Queries, constraints, updates and transactions within a logic-based language

Danilo Montesi[1], Elisa Bertino[2]

1) Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza L. da Vinci 32, 20133 Milano, Italy
montesi@ipmel2.elet.polimi.it

2) Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova
Viale Benedetto XV, 3 16132 Genova, Italy
bertino@disi.unige.it

Abstract

In this paper we develop a logic-based language to express queries, constraints, updates and transactions. The language supports the definition of static constraints and extensional updates in database rules or in the transactions. The constraints defined in a transaction are temporary, that is they hold only for that transaction, while those defined in database rules are permanent, that is they hold forever. The motivation of using a unique language is motivated by pragmatic and theoretical considerations, that is a uniform computational model of this language and semantics transaction optimization. Finally we provide a two steps semantics for our language modeling the transactional behavior.

Keywords: Deductive database, updates, constraints and transactions.

1 Introduction

In recent years, logic-based databases have been the focus of intense research, which has resulted in strong advances in theory, systems and applications. The major advantages of deductive databases are then formal setting based on declarative languages [10] to express data, views, queries and integrity constraints and then simple computational models for query evaluation, such as top-down or bottom-up evaluation [1]. However, updates and transactions are difficult to express with logic languages. Several approaches have been proposed in order to integrate updates and transactions with declarative query languages. Almost all these approaches impose a computational model which is top-down if updates are defined in rule bodies and bottom-up in updates are defined in rule heads.

Recently a new approach to avoid the above drawbacks has been introduced in [3, 14]. Updates are expressed in rule bodies and have a non-immediate semantics. They are computed in two phases. During the first phase (called the marking phase) updates are collected and their consistency is checked. These updates are hypothetical since they are not actually applied to the database. In the second phase, (called the update phase) the collected updates are executed altogether modeling a transactional behavior. The main advantage of this approach is to avoid expensive runtime rollback due to transaction failures and to model the transactional behavior of computations. Indeed, if there is a failure the transaction is not executed and therefore is not need to undo the updates which have not yet been executed. Moreover, top-down and bottom-up evaluation are reconciliated and both evaluation methods can be used during the marking phase.

We extend this approach allowing also constraints in rule bodies together with updates. The main motivations of this extension is to express through the same language structural and behavioral features, to model queries, constraints, updates and transactions within a uniform scheme avoiding expensive runtime rollback due to transaction failures. This scheme provides a clear interaction among update, constraint, query and transaction languages. In addition, both top-down and bottom-up evaluation methods can also be used considering constraints. Moreover, our approach allow to change the integrity constraints. Indeed, those defined into the database are permanent and should not change. By contrast, those defined in the transaction can change and therefore are temporary. Indeed there are many example in which we some of the constraints are permanent and therefore will never change. Instead other constraints have a shorter life, that is they can hold only within a transaction or they must hold only for a specific kind of transaction. Finally, semantic transaction optimization are possible extending the approach to semantic query optimization considered in [7] and the approach to transaction optimization considered in [2]. The above features lead to a powerful rule language to express knowledge bases with updates and permanent and/or temporary constraints.

In our proposal constraints and updates have a non-immediate semantics following the guidelines of updates with non-immediate semantics, that is, the computation is performed in two phase. During the first phase, updates and constraints are collected while updates consistency and constraints consistency are checked. Updates are hypothetical since they are not actually applied to the database. In the

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

CIKM '93 - 11/93/D.C., USA

© 1993 ACM 0-89791-626-3/93/0011 ....$1.50

second phase, the collected constraints are checked in the current state and then the collected updates are executed altogether if the new database state will satisfies the collected constraints. If constraints checking fails before or after updates execution then the transaction aborts.

The remainder of this paper is organized as follow. In the following we introduce informally the approach of non-immediate semantics for updates. Section 2 introduces the language to express queries, constraints, updates and transactions and its informal behavior by means of several examples. Section 3 introduces formally the U-C-Datalog language. Section 4 presents the semantics for such language in two steps modeling the transactional behavior. Finally, some conclusions are given in Section 5.

1.1 Non-immediate semantics for updates

A database consists of a set of ground base relations called *extensional* database (EDB) and a set of virtual relations defined using rule, called *intensional* database (IDB). In order to express updates to base relations in Datalog language [6], we consider updates in rule bodies. The *state* of a database is the EDB and we denote the updates (to the extensional database) as in $\mathcal{LDL}$ [16]. Intuitively, the meaning of $-p(\tilde{t})$ ($+p(\tilde{t})$) is to delete (insert) tuple $\tilde{t}$ from (into) the base relation p. The non-immediate semantics of updates is introduced by means of the following example.

Example 1.1 *Consider the database $DB_1 = IDB \cup EDB_1$ where*

$$EDB_1 = \mathtt{q(b)}$$

and the intensional database IDB is

$$\begin{aligned} IDB = \quad & \mathtt{p(X)} \leftarrow -\mathtt{q(X)}, \mathtt{q(X)} \\ & \mathtt{r(X)} \leftarrow +\mathtt{t(X)}, \mathtt{p(X)} \\ & \mathtt{s(X)} \leftarrow \mathtt{t(X)} \\ & \mathtt{k(X)} \leftarrow +\mathtt{q(X)} \end{aligned}$$

The goal $G_1 = ?\mathtt{r(X)}$ is computed in two phases. The first one collects the binding $\mathtt{X} = \mathtt{b}$ and the tentative updates $-\mathtt{q(b)}$ and $+\mathtt{t(b)}$. During the second phase the updates are executed. Informally the new extensional database $EDB_2 = \{\mathtt{t(b)}\}$ is the result of the application of those updates to EDB_1. The goal $G_2 = ?\mathtt{s(X)}$, evaluated in $DB_2 = IDB \cup EDB_2$, collects the binding $\mathtt{X} = \mathtt{b}$, and does not compute any update and thus the extensional database is still EDB_2. The goal $G_3 = ?\mathtt{k(b)}$ evaluated in DB_2 collects the tentative update $+\mathtt{q(b)}$ computing the new state $EDB_3 = \{\mathtt{t(b)}, \mathtt{q(b)}\}$. $G_4 = ?\mathtt{k(X)}, \mathtt{p(X)}$ collects the tentative updates $+\mathtt{q(b)}, -\mathtt{q(b)}$. They are inconsistent therefore the computation of the goal aborts leaving the database in the current database state.

◇

Note that insertions and removals are not immediately triggered as in $\mathcal{LDL}$. The computation is performed in two steps [4]. The first phase computes the bindings and collect updates while checking update consistency. By updates consistency we mean that complementary updates are not allowed. Moreover, the updates collected in the first phase form a set. By contrast, in DLP [13] and $\mathcal{LDL}$ they are considered as a sequence. Therefore, rule bodies with updates are all evaluated, in our approach, in the same state, namely on the state prior to any change. Only after a successful computation of the first phase, updates are executed. Note that we will consider only databases which are safe through goal invocation, that is databases for which the rule are safe [18] (i.e., the variable in the head also appear in the body) or became safe through goal invocation (i.e., the query provides the binding. We refer to a goal (or query) also as a *transaction* to give evidence to the transactional behavior of a goal (or query). The semantics of the above language, modeling the transactional behavior is given in [3, 14].

2 Updates, constraints and query languages

Updates and constraints can be defined either outside or within the query language. In Glue-Nail [15] updates and queries are defined by means of two separate language. By contrast, RDL1 [12] and $\mathcal{LDL}$ [16] merge the two languages. Even if in logic-based databases the constraints can be expressed as rules, constraints and queries have been separate issues. Indeed the classic view consider the constraints as "knowledge" (or meta data). This implies that constraints are a separate component of the database, that is they are considered as an integrity constraint theory that the database (expressing data) must satisfy. Therefore, even if they can be expressed by means of the same language, they are conceptually different.

Example 2.1 *Consider the database components IDB and IC*

$$\begin{aligned} IDB = \quad & \mathtt{Hsaledept(Depno,Mgrno,Floorno,Item,Vol)} \leftarrow \\ & \qquad \mathtt{Dept(Depno,Mgrno,Floorno)}, \\ & \qquad \mathtt{Sale(Depno,Item,Vol)}, \mathtt{Vol} > \mathtt{80000}. \end{aligned}$$

$$IC = \quad \leftarrow \mathtt{Dept(X,Y,f2)}, \mathtt{Dept(X,Y,f6)}$$

IC represents the integrity constraint theory expressing the condition that "no departments can be on the floor `f2` *and* `f6`*".*

◇

Constraints are expressed through rules. Constraint updates are difficult to model. This is the same problem of view updates. Constraint and view updates are not modeled by first order logic [5]. The motivations to consider updates, constraints and queries expressed into a single language are practical and theoretical. From a practical point of view, the database designer needs to understand and handle only one language. Moreover, by considering constraints in rule bodies and goals (or transactions) we support changes to the constraints. Many applications need to change the constraints. For example during the prototypal phase of a database several changes to constraints can simulate different integrity constraints theories. Finally, constraints can be used for optimization, if they are merged with updates and queries. Indeed this is the trend of many rule-based database languages such as Starburst [19] and HiPAC [9]. The above language are based on active rules, therefore its easier to express updates, queries and constraints, but the price to pay is to lack a formal semantics and the nice setting of goal driven computation. The theoretical motivation of our approach is to have a clear interactions among the constraint, update, query and transaction languages while preserving the top-down and bottom-up evaluation schemes. Note that active databases lack the notion of queries and the computational model is only bottom-up. In the following we will consider updates defined in rule bodies.

Example 2.2 *Consider the rule*

```
Movedept(Depno,From,To) ←
        Dept(Depno,Mgrno,From),
        −Dept(Depno,Mgrno,From),
        +Dept(Depno,Mgrno,To)
```

The rule moves a department from one floor to another according to values given into the goal. For example, the goal ?Movedept(56,f2,f4) *moves the department number* 56 *from the floor* f2 *to the* f4.

◇

A database with updates, queries and constraints can be defined considering as new intensional database the rules of Examples 2.1 and 2.2 and as IC the constraint of Example 2.1.

Example 2.3 *Consider the database*

```
IDB =  Hsaledept(Depno,Mgrno,Floorno Item,Vol) ←
            Dept(Depno,Mgrno,Floorno),
            Sale(Depno,Item,Vol),Vol > 80000
       Movedept(Depno,From To) ←
            Dept(Depno,Mgrno,From),
            −Dept(Depno,Mgrno,From),
            +Dept(Depno,Mgrno,To)
```

```
IC =  ← Dept(X,Y,f2),Dept(X,Y,f6)
```

IC represents the integrity constraints theory. ◇

The basic idea of integrity constraints is that only certain states are considered acceptable and IC is meant to enforce these legal states. We believe that IC should also be integrated with the query-update language in order to associate the relevant part of the constraint theory with the updates which could potentially violate the constraints. Therefore, we move the constraints from the status of being *global* where they should be checked for any update (apart from optimization methods [17]) into the status of *local* constraints, i.e., into the rule potentially responsible for the constraints violation. Note that global constraints can still be supported using repeating all the constraints into all rules.

Example 2.4 *Consider the database*

```
EDB =  Dept(9,2,f2)
       Dept(6,1,f4)
       Sale(9 shoe,90000)
```

```
IDB^IC =  Hsaledept(Depno,Mgrno,Floorno,Item,Vol) ←
              Dept(Depno,Mgrno,Floorno),
              Sale(Depno,Item,Vol),Vol > 80000
          Movedept(Depno,From,To) ←
              (¬Dept*(X,Y,f2)∨
              ¬Dept*(X,Y,f6)),
              Dept(Depno,Mgrno,From),
              −Dept(Depno,Mgrno,From),
              +Dept(Depno,Mgrno,To)
```

The second rule express in its body the constraint IC[1]*. We can also specify a constraint into a goal. Consider*

```
?  (¬Dept*(X,Y,f3) ∨ ¬Dept*(X,Y,f4)),
   Movedept(Depno,f0,f3).
```

The constraints in the query express the fact that no department can be on the floor f3 *and* f4*. In addition, the constraint in the database expresses conditions for floors* f2 *and* f6*. The first phase collects the bindings (for the variable of the query minus those of the constraints), the updates and the constraints:*

- Depno = 6, From = f4, To = f6
- −Dept(6,1,f4), +Dept(6,1,f6)
- ¬Dept*(X,Y,f2) ∨ ¬Dept*(X,Y,f6)
  ¬Dept*(X,Y,f3) ∨ ¬Dept*(X,Y,f4)

The constraints are satisfied in the EDB, but they are not satisfied in $EDB' = EDB \cup$ Dept(6,1,f6) \ Dept(6,1,f4). Therefore, the goal aborts.

◇

Note that the classic hypothesis that IC is satisfied by the current database state does not hold in our approach. Indeed, due to temporary constraints which can be added we need always to check whether the current state of the database satisfies the constraints. Considering a set oriented behavior, the marking phase collect (besides to bindings) several updates and constraints due to different successful derivations.

Example 2.5 *Consider the database $DB_1 = IDB \cup EDB_1$ where*

$$EDB_1 = \mathtt{q(b)}$$

and the intensional database IDB is

```
IDB =  p(X) ← −q(X),q(X)
       r(X) ← +t(X),(¬t*(A) ∨ ¬q*(A)),p(X)
       s(X) ← t(X).
       k(X) ← +q(X),(¬q*(a))t(X)
       k(X) ← +v(X),(¬v*(A) ∨ ¬q*(A)),t(X).
```

The goal G_1 = ?r(X) is computed in two phases. The former one collects the the binding X = b*, the tentative updates* −q(b), +t(b) *and the constraint theory* (← t(A), q(A))*. During the second phase the updates are executed because they do not violate the constraint theory. Informally the new extensional database EDB_2 = {t(b)} is the result of the application of those updates to EDB_1. The goal G_2 = ?(¬t*(b)), s(X) evaluated in $DB_2 = IDB \cup EDB_2$ fails due to the violation of the constraint theory* (← t(b)) *and thus the extensional database is still EDB_2. The goal G_3 = ?k(X) collects the binding* X = b*, the tentative update* +q(b), +v(b) *and the constraint theory*

```
←  v(A),q(A)
←  q(a)
```

which is satisfied in EDB_2 but not in EDB_3 = {t(b), q(b), v(b)}. Therefore the goal fails.

◇

[1] The constraint $\leftarrow A_1 \wedge \ldots \wedge A_n$ can be expressed as $\neg A_1^* \vee \ldots \vee A_n^*$. In the following we use these two notations equivalently. Moreover, we use the * to denote the atoms of the constraint instead that the atom of an ordinary rule. This is related to the fact that we are merging data and meta-data.

3 Just one language

In Section 2 we considered an extension of Datalog language with updates and constraints defined in rule bodies. Such language is an instance of the Constraint Logic Programming (CLP) scheme [8, 11]. We recall that CLP scheme provides a formal framework for reasoning with and about constraints in the rule-based context of logic programming. Within this framework, a class of rule-based customized languages can be generated to deal with constraints over a specific domain of computation. Indeed this is our approach. The key idea here is that the important properties of Horn clauses do not depend on the Herbrand universe or unification. These semantic properties hold for many domains. Consequently even though one deals with different languages working in different domains, they all share fundamental semantic properties and form a well defined class of languages.

The update and constraint parts of rule languages introduced informally in Section 2 are but a special kind of constraints over a specific domain of computation in CLP. Our extension of Datalog is a syntactic restriction of CLP over function-free finite domains. Syntactic restriction is due to the fact that relations have to be either extensionally or intensionally defined (see Definition 3.4). Before introducing the language, we recall that the Examples of Section 2 have shown that the the language to express queries, constraints, updates and transactions should have the following characteristics.

1. Updates and constraints are expressed in rule bodies. The rule must also compute the bindings for the variables of the rule which are not in the constraint part.
2. There is no order among atoms in rule bodies.
3. The marking phase, that is, the answering process must compute the bindings for the variables, collect the set of constraints and the set of hypothetical updates.

In the following we introduce a rule-based language corresponding to the language informally introduced by means of the examples in Section 2.

We consider a fixed (Π, Σ, V)-language. Programs and queries can use only the finite set of predicate symbols Π, the finite set of constant symbols Σ and the variables of V. We assume that the language contains enough variables, constant symbols and predicates to write a program and to effectively query it. The set of predicate symbols Π is partitioned into four pairwise disjoint sets of symbols: Π^e the *extensional predicate symbols*, Π^i the *intensional predicate symbols*, the Π^u the *update predicate symbols*, the Π^c the *constraint predicate symbols* and Π^b the *bindings predicate symbol*. Considering only extensional updates, and extensional constraints the sets Π^u and Π^c are built over the extensional predicates, that is, $\Pi^u = \{+p, -p \mid p \in \Pi^e\}$ and $\Pi^c = \{\neg p^* \mid p \in \Pi^e\}$. The only binding predicate considered is $=$, therefore $\Pi^b = \{=\}$. The set of variable symbols V is partitioned into two pairwise disjoint sets of symbols: V^c the *constraint variable symbols* and V^i the *variable symbols*.

We denote the atoms built from the sets Π, Σ and V as (Π, Σ, V)-atoms. In a similar way we denote ground extensional relations as (Π^e, Σ)-atoms, intensional relations as (Π^i, Σ, V^i)-atoms. An update (atom) is a (Π^u, Σ, V^i)-atom, a constraint (atom) is a (Π^c, Σ, V^c)-atom a binding (atom) is a (Π^b, Σ, V^i)-atom and a query atom is a $(\Pi^i \cup \Pi^e, \Sigma, V^i)$-atom.

Definition 3.1 *(Extensional database) Let $\mathcal{B}^e$ be the Herbrand base built over Π^e and Σ, then a state or extensional database $EDB \in 2^{\mathcal{B}^e}$ is a (possibly empty) set of (Π^e, Σ)-atoms, i.e. ground base relations.*

In the following we denote with $EDB_i, i = 1, \ldots, n$ the possible extensional databases.

Definition 3.2 *(Intensional database) The IDB is a set of (constrained) rules of the form*

$$H \leftarrow b_1 \wedge \ldots \wedge b_k \wedge u_1 \wedge \ldots \wedge u_s \wedge c_1 \vee \ldots \vee c_r \wedge B_1, \ldots, B_t.$$

where $B_1 \wedge \ldots \wedge B_t$ (as in Datalog) is the query part, $u_1 \wedge \ldots \wedge u_s$ is the update part, $b_1 \wedge \ldots \wedge b_k$ is the binding part and $c_1 \vee \ldots \vee c_r$ is the constraint part[2]*. B_i's are $(\Pi^i \cup \Pi^e, \Sigma, V^i)$-atoms and H is a (Π^i, Σ, V^i)-atom. b_h's are (Π^b, Σ, V^i)-atoms, u_j's are (Π^u, Σ, V^i)-atoms c_l's are (Π^c, Σ, V^c)-atoms. The update and query parts cannot be both empty.*

The intuitive meaning of a rule is: "if $B_1, \ldots, B_t$ is true, the bindings $b_1, \ldots, b_k$, the updates $u_1, \ldots, u_s$ and the constraints $c_1, \ldots, c_r$ are solvable, then H is true". The notion of solvability intuitively express the fact that the bindings $\mathtt{X} = \mathtt{bob}, \mathtt{X} = \mathtt{tom}$ are not solvable, while the bindings $\mathtt{X} = \mathtt{Y}, \mathtt{Y} = \mathtt{bob}$ are solvable. Similarly, the updates $+\mathtt{p}(\mathtt{X})$, $-\mathtt{p}(\mathtt{X})$, i.e. complementary updates, are not solvable. The updates $+\mathtt{p}(\mathtt{Y}), -\mathtt{p}(\mathtt{X})$ could be solvable if the related bindings are for example $\mathtt{X} = \mathtt{tom}, \mathtt{Y} = \mathtt{bob}$. With the bindings $\mathtt{X} = \mathtt{tom}, \mathtt{Y} = \mathtt{tom}$, $+\mathtt{p}(\mathtt{Y})$, $-\mathtt{p}(\mathtt{X})$ are not solvable. Note that constraints are never inconsistent. Indeed our approach allow to incrementally specify constraints.

Definition 3.3 *(Query) A query is a constrained rule with no head of the form*

$$b_1, \ldots, b_k, u_1, \ldots, u_s, c_1, \ldots, c_r, B_1, \ldots, B_t$$

where B_i's, u_j's, b_h's and c_l's are as in Definition 3.2 and $B_1, \ldots, B_t$ cannot be empty.

Following the tradition in the examples we always prefix a query with the symbol '?'. In the following the words query and update query are considered as synonym.

Definition 3.4 *(U-C Datalog program) An U-C Datalog program (or database) $DB = IDB \cup EDB$ consists of the extensional database EDB and of the intensional database IDB. We will always consider databases which are safe through query invocation.*

Example 3.1 *Consider the Example 2.4, then the constant, predicate and variable symbols are:*

- $\Sigma = \{\mathtt{9}, \mathtt{2}, \mathtt{6}, \mathtt{1}, \mathtt{f0}, \mathtt{f2}, \mathtt{f3}, \mathtt{f4}, \mathtt{f6}, \mathtt{shoe}\}$
- $\Pi^e = \{\mathtt{Dept}, \mathtt{Sale}\}$

[2] For the sake of simplicity we will denote from now on both conjunction and disjunction by ","

- $\Pi^{i} = \{\texttt{Movedept}, \texttt{Hsaledept}\}$
- $\Pi^{u} = \{+\texttt{Dept}, -\texttt{Dept}, +\texttt{Sale}, -\texttt{Sale}\}$
- $\Pi^{c} = \{\neg\texttt{Dept}^{*}, \neg\texttt{Sale}^{*}\}$
- $\Pi^{b} = \{=\}$
- $V^{i} = \{\texttt{Deptno}, \ldots\}$
- $V^{c} = \{\texttt{X}, \texttt{Y}\}$

Example 3.2 *Consider*

- $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}\}$
- $\Pi^{c} = \{\texttt{p}, \texttt{q}\}$
- $\Pi^{i} = \{\texttt{k}, \texttt{s}\}$
- $\Pi^{u} = \{+\texttt{p}, -\texttt{p}, +\texttt{q}, -\texttt{q}\}$
- $\Pi^{c} = \{\neg\texttt{p}^{*}, \neg\texttt{q}^{*}\}$
- $\Pi^{b} = \{=\}$
- $V^{i} = \{\texttt{X}, \texttt{Y}, \ldots\}$
- $V^{c} = \{\texttt{A}, \texttt{B}\}$

The query

$$?\ (\neg\texttt{q}^{*}(\texttt{a}), \neg\texttt{q}^{*}(\texttt{b})), +\texttt{q}(\texttt{Y}), \texttt{s}(\texttt{X}, \texttt{Y})$$

evaluated with respect to the database

$$\begin{array}{l}
\texttt{s}(\texttt{X},\texttt{Y}) \leftarrow +\texttt{q}(\texttt{X}), \texttt{k}(\texttt{X},\texttt{Y}). \\
\texttt{k}(\texttt{X},\texttt{Y}) \leftarrow -\texttt{p}(\texttt{X}), \texttt{p}(\texttt{X}), \texttt{q}(\texttt{Y}). \\
\texttt{p}(\texttt{a}) \\
\texttt{q}(\texttt{b})
\end{array}$$

computes during the marking phase:

- *the bindings* $\texttt{X} = \texttt{a}, \texttt{Y} = \texttt{b}$,
- *the updates* $+\texttt{q}(\texttt{a}), +\texttt{q}(\texttt{b}), -\texttt{p}(\texttt{a})$ *and*
- *the constraint* $\leftarrow \texttt{q}(\texttt{a}), \texttt{q}(\texttt{b})$.

The updates violate the constraint and the query fails. ◇

From Example 3.2 it is clear that the constraints can be very useful for semantic transaction optimization. Indeed inconsistent transactions leading to aborts can be detected during the marking phase, and therefore they can be efficiently founded. During the marking phase several condition can be checked in order to determine potential or secure aborts. This conditions can be computed considering the intensional database and optionally the transaction.

4 Semantics

In this section we define the semantics of U-C Datalog. We recall that the marking phase is just a special case of the semantics of CLP while the update phase is defined using the result of the marking phase and a semantic function.

4.1 Marking phase semantics

In the following we recall only the necessary results specializing them for U-C Datalog to model the marking phase. We denote a successful derivation of a goal G which yield the answer A, by $G \longrightarrow A$. $\mathcal{B}$ denotes the Herbrand base for CLP [11]. As answer of the marking phase we consider the bindings, the collected updates and constraints i.e., $A = \tilde{b}, \tilde{u}, \tilde{c}$.

Definition 4.1 *Let DB be an U-C Datalog program. The operational semantics $\mathcal{O}(P)$ is defined as follows*

$$\mathcal{O}(DB) = \{\ p(\tilde{X}) \leftarrow A \in \mathcal{B}\ \mid true, p(\tilde{X}) \longrightarrow A\ \}$$

Let us define the fixpoint semantics.

Definition 4.2 *Let DB be an U-C Datalog program and let $I \subset \mathcal{B}$ be an interpretation, then*

$$\begin{array}{ll}
T_{DB}(I) = \{ & p(\tilde{X}) \leftarrow \tilde{b}, \tilde{u}, \tilde{c} \mid \\
& \exists \text{ a renamed clause} \\
& p(\tilde{t}) \leftarrow \tilde{b}_0, \tilde{u}_0, \tilde{c}_0, p_1(\tilde{t}_1), \ldots, p_n(\tilde{t}_n) \text{ in } DB \\
& \exists\ p_i(\tilde{X}_i) \leftarrow \tilde{b}_i, \tilde{u}_i, \tilde{c}_i \in I, 1 \leq i \leq n \\
& \text{which share no variables} \\
& \tilde{b} = \tilde{b}_0 \cup \{\tilde{X}_1 = \tilde{t}_1\} \cup \ldots \cup \{\tilde{X}_n = \tilde{t}_n\} \cup \\
& \{b_1, \ldots, b_n\} \cup \{\tilde{X} = \tilde{t}\} \\
& \tilde{u} = \tilde{u}_0 \cup \{\tilde{u}_1, \ldots, \tilde{u}_n\} \\
& \tilde{c} = \tilde{c}_0 \cup \{\tilde{c}_1, \ldots, \tilde{c}_n\} \\
& \tilde{b}, \tilde{u}, \tilde{c} \text{ is solvable } \}.
\end{array}$$

Definition 4.3 *Let DB be an U-C Datalog program. The fixpoint semantics $Fix(DB)$ of DB is defined as $Fix(DB) = T_{DB} \uparrow n$*

Theorem 4.1 states the expected result on the relationship between the operational and the fixpoint semantics. This theorem allow us to compute the marking phase in a top-down style or equivalently in a bottom-up style.

Theorem 4.1 *(Equivalence of the operational and the fixpoint semantics) Let DB be an U-C Datalog program. Then $\mathcal{O}(DB) = Fix(DB)$.*

4.2 Update phase semantics

The semantics of the marking phase does not include the execution of the collected updates neither consider the transactional behavior of a query. In order to model those features we define the semantics of a query T [3] with respect to a database $IDB \cup EDB$. First we note that database systems use as default a set-oriented semantics, that is, the query-answering process computes a set of answers. Therefore we denote with

$$Set(T, IDB \cup EDB) = \{\langle \tilde{b}_i, \tilde{u}_i, \tilde{c}_i \rangle \mid T \longmapsto \langle \tilde{b}_i, \tilde{u}_i, \tilde{c}_i \rangle\ \}$$

the set of (bindings, updates and constraints) computed as the answers of the query T. Such answers can be computed by means of the operational semantics (see Definition 4.1) in a top-down style and according to Theorem 4.1 they can

[3] We denote a query with T to emphasize its transactional behavior.

be computed by means of the equivalent fixpoint semantics (see Definition 4.3) in a bottom-up style. Therefore we achieve the independency from the computational model. In addition, we define a function which takes a set of ground updates, the current extensional database and returns the new extensional database.

Definition 4.4 *Let EDB_i be the current database and u is the consistent set of ground updates. Then the new database EDB_{i+1} is computed by means of the function $\Delta : 2^{B^e} \times 2^U \rightarrow 2^{B^e}$ as follows:*

$$\Delta(EDB_i, u) = \begin{cases} EDB_i \setminus \{p(\tilde{t}) \mid -p(\tilde{t})\} \cup \\ \{p(\tilde{t}') \mid +p(\tilde{t}')\} & \text{if } YES \\ EDB_i & \text{otherwise} \end{cases}$$

where $\hat{c}$ denotes the union of all the collected constraints. 2^{B^e} denotes the set of possible databases, 2^U denotes the set of possible updates and the condition YES denotes $EDB_i \vdash \hat{c} \wedge \Delta(EDB_i, u) \vdash \hat{c}$

We briefly recall that we are interested on modeling as observable property of a transaction *the set of answers*, *the database state* and *the result of the transaction* itself. The observable properties is called $Oss_i = \langle Ans, State, Res \rangle$ where Ans is the set of answers to a query, $State$ is the database state, and Res is the transactional result, that is, either Commit or Abort.

The set of possible observables Oss_i is OSS. In the following we define the semantics of a transaction T with respect to the intensional database IDB as a function from observables to observables.

Definition 4.5 *(Query as transaction) Let $DB_i = IDB \cup EDB_i$ be the database. The semantics of a transaction is denoted by the function $\mathcal{S}_{IDB}(T) : OSS \longrightarrow OSS$. If a transaction T is a query $\tilde{b}, \tilde{u}, \tilde{c}, \tilde{G}$, then*

$$\mathcal{S}_{IDB}(T)(Oss_i) = \begin{cases} Oss_{i+1} & \text{if } OK \\ \langle \emptyset, Oss_i.2, Abort \rangle & \text{otherwise} \end{cases}$$

where $Oss_{i+1} = \langle \tilde{b}, EDB_{i+1}, Commit \rangle$. The set of bindings $\tilde{b}$ is $\{\tilde{b}_j \mid \langle \tilde{b}_j, \tilde{u}_j, \tilde{c}_j \rangle \in Set(T, DB_i)\}$ and EDB_{i+1} is computed by means of $\Delta(EDB_i, \bar{u})$. The condition OK expresses the fact that the set $\bar{u} = \bigcup_j \tilde{u}_j \tilde{b}_j$ is consistent and $\hat{c}$ satisfies EDB_i and $\Delta(EDB_i, u)$. By consistency we means that there are no complementary ground updates. $\tilde{u}_j \tilde{b}_j$ denotes the ground updates obtained by substituting the variables in $\tilde{u}_j$ with the ground terms associated with the variables in $\tilde{b}_j$. $Oss_i.n$ denotes the n-th component of the tuple Oss_i.

5 Conclusions

In this paper we have presented a new approach to integrate queries, updates, constraints and transactions in a logic-based language. This approach is based on a non-immediate semantics, where updates and constraints are expressed in rule bodies.

The basic idea of our approach is to consider a two steps computation. In the first step the updates and constraints are collected and in the second step the updates are executed only if the constraints are satisfied. The first step can be computed with top-down or bottom-up methods. In this way, forward and backward reasoning are reconciliated.

In our view, the main advantage of our approach is to allow permanent and temporary constraints. That is we consider updates to constraints. Expensive runtime rollback is not required for transactions due to the two phases computation. A two steps semantics for U-C Datalog language is provided. One the ongoing research directions is related to semantics transaction optimization based on abstract analysis of transactions.

References

[1] F. Bancilhon and R. Ramakrishnan. Performance Evaluation of Data Intensive Logic Programs. In J. Minker, editor, *Foundation of Deductive Databases and Logic Programming*, pages 439–519. Morgan-Kaufmann, 1987.

[2] E. Bertino, B. Catania, G. Guerrini, M. Martelli, and D. Montesi. Formalizzazione e ottimizzazione di transazioni di modifica in CLP(AD). In D. Saccà, editor, *Proc. Eighth Italian Conference on Logic Programming*, pages 501–516, 1993.

[3] E. Bertino, M. Martelli, and D. Montesi. An Incremental Semantics for CLP(AD). In A. Marchetti, Spaccamela, P. Mentrasti, and M. Venturini Zilli, editors, *Proc. Fourth Italian Conference on Theoretical Computer Science*, pages 53–67. World Scientific, 1992.

[4] E. Bertino, M. Martelli, and D. Montesi. Modeling Database Updates with Constraint Logic Programming. In U. W. Lipeck and B. Thalheim, editors, *Proc. Fourth Int'l Work. on Foundations of Models and Languages for Data and Objects*, pages 120–132, 1992.

[5] F. Bry. Intensional Updates: Abduction via Deduction. In D.H.D. Warren and P. Szeredi, editors, *Proc. Seventh Int'l Conf. on Logic Programming*, pages 562–575. The MIT Press, Cambridge, Mass., 1990.

[6] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer-Verlag, Berlin, 1990.

[7] U. S. Chakravathy, J. Grant, and J. Minker. Logic-Based Approach to Semantic Query Optimization. *ACM Transaction on Database Systems*, 15(2):162–207, June 1990.

[8] J. Jaffar et al. The CLP(R) Language and System. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, July 1992.

[9] U. Dayal et al. The HiPAC project: Combining active databases and timing constraints. *SIGMOD Record*, 17(1):51–70, March 1988.

[10] H. Gallaire, J. Minker, and J. M. Nicolas. Logic and database: A deductive approach. *ACM Computing Surveys*, 16(2):153–185, June 1984.

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

[12] C. Mancheville and E. Simon. Modelling Non Deterministic Queries and Updates In Deductive Databases. In F. Bancilhon and D. J. DeWitt, editors, *Proc. Fourteenth Int'l Conf. on Very Large Data Bases*, pages 395–406, 1988.

[13] S. Manchanda and D. S. Warren. A Logic-based Language for Database Updates. In J. Minker, editor, *Foundation of Deductive Databases and Logic Programming*, pages 363–394. Morgan-Kaufmann, 1987.

[14] D. Montesi. *A Model for Updates and Transactions in Deductive Databases*. PhD thesis, Dipartimento di Informatica, Università di Pisa, March 1993.

[15] K. Morris, J. D. Ullman, and A. V. Gelder. Design Overview of the NAIL! System. In E. Y. Shapiro, editor, *Proc. Third Int'l Conf. on Logic Programming*, volume 225 of *Lecture Notes in Computer Science*, pages 554–568. Springer-Verlag, Berlin, 1986.

[16] S. Naqvi and S. Tsur. *A Logic Language for Data and Knowledge Bases*. Computer Science Press, 1989.

[17] J-M. Nicolas. Logic for Improving Checking in Relational Data Bases. *Acta Informatica*, 18(3) 227–253, 1982. Springer-Verlag, Berlin.

[18] J. D. Ullman. *Database and Knowledge-Base Systems*. Computer Science Press, 1989.

[19] J. Widom and S. J. Finkelstein. Set-Oriented Production Rule in Relational Databases Systems. In H. Garcia-Molina and H. V. Jagadish, editors, *Proc. Int'l Conf. ACM on Management of Data*, pages 259–270, 1990.