to formalize the system, and proved to be equivalent for stratifiable databases. Both semantics are interesting *per se*. The former allows to represent the meaning of a database query by means of a derived constraint answer. In addition, the uniformity of the sequent calculus that governs $HH_{\neg}(\mathcal{C})$ is preserved, because only a rule introducing the connective $\neg$ on the right (but not on the left) of the sequent is added. Then, proofs remain guided by the structure of the goal. However, it can not be considered as a practical operational semantics. Some aspects, as finiteness of the set of answers for an atom A when $\neg A$ is computed, are obviated.

The fixed point semantics relies on stratified negation and constitutes a formal basis for practical implementations. Stratification has been defined as a syntactical criterion to determine if a query database can be potentially be computed in a finite number of steps. When $\neg A$ are going to be proved, the stratum of A has been previously saturated and $\neg A$ can be correctly computed.

The prototype introduced at the end of this paper is based in the formal mechanisms that support this semantic approach for a concrete constraint system. In addition, this semantics supplies a framework in which properties of databases can be analyzed. For instance, if Δ_1, Δ_2 are two stratifiable databases (considering the same stratification for simplicity), it can be said that Δ_1 and Δ_2 are equivalent if $fix(\Delta_1) = fix(\Delta_2)$.

Regarding future work: First, investigate the relaxation of the strong requirement about program stratification, as done in answer set programming [6], also extended to include constraints [4,12]. Second, analyze the requirements that should be imposed to $\mathcal{C}$ in order to obtain a safe instance of $HH_{\neg}(\mathcal{C})$, as done via safety levels in constraint database languages [17]. Finally, develop the current prototype implementation to deal with other particular instances based on useful constraint systems handling combined constraint domains [8].

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Apt, K., Bol, R.: Logic Programming and Negation: A Survey. Journal of Logic Programming 19&20, 9–71 (1994)
3. Benedikt, M., Libkin, L.: Safe constraint queries. In: PODS 1998: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 99–108. ACM Press, New York (1998)
4. Bonatti, S.B.P.A., Gelfond, M.: Towards an integration of answer set and constraint solving. In: Gabbrielli, M., Gupta, G. (eds.) ICLP 2005. LNCS, vol. 3668, pp. 52–66. Springer, Heidelberg (2005)
5. Bonner, A.J., McCarty, L.T., Vadaparty, K.: Expressing Database Queries with Intuitionistic Logic. In: Lusk, E.L., Overbeek, R.A. (eds.) Proceedings of the North American Conference on Logic Programming, pp. 831–850 (1989)
6. Ferraris, P., Lifschitz, V.: Mathematical foundations of answer set programming. In: Artëmov, S.N., Barringer, H., d'Avila Garcez, A.S., Lamb, L.C., Woods, J. (eds.) We Will Show Them (1), pp. 615–664. College Publications (2005)
7. García-Díaz, M., Nieva, S.: Providing Declarative Semantics for HH Extended Constraint Logic Programs. In: Proceedings of the 6th ACM SIGPLAN Int. Conf. on PPDP, pp. 55–66 (2004)
8. Hofstedt, P., Pepper, P.: Integration of declarative and constraint programming. Theory Pract. Log. Program. 7(1-2), 93–121 (2007)
9. Jaffar, J., Lassez, J.-L.: Constraint Logic Programming. In: 14th ACM Symp. on Principles of Programming Languages (POPL 1987), Munich, Germany, January 1987, pp. 111–119. ACM Press, New York (1987)
10. Leach, J., Nieva, S., Rodríguez-Artalejo, M.: Constraint Logic Programming with Hereditary Harrop Formulas. TPLP 1(4), 409–445 (2001)
11. Lipton, J., Nieva, S.: Higher-order logic programming languages with constraints: A semantics. In: Della Rocca, S.R. (ed.) TLCA 2007. LNCS, vol. 4583, pp. 272–289. Springer, Heidelberg (2007)
12. Mellarkod, V.S.: Integrating ASP and CLP Systems: Computing Answer Sets from Partially Ground Programs. PhD thesis, Texas Tech University (2007)
13. Miller, D., Nadathur, G., Pfenning, F., Scedrov, A.: Uniform Proofs as a Foundation for Logic Programming. Annals of Pure and Applied Logic 51, 125–157 (1991)
14. Nieva, S., Sáenz-Pérez, F., Sánchez, J.: Towards a constraint deductive database language based on hereditary harrop formulas. In: Lucio, P., Orejas, F. (eds.) Sextas Jornadas de Programación y Lenguajes, PROLE, pp. 171–182 (2006)
15. Pustejovsky, J., Revesz, P.Z. (eds.): Proc. 13th International Symposium on Temporal Representation and Reasoning. IEEE Computer Society Press, Los Alamitos (2006)
16. Revesz, P.Z.: Datalog and Constraints. In: Kuper, G., Libkin, L., Paredaens, J. (eds.) Constraint Databases, ch. 7, pp. 151–174. Springer, Heidelberg (2000)
17. Revesz, P.Z.: Introduction to Constraint Databases. Springer, Heidelberg (2002)
18. Scholl, P.R.M., Voisard, A.: Spatial databases with application to GIS. Morgan Kaufmann Publishers Inc., San Francisco (2002)
19. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5, 285–309 (1955)
20. Zaniolo, C., Ceri, S., Faloutsos, C., Snodgrass, R.T., Subrahmanian, V.S., Zicari, R.: Advanced Database Systems. Morgan Kaufmann Publishers Inc., San Francisco (1997)