Static Analysis of Transactional Intensional Databases

Extended Abstract

E. Bertino[1], B. Catania[1], G. Guerrini[2], D. Montesi[3]
[1]Dipartimento di Scienze dell'Informazione, Università di Milano
Via Comelico, 39 20133 Milano, Italy
{bertino,catania}@disi.unige.it
[2]Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova
Viale Benedetto XV, 3 16132 Genova, Italy
guerrini@disi.unige.it
[3]Informatics Department, Systems Engineering Division
Rutherford Appleton Laboratory, SERC
Chilton, Didcot, Oxon OX11 0QX, UK
danilo@inf.rl.ac.uk

Abstract

This paper proposes a formal framework to statically analyze intensional databases modeling transactional behavior. The considered language is U-Datalog, an extension of Datalog modeling updates and transactions. The analysis, performed at compile time, provides information about certain types of situations that may arise at execution time such as aborts, inconsistencies and failures. These information can be used to check the correctness of the database and to optimize transactions saving useless computation. The paper proposes a formal model and an algorithm for the analysis.

1 Introduction

Optimization of queries is a primary issue of research in database area. Most of the current works focuses on query optimization [19, 4, 9] in relational and deductive databases, due to a clear query model in these contexts [18, 15]. Several approaches to optimization have been proposed. It is possible to make ad-hoc optimization [9], consider semantic constraints [6, 10] or implementation issues [17]. These approaches to optimization are not alternative but integrative. For instance, ad-hoc optimization based on special physical data structures can be done altogether with optimization based on semantic constraints. They work at different levels (i.e., physical and logical respectively) to achieve the same task, that is saving useless computation. Roughly speaking, the difference between the types of approaches to optimization is related to the use of different types of information. For instance, physical optimization uses information to reorganize/build new or more sophisticated data structures. Semantic optimization uses information about constraints to transform the database into a new more efficient one. However, if query optimization can save useless computation, further savings can be obtained with transaction optimization. Transactions are execution units consisting of several updates, verifying the atomicity property. Transaction optimization has not been applied to Datalog databases but only to relational databases [2]. This is related to the difficulty of integrating updates with declarative rule based query languages and therefore of encapsulating them into transactions.

Solutions proposed so far to model updates in the Datalog language (see [1] for a survey) do not satisfactorily handle updates. The main drawback of those proposals is that updates are executed as soon as rules containing them (either in the left or right hand sides) are evaluated. This approach to update processing leads to complex semantics, that have made difficult the smooth integration of update capabilities in rule languages. Therefore, in many cases, updates are handled by an ad-hoc language, different from the language used for describing rules, making transaction optimization difficult [13]. To overcome these drawbacks we use a new approach to integrate declarative query language and updates, modeling transactions. The language we use is U-Datalog. The basic idea of U-Datalog [5, 20] is that (extensional) updates are not executed as soon as they are evaluated. Rather, they are collected and are executed only at the end of the query-answering process. Updates, due to the non-immediate semantics, are expressed in a declarative way by using constraints, that is $+p(a)$ states that in the new state $p(a)$ must be true. Those constraints form an (hypothetical) constraint theory which can be inconsistent. If the set is not consistent the transaction is *aborted*, that is, no update in the set is performed. The notion of consistency is an important one, in that it prevents a set of updates containing both an insertion and a deletion of the same information to be executed. For instance, $+p(a), -p(a)$ represents the constraint theory $p(a), \neg p(a)$ which is inconsistent for any transaction and for any databases state. Updates are expressed in the rules of the intensional database. Thus the set of updates performed by a transaction, issued by a user, includes all the updates gathered during the transaction execution. This is the major difference with relational databases where updates are explicitly specified in the transaction[1].

The implicit representation of updates makes difficult to statically analyze transaction execution if compared with the relational case. With static analysis we mean an analysis considering only the intensional database, that is the time invariant component of the database, without regarding the user transaction and the database state. A static analysis of intensional databases modeling transactions is useful both to check the correctness of the intensional database and to reduce the cost of the run-time computations. In fact, it allows to collect compile-time information on the intensional database. The retrieved information can be used to identify anomalous situations related to the definition of predicates. For example, the analysis may discover that a predicate always (that is for each extensional predicates) does not generate any solution. After the compilation, this information can be returned to the user that can decide to modify the definition of such predicate to remove the error. The same approach can be used as a semantic transaction optimization[2]. For example, in order to avoid useless computations, it is important to analyze the intensional database to determine whether rules exist that may generate inconsistent updates. In this case, an abort condition may arise and the result of a transaction is known without executing it.

In this paper we propose a new approach to statically analyze U-Datalog intensional databases. The approach we propose is new with respect to previous works. Indeed, because of the difficulties in modeling updates in rule databases, to the best of our knowledge no attempts exist aiming at identifying potential errors that may arise and to define how to detect

[1]Note that this difference is also present in active databases.

[2]The semantic information concerns the meaning of the analyzed program.

them. Past research has mainly focused on query optimization. For example, the approach proposed in [21] aims to analyze databases to detect rules and predicates that will never be satisfiable. Those rules and predicates can be eliminated from the intensional database. When dealing with rules with updates, they may contain rules that generate inconsistent updates, for any database state. Therefore, such rules should be eliminated from the intensional database. In other situations, the inconsistencies may be potential, since they depend on the specific database state. In both cases, it is important to be able to properly characterize potential or certain inconsistency, by providing information about the involved rules.

The main contributions of this paper can be summarized as follows: (1) identification of the kinds of errors that can arise because of updates; (2) definition of a formal framework to characterize the rules of an intensional database with respect to these kinds of errors - this characterization is provided as a set of annotations on the rules; (3) definition of an algorithm to generate the annotations for a given U-Datalog intensional database.

The paper is organized as follows. Section 2 presents an overview of U-Datalog. Section 3 first introduces the various types of errors we are interested in, then presents a formal model for the analysis. Section 4 presents a simple algorithm to generate correct annotations. Finally, Section 5 outlines some conclusions and presents future work.

2 Overview of U-Datalog

A Datalog program [9] consists of a set of *base relations* (EDB) and a set of *rules* (IDB). Many extensions to Datalog have been proposed to express updates. In the following we summarize a new approach based on a non-immediate update semantics.

Update-Datalog (U-Datalog) is a rule language which allows declarative specification of updates in Datalog rules [9]. The main motivations to consider updates in the body of the rules are: (i) The approach with updates in the head does not allow the transmission of runtime parameters (for such languages only a bottom-up computational model is allowed); (ii) Updates with non-immediate semantics in rule body result in a language which fits nicely into the formal setting of constraint logic programming [5] and for which both a bottom-up and a top-down computational models, proved equivalent, can be defined.

The execution model of U-Datalog consists of two phases, the marking phase and the update phase. The first phase collects the updates found during the evaluation process, without, however, executing them. During the update phase they are executed altogether only if they are consistent. If the set is not consistent the transaction is *aborted* and no update in the set is performed. The notion of consistency is an important one, in that it prevents a set of updates containing both an insertion and a deletion of the same fact to be executed. If an abort occurs, none of the generated updates is executed, following the database principle by which if an abort occurs the state must not be changed. By contrast in DLP and $\mathcal{LDL}$, updates are executed as soon as they are evaluated, that is, they are executed as side effect of the derivation process. For these languages a bottom-up interpretation of rules with updates has no meaning. By contrast, the two phases semantics given to U-Datalog removes this drawback. For it, both a top-down and a bottom-up computational model can be defined.

In the following we recall the language and its informal semantics (see [20] for a complete description). In that, we will assume the reader familiar with the basic notions of logic programming [3].

We consider pairwise disjoint sets of *constants* Σ, *predicate names* Π and *variables* V. Predicates have a certain arity. A *term* is either a constant or a variable. The *atoms* are expressions of the form $p(t_1, ..., t_n)$, where the $t_i, 1 \leq i \leq n$, are terms and p is a predicate name. $\tilde{t}$ denotes the tuple $t_1, ..., t_n$. *Update atoms* are extensional atoms prefixed with $+$ or $-$ to denote respectively insertions or deletions. The extensional atoms are those built starting from the extensional predicates (Π_{EDB}). Predicate symbols are composed by two disjoint sets: Π_{EDB}, and the IDB-*predicate symbols* (Π_{IDB}), occurring in the intensional database but not in the intensional one. A *rule* is an expression of the form $H \leftarrow U_1, \ldots, U_i, B_{i+1}, \ldots, B_n, n \geq 0$, where the *body* $U_1, \ldots, U_i, B_{i+1}, \ldots, B_n$ is a sequence of atoms and updates and the head H is an intensional atom. If $n = 0$, a rule is also called a *fact*. We require rules to be *safe*. Our notion of safety extends that in [22]. A rule is safe if each variable in the head and each variable in an update atom of the rule appears in a non update atom in the body. The intuitive meaning of a rule is: "if $B_{i+1}, \ldots, B_n$ is true, the updates $U_1, \ldots, U_i$ are consistent, then H is true". Note that we do not consider updates in rule heads. An *extensional* database (EDB) is a set of facts. An *intensional* database (IDB) is a finite set of rules. Atoms and updates which do not contain variables are called *grounds*. An *U-Datalog program* (or database) with updates $DB = IDB \cup EDB$ consists of the extensional database EDB and of the intensional database IDB. A *transaction* is a rule without body composed by only one non update atom. We require the arguments in the transaction be distinct variables[3]. A *complex transaction* T is a sequence of transactions $T_1; \ldots ; T_n$.

Example 1 *The following is a U-Datalog intensional database IDB:*

$p(X) \leftarrow -q(X), q(X).$ $\quad r(X) \leftarrow +t(X), p(X).$ $\quad s(X) \leftarrow t(X).$

$v(X) \leftarrow +q(X), s(X).$ $\quad w(X) \leftarrow +q(X), p(X).$ $\quad \square$

The semantics of an U-Datalog program is given in two steps. The first step semantics models the *marking phase*, gathering updates in a set[4]. The answers to the marking phase must be provided by means of a top-down and an equivalent bottom-up semantics. This step provides a set of pairs (bindings and updates) computed as answers of the query on the database. This set includes only *consistent* sets of updates, that are sets which don't contain complementary updates, i.e. updates respectively of the form $-p(\tilde{t})$ and $+p(\tilde{t})$ (the pairs with an inconsistent set of updates are not inserted in the set).

The second step provides the transactional behavior, modeling the *update phase*. The hypothetical updates computed by the marking phase are executed with a transactional mechanism, i.e. an *all or nothing* style. Due to the query-update features of the language, as *observable property*[5] of a transaction we consider *the set of answers, the database state* and *the result of the transaction* itself, which can be *Commit* o *Abort*. In this phase we consider the sets of updates collected by the marking phase, to which the related bindings have been applied. Let us consider the set obtained as the union of all the updates gathered by the different solutions, appropriately instantiated[6]. If the collected updates are consistent, the new database state is computed (by applying the insertions and the deletions in the set) and the transaction commits. If the set is inconsistent, the transaction aborts and no updates are performed, following the database principle by which if an abort occurs the state must not be changed. Therefore, a transaction T aborts whenever a set of solutions is found for it such

---

[3]Note that each rule without body can be translated in this form, modifying the intensional database. In this sense, also the intensional database may be seen as a transaction.

[4]The semantics of constraint logic programming gives the formal setting to model the marking phase of an U-Datalog transaction execution.

[5]An observable property is the property chosen to point out the behavior of a program. It must be fixed before defining the semantics of a program. Such concept has been widely used in the programming languages and concurrency areas.

[6]Note that the proposed semantics models a set-oriented transactional behavior.

that each solution is consistent, but a solution, itself consistent, contains one or more updates which are inconsistent with updates contained in other consistent solutions. Note that the result of the execution of an inconsistent set of updates depends on the order of the execution of each update. Therefore, the notion of consistency avoids the execution of order dependent sets of updates.

Example 2 *Consider* $EDB_i = \{q(b)\}$ *and the intensional database of Example 1. The user transaction* $T_1 = ?r(X)$ *evaluated in* $EDB_i \cup IDB$ *computes the binding* $X = b$ *and collects the updates* $-q(b), +t(b)$*. Note that such updates form a transaction that we call induced transaction. Informally the new extensional database* $EDB_{i+1} = \{t(b)\}$ *is the result of the application of these updates to* EDB_i*. The transaction* $T_2 = ?s(X)$ *evaluated in* $EDB_{i+1} \cup IDB$ *computes the binding* $X = b$ *and does not compute any update, thus the new extensional database is still* EDB_{i+1}*. The transaction* $T_3 = ?v(X)$ *evaluated in* $EDB_{i+1} \cup IDB$ *computes the binding* $X = b$ *and collect the update* $+q(b)$*, thus the new extensional database is* $EDB_{i+2} = \{t(b), q(b)\}$*. The transaction* $T_4 = ?w(X)$ *would return the binding* $X = b$*, and the updates* $+q(b), -q(b)$*. The collected update set is inconsistent, so the solution is not considered. Therefore* T_4 *does not return any solution, and it does not perform any update.* $\Box$

The language allows to use sequences of transactions. Here, we do not discuss their semantics. It is however useful to remind that the abort of a simple transaction in a sequence results in the abort of the entire sequence.

The two phase semantics induces an interesting equivalence between (complex) transactions. This is very important because we can transform a (complex) transaction into a semantically equivalent one which is computationally less expensive.

Finally note that even if the language is *active* in the sense that the extensional database may be changed by a transaction execution (each update is therefore an action), it cannot be considered *reactive*. This means that U-Datalog rules are not automatically activated by the generation of some events but only by an explicit user transaction execution. In this sense, U-Datalog rules are not *active rules* [16].

3 Formal model for a static analysis of intensional databases

The compilation of a program usually returns a set of additional information on it. These information are useful to identify errors or anomalous situations. For example, compiling a Prolog program an error arises if the syntax of some constructs in the program is wrong but only a *warning* arises if the program is correct but some dangerous situations, i.e. a free variables in the head or in the body, arise. These messages help the programmer to write correct and reliable code.

A similar error detection may be useful compiling an U-Datalog intensional database. Indeed, the capabilities of the language may generate error situations like abort, inconsistencies and failures. We can think of these situations as of errors taken over by the compilation of an intensional database. Of course, the detection of these errors must be done at compile time and must be independent from the current extensional database. The independence from the extensional database allows the development of a *time independent analysis*, that is an analysis whose results do not change when the extensional database changes.

The errors we want to identify may be certain, in this case they arise for each possible extensional database, or potential. In this case the compiler is not sure that the error arises because either it exists an extensional database for which the error does not arise or the analysis of the compiler is not enough precise. For example, considering the rule $q(X) \leftarrow +s(X), -s(Y), g(X,Y)$, the transaction $?q(X)$ may abort (for example if the extensional database contains the fact $g(a,a)$) but an analysis that is independent from the extensional database cannot generate this information, that therefore is only potential. Both potential errors and certain errors are useful to identify anomalous executions and therefore both should be considered by the analysis.

The identification of errors in the compilation phase of an intensional database can be useful also to optimize transactions. For example, if it is known that two rules defining a predicate p generate a certain abort condition, the result of transaction $?p(\tilde{X})$ is known without any execution (no bindings are generated and no updates are executed).

In order to statically analyze a U-Datalog intensional database, first of all, given an intensional database, a *normal form* is generated at compile time. A normal form of an intensional database is an intensional database equivalent to the original one but syntactically simpler. In this paper, if the IDB is not recursive, the considered normal form is the program obtained unfolding the original one. If IDB is recursive, we use a particular normal form [7] which is currently under definition. For the time being, we suppose to use as normal form the *finite semantics* of IDB [14]. The finite semantics of IDB is a set of rules equivalent to IDB having as body literals only extensional atoms and based on the hypothesis of the finiteness of the alphabet symbols. The normal form is therefore composed in any case by rules having only extensional predicates in the body.

Given the normal form of an intensional database, we statically identify some certain or potential errors related to the semantical behavior of this new intensional database; we maintain these information using a syntactical construct called *annotation*. For each predicate definition the compilation phase generates a set of annotations.

In the following, we first discuss the types of errors we are interested in characterizing. Then, we present a formal model for associating with a program a set of *annotations*. Finally, the main ideas of an algorithm generating annotations for an intensional database are introduced.

3.1 Errors

Because of the presence of updates in an intensional database, different kinds of errors can arise when a goal G is issued against the database. Possible errors are classified as inconsistencies, aborts, failures. Each type of error is discussed in the following.

Inconsistencies

An inconsistency arises if a solution for a given transaction T is found that, however, is inconsistent with respect to the updates that have been collected during its marking phase execution. We say that a rule r generates an inconsistency iff at least one solution obtained firing r is inconsistent. If firing rule r no inconsistencies arise, then T is said to be *inconsistency-free* with respect to r. Inconsistencies may also be certain or potential. In the first case, we are sure that one or more solutions obtained from a rules r for a transaction T will be inconsistent. In the second case, an inconsistent solution may exist, but this information cannot be statically detected in that it depends on the EDB.

Example 3 *Consider the following intensional database IDB:*

$R_1 : q(X) \leftarrow +p(X), s(X), r(X)$ $\quad$ $R_2 : q(X) \leftarrow -u(X), +f(Y), r(X), u(Y)$

$R_3 : t(X,Y) \leftarrow -f(X), +f(Y), g(Y), z(X)$ $\quad$ $R_4 : v(X) \leftarrow +u(X), -u(Y), s(X), s(Y), g(Y)$

$R_5 : z(X) \leftarrow k(X)$ $\quad$ $R_6 : z(X) \leftarrow z(Y), h(X,Y)$

The execution of the transaction $?q(X)$ *cannot generate any inconsistent solution, because the rules used in its execution do not contain complementary update atoms over the same predicate. Thus,* $?q(X)$ *is inconsistency-free with respect to* R_1 *and* R_2*. On the other hand, if* $g(c), k(a), h(a, c)$ *are in the current EDB (and thus* $z(c)$ *is derivable), then the execution of the transaction* $?t(X, Y)$ *using rule* R_3 *generates* $+f(c)$ *and* $-f(c)$*, that is an inconsistent solution. However, if no information about* g*,* k *and* h *are known, the inconsistency is only potential (for some EDB* R_3 *may generate some inconsistent solutions). Finally, we note that the execution of the transaction* $?v(X)$ *will certainly generate some inconsistent solutions, if at least one solution for this transaction exists. In fact in this case the bindings for variable* X *are always a superset of the bindings for variable* Y*. This means that at least one solution generated by this rule is inconsistent.* □

Aborts

Aborts may be, as inconsistencies, certain or potential. If no aborts (either certain or potential) can arise for a given transaction T, then T is said to be *abort-free*, that is, transaction T will commit.

Example 4 *Consider the following intensional database IDB:*

$R_1 : q(X) \leftarrow +p(X), s(X), t(X) \qquad R_2 : q(X) \leftarrow -p(X), +m(Y), r(X), t(Y)$

$R_3 : v(X) \leftarrow +g(X), t(X)$

We note that the update part of R_1 *contains the update atom* $+p(X)$*, while the update part of* R_2 *contains the update atom* $-p(X)$*. Thus, with respect to the state of the extensional predicates* s*,* t *and* r*, the transaction* $?q(X)$ *may abort. In particular, an abort arises if the predicates* s*,* t *and* r *are not disjoint. Finally, we note that the transaction* $?v(X)$ *never aborts, because it never generates an inconsistent update set. Thus, this transaction is abort-free.* □

Note that if a transaction T aborts in the context of a sequence of transactions, the entire transactions is aborted. Therefore, the execution of the transactions following T in the complex transaction is not started and all the updates performed by transactions preceding T are undone. The detection of abort situations, either certain or potential, is quite important to prevent expensive transaction roll-back operations.

Failures

It might happen that the use of a clause during the deduction of a transaction T does not generate any solution. Therefore, this clause is not relevant to the evaluation of T. By contrast, sometimes a rule allows to always (that is, for each possible extensional database) obtain at least one solution. In the first case, we say that the rule *fails*, in the second case we say that the rule *succeeds*.

Example 5 *Consider the following intensional database IDB:*

$R_1 : q(a) \leftarrow +m(a) \qquad R_2 : u(X) \leftarrow t(X), s(X) \qquad R_3 : v(X) \leftarrow +p(X), -p(X), t(X)$

Rule R_1 *certainly succeeds, independently from the considered EDB. In facts, firing this rule, the gathered update set (containing only* $+m(a)$*) is always ground and consistent. Therefore, rule* $R1$ *always generates a solution. Now consider the transaction* $?u(X)$*. Rule* $R2$ *may succeed or fail depending on the extension of predicates* t *and* s *in the considered EDB. In particular it succeeds if at least a common instance of* t *and* s *is present in the EDB. Now consider the transaction* $?v(X)$*. Rule* R_3 *never generates a solution. If no instances of* $t(X)$ *are present in the considered EDB, the rule generates no solution and the transaction fails. However, if at least an instance of* $t(X)$ *is contained in the current the EDB, the rule generates only inconsistent solutions (due to the presence of the complementary updates* $+p(X), -p(X)$ *in the rule body) that are not acceptable and again the transaction fails. So we have a certain failure situation.* □

Note that the detection of failing rules should be considered as an extension of the problem of finding unsatisfiable Datalog rules [21] to a language modeling updates. In this case we consider unsatisfiability due to the inconsistency of a set of updates.

3.2 A formal model for annotations

In this section, we formalize the concept of *annotation*, as a way to maintain information about semantic properties of the intensional predicates. Note that according to our definition of transaction, a predicate definition corresponds to a particular transaction. Therefore the properties we determine are related to all the possible definable transactions for a given IDB.

An annotation for an intensional database IDB contains information on errors related to the definition of the intensional predicates. An annotation for IDB is then a set of simple annotations, each of them related to an intensional predicate. Each simple annotation points out: what is the considered error and what is the state of the error (the error always arises; the error never arises; the error may arise); which is the predicate to which the error is related; which are the rules or update atoms that determine the error. Simple annotations can be therefore modeled as tuples containing three elements.

In order to formalize this notion, we need a simple way to refer to rules and update atoms of a program. We suppose to number all the rules in the program and all the update atoms contained in these rules. For the sake of simplicity, we denote with n_r the number related to rule r and with n_u the number associated with update atom u. As a second step, we need to identify which elements, rules, update atoms or pairs of update atoms, determine the error. For example, if we want to model an Abort/Commit property we need information on pairs of update atoms in the same rule or in different rules. For the inconsistency error, we need information about pairs of update atoms contained in the same rule. For the failure error, we need information about rules. The concept of *information* formalizes this notion.

Definition 1 An information i for the intensional database IDB is an element of the set $\{S \mid S \in 2^{\mathcal{N}}, card(S) \leq 2\}$, where $\mathcal{N}$ denotes the set of natural numbers.
If $i = \{n_{a_1}, n_{a_2}\}$ we call this information *update information* (upi) and we require that one of the following conditions holds:

- there exist two update atoms a_1 and a_2 contained in the same rule in IDB; we call this information *homogeneous update information* (houpi)
- there exist two update atoms a_1 and a_2 contained in different rules in IDB; we call this information *heterogeneous update information* (heupi)

If $i = \{n_r\}$, we call this information *unit information* (uni) and we require that there exists a rule $r \in IDB$.
If $i = \{\}$, we call i *null information* (ni) and we do not require anything else.
We denote with $INFO_{IDB}$ the set of all the definable information for an intensional database IDB. $\Box$

In order to denote the state of an error, we introduce the concept of *property*.

Definition 2 $PROP$ is the set $\{CM, AB, PA, IN, PI, WIN, SC, PF, FL\}$, where each symbol has the following meaning: CM commit, AB abort, PA potential abort, IN inconsistency, PI potential inconsistency, WIN without inconsistency (inconsistency free), SC success, FL failure, PF potential failure. Each $prop \in PROP$ is called *property* and it identifies a particular state for a particular error. $\Box$

A property is certain if the corresponding error will arise in every possible EDB. By contrast, the error corresponding to a potential property will not necessarily arise in every EDB. Note that, for each kind of error, we need three identifiers, in order to denote the following three facts: the error will always arise, the error may possibly arise, the error will never arise. Properties CM, AB, PA are related to abort, WIN, IN, PI are related to inconsistencies and SC, FL, PF are related to failure conditions. We call CM, WIN, SC *positive certain properties*, AB, IN, FL *negative certain properties* and PA, PI, PF *potential properties*.

Now, we are able to relate predicates to information and properties, defining the concept of *simple annotation*.

Definition 3 Let IDB be an intensional database, $p \in \Pi_{IDB}$. A *simple annotation* for p, denoted as $\mathcal{C}(p)$, is defined as $\mathcal{C}(p) = \langle p, i, prop \rangle$ where $i \in INFO_{IDB}$ and refers to rules and updates defining predicate p whereas $prop \in PROP$ and

- if i is a (ni) then $prop \in \{CM, AB\}$
- if i is a (uni) then $prop \in \{SC, PF, FL, WIN\}$
- if i is a (houpi) then $prop \in \{IN, PI, PA\}$
- if i is a (heupi) then $prop \in \{PA\}$ □

The meaning of each simple annotation is straightforward. For example $\mathcal{C}(p) = \langle p, -, AB \rangle$ means that the transaction $?p(\tilde{X})$ ABorts by the generation of an inconsistent set of updates; $\mathcal{C}(p) = \langle p, \{n_u, n_m\}, PA \rangle$ means that the transaction $?p(\tilde{X})$ may Potentially Abort by the generation of a set of an inconsistent set of updates generated. The inconsistent updates may be generated by the update atoms u and m.

Example 6 *Consider the IDB of Example 3. The numbers near the rules and the update atoms identify the values returned by the enumeration function.*

$1 : q(X) \leftarrow 1 : +p(X), s(X), r(X)$
$2 : q(X) \leftarrow 2 : -u(X), 3 : +f(Y), r(X), u(Y)$
$3 : t(X, Y) \leftarrow 4 : -f(X), 5 : +f(Y), g(Y), z(X)$
$4 : v(X) \leftarrow 6 : +u(X), 7 : -u(Y), s(X), s(Y), g(Y)$
$5 : z(X) \leftarrow k(X)$
$6 : z(X) \leftarrow z(Y), h(X, Y)$

Consider rules 1 *and* 2. *None of them can generate inconsistent solutions, this is represented by the annotations $\langle q, \{1\}, WIN \rangle$ and $\langle q, \{2\}, WIN \rangle$. Now consider rule* 3. *Its potential inconsistencies are represented by the annotation $\langle t, \{4, 5\}, PI \rangle$. This annotation points out that rule* 3 *may generate some inconsistent solutions for the transaction $?t(X, Y)$, because of the update atoms numbered respectively* 4 *and* 5, *but this condition is not certain. Indeed, it exists an extensional database for which rule* 3 *does not generate any inconsistent solution (iff the set of extensional facts for the predicate g and the set of instances deduced for z are disjoint). Consider now rule* 4. *Its inconsistencies are represented by the annotation: $\langle v, \{6, 7\}, IN \rangle$. This annotation points out that rule* 4 *generates some inconsistent solutions for the transaction $?t(X, Y)$, whenever a solution exists, because of the update atoms numbered respectively* 6 *and* 7. *This condition is certain, because the set of facts inserted by the update $+u(X)$ and the set of facts deleted by the update $-u(Y)$ are never disjoint, unless these sets are empty. Moreover, the rules of IDB cannot produce abort conditions. This information is represented by the annotations $\langle q, -, CM \rangle$, $\langle t, -, CM \rangle$, $\langle v, -, CM \rangle$, $\langle z, -, CM \rangle$.*
Now, consider the IDB of Example 4.

$1 : q(X) \leftarrow 1 : +p(X), s(X), t(X) \quad 2 : q(X) \leftarrow 2 : -p(X), 3 : +m(Y), r(X), t(Y)$
$3 : v(X) \leftarrow 4 : +g(X), t(X)$

The potential abort of the transaction $?q(X)$, for inconsistency of the update set generated, is represented by the annotation $\langle q, \{1,2\}, PA\rangle$. This annotation points out that the transaction $?q(X)$ may abort, because of the update atoms identified by the values 1 and 2. Indeed, these update atoms can generate some inconsistent updates. By contrast, rule 3 never generates abort conditions. This information is represented by the property $\langle v, -, CM\rangle$.
Finally, consider the IDB of Example 5.

$1 : q(a) \leftarrow 1 : +m(a) \qquad 2 : u(X) \leftarrow t(X), s(X) \qquad 3 : v(X) \leftarrow 2 : +p(X), -p(X), t(X)$

As seen, rule 1 always succeeds. This situation is represented by the annotation $\langle q, \{1\}, SC\rangle$. By contrast, rule 2 may succeed or fail, depending on the considered EDB. This situation is represented by the annotation $\langle u, \{2\}, PF\rangle$, which denotes the potential failure of the rule. Finally, rule 3 is known to produce no useful solution. This situation is represented by the annotation $\langle v, \{3\}, FL\rangle$, which denotes the certain failure of the rule. □

An important property for a simple annotation is the *correctness*. Informally, a simple annotation for a predicate p is correct if the simple annotation corresponds to a real error of the predicate. Considering the IDB of Example 4, the simple annotation $\langle q, \{1\}, SC\rangle$ is not correct, because if in the EDB the set of facts for s and t are disjoint no solution is generated using rule R_1. The following is the formal definition of correctness.

Definition 4 Given an intensional predicate p and an intensional database IDB a characterization $\mathcal{C}(p)$ is *correct* iff the property is negative and certain and the error identified by $\mathcal{C}(p)$ actually occurs during the deduction of the transaction $?p(\tilde{X})$ in $IDB \cup EDB$ for any possible EDB, or the property is positive and certain and the error identified by $\mathcal{C}(p)$ does not occur during the deduction of the transaction $?p(\tilde{X})$ in $IDB \cup EDB$ for any possible EDB, or the property is only potential and the error may occur during the deduction of the transaction $?p(\tilde{X})$, however at least one possible extensional database EDB may exist for which the identified error does not occur, during the deduction of the transaction $?p(\tilde{X})$ in $IDB \cup EDB$. □

Note that a given predicate may have several correct simple annotations associated with it. The set of all the correct simple annotations for all the intensional predicates of an intensional database IDB defines an *annotation* for IDB.

Definition 5 Given an intensional database IDB, the annotation of IDB, denoted as $\mathcal{IC}(IDB)$ is defined as $\bigcup_{p \in \Pi_{IDB}} \{\mathcal{C}(p) \mid \mathcal{C}(p)$ is a correct simple annotation for $p\}$. □

An intensional database IDB may have different correct annotations. An important question is therefore which are the best ones. For this purpose, an order can be introduced on all the possible correct annotations for a given intensional database. The definition of such order is presented in Appendix B.

4 A simple algorithm for the generation of a correct annotation

In this section an algorithm to generate a correct annotation for *non recursive* normal forms of intensional databases, having only extensional predicates in rule bodies, is proposed. The proposed algorithm is static, that is, it can be applied without any run-time information and it is therefore independent from the current extensional database, that may change at run-time. For each of the possible error - abort, inconsistency, failure - the algorithm generates a number of correct annotations for each intensional predicate. The set of the generated annotations can be maintained together with the intensional database and may be used for testing the correctness of the intensional database and to optimize transactions.

In order to determine when some update atoms generate abort or inconsistencies conditions the algorithm uses the notion of *covering condition*. Informally, two update atoms satisfy the covering condition if the set of facts updated by one of them is contained in the set of facts updated by the other one. Consider the following example.

Example 7 *Consider the following intensional database:*
$R_1 : q(X) \leftarrow +p(X), s(X), r(X) \quad R_2 : q(X) \leftarrow -p(X), +v(Y), r(X)$
We note that the update atom $-p(X)$ in R_1 covers the update atom $+p(X)$ in R_2. In fact, the set of updates generated by the update atom $+p(X)$ is contained in the set of updates generated by the update atom $-p(X)$. This is due to the fact that the facts inserted by $+p(X)$ are those for which $s(X)$ and $r(X)$ are in the current EDB, while the facts deleted by $-p(X)$, if any, are those for which $r(X)$ is in the current EDB. This fact allows to identify an abort condition.
□

Let A be an atom. In the following, $\overline{A}$ denotes the atom obtained from A replacing each argument with a distinct variable. The following definition formalizes the previous concept.

Definition 6 Let $r_i = H_i \leftarrow u_1^i, ..., u_n^i, \alpha p(\tilde{t}), B_1^i, ..., B_t^i$, $r_j = H_j \leftarrow b_1^j, ..., b_m^j, \beta p(\tilde{u}), B_1^j, ..., B_q^j$, where u_k^i and b_h^j are update atoms, $k = 1, ..., n$, $h = 1, ..., m$ and B_p^i and B_f^j are deduction atoms, $p = 1, ..., t$, $f = 1, ..., q$. We say that $\alpha p(\tilde{t})$ *covers* $\beta p(\tilde{u})$, denoted as $n_{\alpha p(\tilde{t})} \rhd n_{\beta p(\tilde{u})}$, iff the set of facts inserted (deleted) by $\beta p(\tilde{u})$ is contained in the set of facts inserted (deleted) by $\alpha p(\tilde{t})$, during the execution of transactions $\overline{H_i}$ and $\overline{H_j}$. □

In order to use the previous condition in the algorithm we need a static condition to establish when the covering condition is verified. This problem is related to the query containment problem [22]. The covering condition can be rewritten as:

the set of answers for transaction $?\overline{p(\tilde{t})}$ where
$p(\tilde{t}) \leftarrow u_1^i, ..., u_n^i, \alpha p(\tilde{t}), B_1^i, ..., B_t^i$
contains in the set of answers for transaction $?\overline{p(\tilde{u})}$ where
$p(\tilde{u}) \leftarrow u_1^j, ..., u_n^j, \beta p(\tilde{u}), B_1^j, ..., B_t^j$.

The satisfiability of the covering condition between two update atoms is therefore related to the satisfiability of a containment condition between two conjunctive queries. However, in this case we have to consider the presence of update atoms, that may determine failure conditions for inconsistency. It is known that the complexity of testing the containment of conjunctive queries is exponential. However, note that the analysis can be performed only once at compile time. Therefore, in any case, no overhead for the run-time system is generated by the analysis. Here, we do not further discuss this point but we give the complete algorithm considering the test for the covering condition as a black box.

The algorithm generates an annotation for an intensional database, that is a set of correct simple annotations for each intensional predicates. It is composed by three steps. In the first step, a set of correct simple annotations is constructed from a given set of rules, such that if the premises of a rule are true the consequence is a correct simple annotation for the considered intensional database. The rules in this step have not correct simple annotations in the premises. The properties of the simple annotations generated in the first step are related to certain abort, certain inconsistencies and success properties (AB, IN, SC). Let $\mathcal{A}1$ be the set generated by these rules. In the second step some other simple annotations are generated. The rules in this set may have correct annotations contained in the set $\mathcal{A}1$ in the premises.

The properties of the simple annotations generated in the second step are: potential abort, potential inconsistency and potential failure (PA, PI, PF). Let $\mathcal{A}2$ be the set generated in this step. Finally, in the third step another set of annotations is generated. The rules in this set have correct annotations contained in the set $\mathcal{A}2$ in the premises. The properties of the simple annotations generated in the second set are related to the commit and inconsistency freedom properties (CM, WIN). Let $\mathcal{A}3$ be the set generated in this step. The generated annotation is $\mathcal{A}1 \cup \mathcal{A}2 \cup \mathcal{A}3$.

The algorithm is given in the appendix A. The proof of its correctness can be found in [8]. Here, we simply analyze the behavior of the rules of the algorithm, considering in the discussion the definition of a single predicate q. For each rule, we point out the step in which it is used. Note that speaking of predicates instead of transactions cannot generate any ambiguity because of the definition of transaction (see Section 2).

Abort

- We are sure that an abort for q arises if at least one pair of update atoms, each of them contained in a rule defining q, verifies the covering condition. (STEP 1)
- If for a pair of update atoms $+p(\tilde{t})$ and $-p(\tilde{u})$ appearing respectively in rule r_1 and rule r_2 defining q it is not know if the covering condition is verified but the bindings given to these updates are compatible (that is the situation $+p(a, X)$ and $-p(b, Y)$, with $a \neq b$ is not allowed), then we identify a potential abort condition for the generation of an inconsistent set of updates. (STEP 2)
- If for all the rules defining q none of the previous conditions is verified, we identify a commit property for the predicate. (STEP 3)

Inconsistencies

- If two update atoms contained in the same rule r defining q satisfy the covering condition, at least one inconsistent solution will be generated. Therefore we identify a certain inconsistency property in r. (STEP 1)
- Suppose that two update $+p(\tilde{t})$ and $-p(\tilde{u})$ are contained in a rule r defining q but it is not know if the covering condition is satisfied. In this case we identify a potential inconsistency property if the binding given to the two update atoms are compatible (that is the situation $+p(a, X)$ and $-p(b, Y)$, with $a \neq b$ is not allowed). Indeed, if previous property does not hold, surely the covering condition is not verified. (STEP 2)
- If for a rule none of the previous conditions is verified, the rule is inconsistency free. (STEP 3)

Failure

- If a rule r defining q has no body and does not contain any pairs of updates satisfying the covering condition, then at least one solution will be generated (at least one consistent solution exists). We identify a certain success property. (STEP 1)
- The failure for a rule r defining q is certain if all the solutions are inconsistent (note that the other cases of failure depend on the extensional database). This can be decided statically if for each pair of updates like $+p(\tilde{t})$ and $-p(\tilde{u})$ inside a rule, $p(\tilde{t}) = p(\tilde{u})$. But in this case, the canonical form, obtained as the unfolding of the intensional database, does not contain this rule [20]. Therefore, no certain failure properties arise from the canonical form of the intensional database.
- If for a rule r defining q none of the previous conditions is verified then the failure property is only potential. (STEP 2)

5 Conclusions and future work

In this paper we have presented a formal method to statically analyze intensional databases modeling transactions. The language we consider, U-Datalog, is an extension of Datalog, modeling updates and transactions. The goal of the analysis is to statically identify (and to maintain in a set, called *annotation*) several conditions (inconsistency, abort/commit, failure/success) that may occur during transaction evaluation on an intensional database. The information contained in the annotation can be used as a base to test the correctness of an intensional database and for transaction optimization. An algorithm to compute a correct annotation for non recursive normal forms of intensional databases is proposed.

Currently we are working on the definition of a canonical form for the recursive case [7] Future work includes the formalization of conditions allowing an efficient test for the covering condition and the definition of an algorithm to generate an annotation for recursive normal form of intensional databases. Another interesting topic concerns the study of the relationships between our technique and abstract interpretations [11, 12]. We claim that abstract interpretation can be used as a formal model for our analysis.

References

[1] S. Abiteboul. Updates, a New Frontier. In M. Gyssens, J.Paredaens, and D. Van Gucht, editors, *Proc. Second Int'l Conf. on Database Theory*, volume 326 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, Berlin, 1988.

[2] S. Abiteboul and V. Vianu. Equivalence and Optimization of Relational Transactions. *Journal of the ACM*, 35(1):70–120, January 1988.

[3] K.R. Apt. Introduction to Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 493–571. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990. Volume B: Formal Models and Semantics.

[4] F. Bancilhon and R. Ramakrishnan. An Amatures's Introduction to Recursive Query Processing Strategies. In *Proc. Int'l Conf. ACM on Management of Data*, pages 16–52, 1986.

[5] 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.

[6] E. Bertino and D. Musto. Correctness of Semantic Integrity Checking in Database Management Systems. *Acta Informatica*, 26:25–57, 1988. Springer-Verlag, Berlin.

[7] E. Bertino, A. Bertagna, and B. Catania. A Canonical Form for Recursive Intensional Databases. In preparation.

[8] B. Catania. *Problemi di Ottimizzazione in un Linguaggio di Vincoli per Basi di Dati Deduttive*. Master Thesis, University of Genova, April 1993. (In Italian).

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

[10] U. S. Chakravarthy, J. Grant, and J. Minker. Foundations of Semantic Query Optimization for Deductive Databases. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 243–273. Morgan Kaufmann, 1988.

[11] P. Cousot, and R. Cousot. Abstract Interpretation and Application to Logic Programs. *Journal of Logic Programming*, 13(3) 327–341, 1990.

[12] S. K. Debray. Dataflow Analysis of Logic Programs. *Journal of the ACM*, 1992.

[13] M. A. Derr, G. Phipps, and K. A. Ross. Glue-Nail: A Deductive Database System. In J. Clifford and R. King, editors, *Proc. Int'l Conf. ACM on Management of Data*, pages 308–317, 1991.

[14] M. Gabbrielli, R. Giacobazzi and D. Montesi. Modular Logic Programs on Finite Domains. In *Atti dell'Ottavo Convegno sulla Programmazione Logica*, 663–678, June 1993.

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

[16] E. Hanson and J. Widom. Rule Processing in Active Database Systems. *International Journal of Expert Systems*, 6(1):83–119, 1993.

[17] J. Gray and A. Reuter. *Transaction Processing Concepts and Tecniques.* Morgan-Kaufmann, 1993.

[18] P.C. Kanellakis. Elements of Relational Database Theory logic. In J. Van Leeuwen, editor, *Handbook of Theoretica Computer Science*, pages 1073–1144. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990. Volume B: Formal Models and Semantics.

[19] M. Karke and J. Koch. Query Optimization in Database Systems. *ACM Computing Surveys*, 16(2):111–152, June 1984.

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

[21] Y. Sagiv. Optimizing Datalog Programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann, 1988.

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

A The algorithm

Before presenting the algorithm, for the sake of simplicity, the following predicates are introduced. Given a rule r, $body(r)$ denotes the set of non-update atoms in the body of r, $head(r)$ the head of r and $c^u(r)$ the set of update atoms in r.

$B(r)$ iff $body(r) = \emptyset$
$C(a,b)$ iff $a = \alpha p(\tilde{t})$ and $b = \beta p(\tilde{u})$, $\alpha \neq \beta$
$D(a,b)$ iff $n_a \rhd n_b$ or $n_b \rhd n_a$
$E(a,b)$ iff $C(a,b)$, the h-th argument of the update a is the constant c, the h-th argument of the update b is the constant d and $c \neq d$.

FIRST STEP: Let $\mathcal{A}1$ be the set of annotations obtained applying the following rules till no more simple annotations are generated.

$$pred(head(r_1)) = pred(head(r_2)) = q,\ a_1 \in c^u(r_1),\ a_2 \in c^u(r_2),\ D(a_1,a_2) \Rightarrow \langle q, -, AB\rangle$$

$$pred(head(r)) = q,\ a_1, a_2 \in c^u(r),\ D(a_1,a_2) \Rightarrow \langle q, \{n_{a_1}, n_{a_2}\}, IN\rangle$$

$$pred(head(r)) = q,\ B(r),\ \forall a_1, a_2 \in c^u(r) : \textbf{not}\ D(a_1,a_2) \Rightarrow \langle q, \{n_r\}, SC\rangle$$

SECOND STEP: Let $\mathcal{A}2$ be the set of annotations obtained applying the following rules till no more simple annotations are generated.

$pred(head(r_1)) = pred(head(r_2)) = q$,$a_1 \in c^u(r_1)$, $a_2 \in c^u(r_2)$, $C(a_1, a_2)$, **not** $E(a_1, a_2)$, $\langle q, -, AB \rangle \notin \mathcal{A}1 \Rightarrow \langle q, \{n_{a_1}, n_{a_2}\}, PA \rangle$

$pred(head(r)) = q$, $a_1, a_2 \in c^u(r)$,$C(a_1, a_2)$, **not** $E(a_1, a_2)$,$\langle q, \{n_{a_1}, n_{a_2}\}, IN \rangle \notin \mathcal{A}1$ $\Rightarrow \langle q, \{n_{a_1}, n_{a_2}\}, PI \rangle$

$\langle q, \{n_r\}, SC \rangle \notin \mathcal{A}1 \Rightarrow \langle q, \{n_r\}, PF \rangle$

THIRD STEP: Let $\mathcal{A}3$ the set of simple annotations obtained applying the following rules till no more simple annotations are generated.

$\langle q, -, AB \rangle \notin \mathcal{A}1$ **and**
$\forall r_1, r_2 \in IDB$: $pred(head(r_1)) = pred(head(r_2)) = q$, $\forall a_1 \in c^u(r_1), \forall a_2 \in c^u(r_2)$
$\langle q, \{n_{a_1}, n_{a_2}\}, PA \rangle \notin \mathcal{A}2$ **and**
$\forall r \in IDB$: $pred(head(r)) = q$ $\forall a_1, a_2 \in c^u(r)$, $\langle q, \{n_{a_1}, n_{a_2}\}, PA \rangle \notin \mathcal{A}2$
$\Rightarrow \langle q, -, CM \rangle$

$pred(head(r)) = q$, $\forall a_1, a_2 \in c^u(r)$, $\langle q, \{n_{a_1}, n_{a_2}\}, IN \rangle \notin \mathcal{A}1$ **and** $\langle q, \{n_{a_1}, n_{a_2}\}, PI \rangle \notin \mathcal{A}2$
$\Rightarrow \langle q, \{n_r\}, WIN \rangle$

return $\mathcal{A}1 \cup \mathcal{A}2 \cup \mathcal{A}3$.

B An ordering relation on the correct annotations for an intensional database

Definition 7 Let $\langle p, i, prop \rangle$ and $\langle p', i', prop' \rangle$ be two simple annotations. $\langle p, i, prop \rangle \leq \langle p', i', prop' \rangle$ if: (i) $p = p'$; (ii) $i = i'$; (iii) $prop$ and $prop'$ are related to the same error and either $prop$ is certain or both $prop$ and $prop'$ are potential. □

Definition 8 Let IDB be an intensional database, let C_1 and C_2 be two correct annotations for IDB. $C_1 \subseteq C_2$ iff $\forall \langle p_2, i_2, prop_2 \rangle \in C_2$ $\exists \langle p_1, i_1, prop_1 \rangle \in C_1$ such that $\langle p_1, i_1, prop_1 \rangle \leq \langle p_2, i_2, prop_2 \rangle$. C_1 is therefore more precise than C_2. □

For example the annotation $\{\langle p, \{1\}, SC \rangle, \langle q, \{2, 3\}, PA \rangle\} \subseteq \{\langle p, \{1\}, PF \rangle\}$. Note that the set of all the possible correct annotations for an intensional database together with the previous order, defines a complete lattice.