BG-16 (1991) H.I [illegible]

Query Optimization by Using Knowledge about Data Semantics

Elisa Bertino (*) and Daniela Musto (**)
(*) Dipartimento di Matematica - Universita' di Genova
Via L.B. Alberti 4, 16132 Genova
e-mail: bertino@igecuniv.bitnet
(**) Istituto di Elaborazione dell'Informazione - CNR
Via S.Maria 46, 56126 Pisa (Italy)

January 29, 1991

Abstract

In this paper we address the problem of query optimization by using semantic properties of data. The discussion is in terms of the relational data model. We present some rules for Relational Algebra (RA) queries that can be used to transform a query into an equivalent one which is more efficient to process, and provide a formal proof of their correctness. Such rules, called *semantic rules*, allow to eliminate joins, to introduce clustering indexes and to test for query emptiness, according to the database properties stated by a set of integrity constraints. The correctness of these rules is formally proved by showing that the obtained queries define exactly the same set of tuples as the original query. We also investigate the problem of using semantic rules within transactions, where any arbitrary sequence of queries and modify operations may occur and semantic integrity can be violated during intermediate steps of processing. Conditions are provided under which the semantic rules presented in this paper can be correctly applied within compiled transactions.

Keywords – Relational Algebra, query optimization, semantic properties of data, semantic integrity of data, transactions.

1 Introduction

The problem of query optimization, in both centralized and distributed databases, has been widely studied and many approaches have been proposed and/or implemented. Surveys are presented in [Jark 84] and [Yu 84], while several aspects are discussed in detail in a comprehensive book [Quer 85]. Traditional approaches to the query optimization problem are based only on the knowledge of data structures and physical organization, such as access paths, and the query itself. Also transformations performed on queries are mainly syntactic ones.

However, in other researches [Wied 84] it has been pointed out the need of investigating the usage of different types of knowledge in the process of query optimization. An interesting approach, often referred to as *semantic query optimization*, is the usage of semantic properties of data to transform a given query Q into a query $\underline{Q}$ which is equivalent to the original one but which is more efficient to process. Data semantic properties are predicates that must be true for each element of a certain data set, or for each combination of elements of a certain group of data sets. A typical usage of semantic properties has been as integrity constraints [Bert 88, Brod 78, Eswa 75, Nico 82, Ston 75] that ensure the correctness of database updates.

However, semantic query optimization is still a rather unexplored area and only few studies have been reported [Hamm 80, King 81, Chak 86, Shen 87]. Some of the issues involved in this

BG-16

area are: (1) identification of the types of data semantic properties useful for query transformation; (2) identification of all possible transformations and formal proof of their correctness; (3) definition of heuristics to guide the process of query transformation.

In [Hamm 80] a framework for semantic query optimization is presented and several important issues are discussed. The authors describe some types of semantic properties to be used for query transformation and a language in which the properties are defined. An important point also made in [Hamm 80] concerns the method used to control the transformation process. In fact, given a query there may be many possible ways of transforming such a query into equivalent ones. Therefore a heuristic must be defined to select only those transformations that may actually reduce the query cost.

In the work reported in [King 81] a system called QUIST (QUery Improvement through Semantic Transformation) is described that demonstrates the feasibility of query processing improvements through query transformations by using semantic rules for a class of relational queries. The system uses detailed knowledge of the data semantic properties as well as detailed knowledge of the database physical organization.

In [Chak 86] an approach called on *semantic compilation* is presented. Under this approach, a set of valid and useful integrity constraints are associated with each relation in the database. Integrity constraints are represented in the form of clauses. A query on a relation can then be transformed by using the constraints associated with the relation without searching the entire database. The major weakness of this approach is that no method is described to determine which rules can be profitably applied for a specific query. This problem is overcome in [Shen 87], where a scheme is proposed that dynamically selects from a large set of applicable integrity constraints only the profitable ones for a relation in a query context. Under this approach, the same formalism as in [Chak 86] is used for integrity constraint definition.

In the present paper we address the problem of providing the correctness of a set of semantic rules for query transformations. In fact, if semantic optimization is to be viable, it is important to provide some formal proofs of the correctness of such transformations. That is, we show that the query obtained from a given query by applying semantic transformation techniques provides the same answer as the original query.

Moreover, we address the problem of semantic query optimization in the framework of general compiled transactions. In a compiled transaction (transaction from now on) the query execution strategy is determined before the transaction is actually executed, so that the optimization cost may be amortized over several executions. Therefore, semantic query transformation using integrity constraints is executed at transaction compile time. A transaction in its general form contains arbitrarily interleaved queries and modify operations. It may happen that some of the intermediate transaction modifications cause an inconsistent state of the database. For example, a program may withdraw money from one account and deposit it into another by updating two tuples. After the first update, the state of the database may violate the integrity constraint concerning the balancing of the accounts. This implies that a transaction must be allowed to violate semantic integrity during intermediate steps of processing. A problem may arise if during a transaction a query is executed which has been transformed at compile time by using some integrity constraint that has been violated by some preceding modify operation. In this case, in fact, the integrity constraint is invalid and its usage in optimizing the query may produce an incorrect answer. In the paper we present conditions preventing this type of incorrect situation. The approach presented here extends our previous work presented in [Bert 88], where the assumption is made that the transaction queries are all executed before the modify operations. In this paper, we release this assumption, by allowing arbitrary interleaving of queries and modify operations.

The discussion will be in terms of the relational data model [Codd 70]. The class of queries for which the transformations are defined is a subset of the queries that can be expressed in terms of the Relational Algebra [Codd 70].

Our work differs with respect to recent works, such as [Chak 86] and [Shen 87], under two aspects. The first is that our techniques are in the framework of Relational Algebra. Since, most query optimizers are based on the set processing primitives of the Relational Algebra, our approach is more directly applicable. However, the most relevant difference is that we address the problem of semantic query optimization in the framework of compiled transactions containing interleaved queries and modify operations. This problem, at the best of our knowledge, has not been addressed elsewhere.

The remainder of this paper is organized as follows. In Section 2 the basic definitions and the notation are introduced. In Section 3 a formal definition of integrity constraint is given in terms of Relational Algebra. In Section 4 some semantic rules for query transformation are proposed. The formal proof of the correctness of these rules is presented in Section 5. Finally in Section 6 the problem of semantic query optimization is discussed in the framework of general compiled transactions.

2 Preliminaries

For the notation and the definitions not explicitly included here refer to [Ullm 82]. RA is an abbreviation for Relational Algebra.

2.1 RA expressions and RA-4 expressions

A RA *expression* is any expression formed legally (i.e., in accordance with the restrictions on the operators) from the base relations by using the *projection*, *selection*, *union*, *difference* and *Cartesian product* operators. Recall that every RA expression defines in turn a relation, whose scheme (resp. content) depends on the scheme (resp. content) of the base relations and on the relational operators involved in it [Codd 72].

Two RA expressions E_1 and E_2 are said to be *equivalent*, written $E_1 \equiv E_2$, if they define the same set of tuples whatever is the content of the base relations involved in them [1]. Some equivalence rules between RA expressions that are based on syntactical properties of the RA operators and that will be used in the sequel are reported in Appendix A.

In the present paper we shall focus on the RA expressions that can be derived by using the operators of projection, selection, union, and Cartesian product, called in the sequel RA-4 *expressions*. We shall use the following notation:

• $\pi_{\mathcal{A}}(R)$	*projection* of relation R onto the set $\mathcal{A}$ of attributes
• $\sigma_P(R)$	*selection* of the tuples of relation R satisfying predicate P
• $R \cup R'$	*union* of relations R and R'
• $R \times R'$	*Cartesian product* of relations R and R'

R and R' above are any base relation or any relation defined by a RA-4 expression. Predicate P in the selection is any RA predicate (see Section 2.2).

Two canonical sets are associated with each RA-4 expression, as follows:

Definition 2.1 Let E be a RA-4 expression. Then the *set of relation names* and the *set of attribute names* involved in E, denoted respectively by $\mathcal{R}el(E)$ and by $\mathcal{A}ttr(E)$, are defined recursively according to the following rules:

[1] In Section 5 we shall consider a looser form of equivalence, called *local* equivalence, which only requires that the returned set of tuples is the same for a given content of base relations.

1. If E is a base relation R, then
   - $\mathcal{R}el(E) = \{R\}$
   - $\mathcal{A}ttr(E)$ is the relational scheme of R
2. If $E = \pi_{\mathcal{A}}(E')$, then
   - $\mathcal{R}el(E) = \mathcal{R}el(E')$
   - $\mathcal{A}ttr(E) = \mathcal{A}$
3. If $E = \sigma_P(E')$, then
   - $\mathcal{R}el(E) = \mathcal{R}el(E')$
   - $\mathcal{A}ttr(E) = \mathcal{A}ttr(E')$
4. If $E = E_1 \times E_2$, then
   - $\mathcal{R}el(E) = \mathcal{R}el(E_1) \cup \mathcal{R}el(E_2)$
   - $\mathcal{A}ttr(E) = \mathcal{A}ttr(E_1) \cup \mathcal{A}ttr(E_2)$
5. If $E = E_1 \cup E_2$[2], then
   - $\mathcal{R}el(E) = \mathcal{R}el(E_1) \cup \mathcal{R}el(E_2)$
   - $\mathcal{A}ttr(E) = \mathcal{A}ttr(E_1) = \mathcal{A}ttr(E_2)$ □

Every attribute name of a base relation R (called in the following *basic attribute name*) will take the form $R.C$, where C denotes a column of R. The *extension* of the column C of R, which is expressed by the RA-4 expression $\pi_{\{R.C\}}(R)$, will be denoted in the sequel by $\mathcal{K}_{R.C}$.

2.2 RA predicates and simple predicates

A RA *predicate* is any boolean combination of *simple predicates* by means of the logical connectives *and*, *or* and *not*, and parentheses. *Simple predicates* are defined as follows:

Definition 2.2 A *simple predicate* is either a *scalar- simple predicate* or a *set- simple predicate*. In particular:

(a) a *scalar- simple predicate* is any predicate having the form $[X\ op\ Y]$, where:
- X is a basic attribute name
- op is a *scalar comparison operator* [3]
- Y is a constant or a basic attribute name;

(b) a *set- simple predicate* is any predicate having the form $[X\ op'\ Y]$, where:
- X is a basic attribute name
- op' is a *set membership operator* [4]
- Y is a set of constants or the extension of a column C of a base relation R. □

The semantics of *simple predicates* is intuitive, and therefore is omitted[5]. Given two RA predicates P_1 and P_2, by $P_1 \Rightarrow P_2$ we shall denote that P_1 *implies* P_2. P_1 and P_2 will be said *equivalent*, written $P_1 \Leftrightarrow P_2$, if both $P_1 \Rightarrow P_2$ and $P_2 \Rightarrow P_1$.

Two canonical sets are associated with each RA predicate, as follows:

[2] For our purpose it is sufficient to assume that E_1 and E_2 have the same relation scheme.
[3] i.e., op is one of the following operators: $>, \geq, =, \neq, <, \leq$.
[4] i.e., op' is one of the following operators: $\in$ (*is in*), $\notin$ (*is not in*).
[5] Remark that the inclusion of the *set membership operators* in the *simple* predicates, and then in the RA predicates, extends the Relational Algebra with respect to the original definition given in [Codd 70].

Definition 2.3 Let P be a RA predicate. Then the *set of relation names* and the *set of attribute names* involved in P, denoted respectively by $\mathcal{P}rel(P)$ and by $\mathcal{P}attr(P)$, are defined recursively according to the following rules:

1. If $P = [R.C \; op \; c]$
   or $P = [R.C \; op' \; \{c_1, c_2, .., c_n\}]$,
   where $c, c_1, c_2, ..., c_n (n \geq 1)$ are constants, then
   - $\mathcal{P}rel(P) = \{R\}$
   - $\mathcal{P}attr(P) = \{R.C\}$
2. If $P = [R.C \; op \; R'.C']$
   or $P = [R.C \; op' \; \mathcal{K}_{R'.C'}]$ then
   - $\mathcal{P}rel(P) = \{R, R'\}$
   - $\mathcal{P}attr(P) = \{R.C, R'.C'\}$
3. If $P = not(P')$ then
   - $\mathcal{P}rel(P) = \mathcal{P}rel(P')$
   - $\mathcal{P}attr(P) = \mathcal{P}attr(P')$
4. If $P = (P' \; and \; P'')$
   or $P = (P' \; or \; P'')$ then
   - $\mathcal{P}rel(P) = \mathcal{P}rel(P') \cup \mathcal{P}rel(P'')$
   - $\mathcal{P}attr(P) = \mathcal{P}attr(P') \cup \mathcal{P}attr(P'')$ □

To give ourselves more convenience, in the sequel we shall include in the set of *simple predicates* also the constant predicates *true* and *false*. The expressive power of the set, clearly, not changes. For $P = true$ and $P = false$ we shall assume $\mathcal{P}rel(P) = \mathcal{P}attr(P) = \emptyset$, where $\emptyset$ denotes the empty set.

2.3 Canonical forms for RA-4 expressions

The operators of projection, selection, union and Cartesian product can be arbitrarily combined together within RA-4 expressions. In turn the predicate P in the selection can be any complex RA predicate. Thus RA-4 expressions may present any arbitrarily complex form[6]. Below we focus on two classes of RA-4 expressions which satisfy some restrictions both on the order on which the relational operators are applied and on the form of the RA predicate in the selection.

Definition 2.4 (a) A RA-4 expression E is said to be in *D-canonical form* iff

$$E = \pi_A \sigma_P(\underline{E}),$$

where:

- $\underline{E}$ is a base relation or a Cartesian product of base relations
- P has the form $[P_1 \; or \; P_2 \; or \; \dots \; P_m](m > 1)$,

and

[6]According to Section 2.2 the predicate P in the selection can also be a constant predicate *true* or *false*. By definition the RA-4 expressions $\sigma_{true}(R)$ and $\sigma_{false}(R)$ return, respectively, the relation R itself and the *empty* relation.

- each P_i in P is a *simple predicate* or a conjunction of *simple predicates*.

(b) A RA-4 expression E is said to be in D_0*-canonical form* iff

$$E = \pi_A \sigma_P(\underline{E}),$$

where:

- $\underline{E}$ is a base relation or a Cartesian product of base relations
- P is a *simple predicate* or a conjunction of *simple predicates*. □

In spite of the restrictions the RA-4 expressions in *D-canonical form* and in D_0*-canonical form* satisfy, the following properties hold:

Property 2.1 *Given any arbitrarily complex RA-4 expression E, there exists a RA-4 expression E' equivalent to E which is either in D-canonical form or is a union of RA-4 expressions in D-canonical form* ([Bert 86a] - Theorem 3.3 and Theorem 4.1). □

Property 2.2 *For each RA-4 expression E in D-canonical form, there exists a RA-4 expression E' equivalent to E which is either in D_0-canonical form or is the union of RA-4 expressions in D_0-canonical form* ([Bert 86a] - Theorem 4.2). □

Thus RA-4 expressions in *D-canonical form* and in D_0*-canonical form* are very general. Without loss of generality we shall therefore consider only RA-4 expressions in *D-canonical form* and in D_0*-canonical form* in the sequel.

2.4 Queries and modify operations

In this section we briefly review queries and modify operations on a database.

2.4.1 Queries

A query (or *read operation*) is any RA expression. In the present paper we shall focus on those queries that are either RA-4 expressions in D_0*-canonical form* or that are union of RA-4 expressions in D_0*-canonical form*. The former will be called in the sequel D_0*-queries* while the others will be called $\mathcal{U}_{D_0}$*-queries*.

Note that the queries we are dealing with in this paper, in spite of their restricted form, cover a wide class of queries and are useful in many applications. In fact by Property 2.1 and Property 2.2 they express the whole set of queries definable by RA-4 expressions (or RA-4 *queries*), and thus they cover all the queries except those involving the RA difference operator. Moreover the framework of compiled transactions we are concerned with ensures a direct application to general RA-4 queries of the results proved for D_0- and $\mathcal{U}_{D_0}$*-queries* in the present discussion.

2.4.2 Modify operations

Modify operations are those operations which makes it possible to alter the content of base relations in a database, i.e. the *delete*, the *insert*, and the *update* operations. The meaning of each of these operations is intuitive, but there are several different definitions in use in the literature. Let R be a base relation. In the present paper we shall assume that:

- the *delete operation* takes the form $\Delta_P(R)$, and removes from R all the tuples satisfying predicate P

- the *insert operation* takes the form $\Upsilon_{\sigma_P(R')}(R)$, and appends to R all the tuples in R' satisfying predicate P
- the *update operation* takes the form $\Psi_{(\mathcal{L},P)}(R)$, where $\mathcal{L}$ is a set of m pairs (a_i, φ_i) $(1 \leq i \leq m \leq k)$, and k is the *degree* of R[7]. Each a_i in $\mathcal{L}$ is a basic attribute name for R, and each φ_i is a constant or a basic attribute name for R different from a_i or an arithmetic expression including both constants and attribute a_i. $\Psi_{(\mathcal{L},P)}(R)$ replaces the value of each attribute a_i in $\mathcal{L}$ with the value of the corresponding expression φ_i. The replacement is made only for those tuples of R that satisfy predicate P.

Predicate P in a modify operation is any RA predicate, whereas relation R' is any constant or base relation having the same relation scheme as R.

2.5 Transactions

A transaction is a work unit consisting of an application-specified sequence of database operations, whose (sequential) execution is viewed as an *atomic* action by the end-user (e.g., refer to [Date 83]). The execution of a transaction can conclude in two different ways:

- the transaction is **committed**, i.e. it terminates with all supported variations permanently stored in the database

or else

- the transaction is **aborted**, i.e. it terminates with all supported variations removed from the database.

Queries and modify operations can be arbitrarily combined together within transactions. Let T be a transaction on a database $\mathcal{D}$. If $O_1, \ldots, O_f$ $(f \geq 1)$ are, in order, all the occurrences of modify operations in T (possibly some queries occur interleaved with them), then during the execution of T the database $\mathcal{D}$ goes through at most $f+1$ *states* (or database *extensions*)

$$S_{T,0}, S_{T,1}, \ldots, S_{T,f}$$

where:

- $S_{T,0}$ (called *initial state*) is the state of $\mathcal{D}$ in which the execution of T starts
- for $1 \leq i \leq f$, $S_{T,i}$ (called *intermediate state*) is the state reached by $\mathcal{D}$ by executing O_i in the state $S_{T,i-1}$

In the following we shall denote the set of states $\{S_{T,0}, S_{T,1}, \ldots, S_{T,f}\}$ by $State(\mathcal{D},T)$[8]. Note that because a query does not modify the content of a database, then $S_{T,0}$ is also the state of $\mathcal{D}$ in which the first modify operation of T (i.e., O_1) is executed. State $S_{T,f}$, which is the state produced by the last modify operation of the transaction, is also called *final state*. All the intermediate states produced by T on the database except the final one are transparent to the end-user, and they are only seen by the transaction. $S_{T,f}$ may be visible or not by the end-user, depending on how T concludes its execution. In fact the state of $\mathcal{D}$ visible by the end-user after the execution of T is either $S_{T,f}$ or $S_{T,0}$. Entering into details, it is:

- $S_{T,f}$, if T is properly terminated (i.e., if T is committed)

[7]Recall that the *degree* of a relation is the number of attributes defining its scheme.

[8]Clearly if the transaction T contains only queries the database does not change state during the execution of T and remains in state $S_{T,0}$. In this case $State(\mathcal{D},T) = \{S_{T,0}\}$.

Consider a database consisting of the following base relations, which describes reviews loan and requests, and employees at I.E.I library:

$REVIEW\_CATALOGUE(review\_code, review\_title, starting\_year, publisher, first\_year, last\_year)$
$SUBSCRIBER(subscriber\_code, name, category, address, year)$
$LOAN\_CATALOGUE(applicant\_code, review\_code, demanded\_year, number, date, period)$
$REQUESTS\_LIST(applicant\_name, topic, review\_title, demanded\_year, number)$.

$EMPLOYEE(name, position, salary, status)$
Some integrity constraints on the database can be the following:

I_1: *Every computer science and astronomical reviews published starting from year 1984 can be found at I.E.I. library.*

I_2: *Every applicant for a computer science review must be a I.E.I. library subscriber.*

I_3: *Only professors may borrow a review from the I.E.I. library for a period greater than fifteen days.*

I_4: *Students cannot borrow a review from the I.E.I. library.*

I_5: *All library employees that are manager have a salary greater than 30,000.*

Figure 1: Integrity constraints on a database: an example

- $S_{T,0}$, if T is aborted.

In other words, when the transaction is committed the database is left in the final state. Otherwise the database is left in the initial state. This is usually achieved rollingback the changes performed by the transaction (see, e.g., [Gray 82]).

3 Integrity Constraints

Knowledge about the semantic properties of data can be represented as a set of integrity constraints. Some examples of integrity constraints are shown in Fig. 1.

3.1 Formal definition

An integrity constraint basically consists of three components:

- a *domain*, representing the data set for which the integrity constraint must hold
- a *condition*, expressing a property for the data in the domain
- a *precondition*, selecting the data in the domain for which the condition must be true.

Other components may be associated with an integrity constraint that are relevant for semantic integrity control. For instance, as described in [Bert 84], a set $\mathcal{W}$ of access types and a set $\mathcal{V}$ of actions can be defined, where $\mathcal{W}$ specifies the types of modify operations upon which it is necessary to verify that the integrity constraint is not violated, and $\mathcal{V}$ specifies the actions to be executed by the system if the integrity constraint is violated. In this paper, however, we only shall consider the three basic components of integrity constraints. In particular, we model integrity constraints as follows:

Definition 3.1 An *integrity constraint* (shortly IC) is a 3-tuple $I = \langle E, P', P'' \rangle$ where:

(a) E (representing the *domain*) is a RA-4 expression

(b) P' (representing the *precondition*) and P'' (representing the *condition*) are RA predicates

and

(c) $\mathcal{P}attr(P') \cup \mathcal{P}attr(P'') \subseteq \mathcal{A}ttr(E)$. $\Box$

Condition (c) of Definition 3.1 simply asserts that the precondition and the condition of an IC are defined only on attributes of its domain, and then ensures that the IC is well-formulated.

Next property follows as a direct consequence of condition (c) of Definition 3.1:

Property 3.1 Let $I = \langle E, P', P'' \rangle$ be an IC. Then

$$\mathcal{P}rel(P') \cup \mathcal{P}rel(P'') \subseteq \mathcal{R}el(E). \ \Box$$

In the sequel we shall denote by $P(t)$ the evaluation of predicate P on tuple t, and by $[\![I]\!]$ the assertion on the database stated by the IC I, which is defined as follows:

Definition 3.2 Let $I = \langle E, P', P'' \rangle$ be an IC. Then

$$[\![I]\!] =^{def} \forall t \in E' \;\; P''(t), \;\text{ where } E' = \sigma_{P'}(E). \Box$$

We shall say that $I = \langle E, P', P'' \rangle$ *holds* in a database state (or, equivalently, that a database state is *consistent* with I) iff $[\![I]\!]$ is verified by that state. Note that whenever I holds then $P' \Rightarrow P''$ for all the tuples in E.

A formal definition of the ICs presented in Fig.1 is shown in Fig.2. Remark that there are different ways in general to formally define the same integrity constraint.

Definition 3.3 Two integrity constraints I and I' are said to be *equivalent*, written $I \equiv I$, iff $[\![I]\!] \Leftrightarrow [\![I']\!]$. $\Box$

Thus two ICs are equivalent if they define equivalent assertions on the database.

Let us briefly consider now the case of sets of ICs. For any set of ICs $\mathcal{I}$ we shall denote by $[\![\mathcal{I}]\!]$ the assertion on the database stated by $\mathcal{I}$.

Definition 3.4 *Let* $\mathcal{I} = \{I_1, I_2, ..., I_n\}$ *be a set of ICs* $(n > 1)$. *Then*

$$[\![\mathcal{I}]\!] =^{def} [\![I_1]\!] \textit{ and } [\![I_2]\!] \textit{ and...and } [\![I_n]\!]. \Box$$

Clearly $\mathcal{I}$ *holds* in a database state (or equivalently a database state is *consistent* with $\mathcal{I}$) iff $[\![\mathcal{I}]\!]$ is verified by that state.

Definition 3.5 Two sets of ICs $\mathcal{I}$ and $\mathcal{I}'$ are said to be *equivalent*, written $\mathcal{I} \equiv \mathcal{I}'$, iff $[\![\mathcal{I}]\!] \Leftrightarrow [\![\mathcal{I}']\!]$. $\Box$

Recall that when a set of ICs is defined for a database the database must preserve the semantic integrity of data with respect to this set. Thus if $\mathcal{I}_{\mathcal{D}}$ is the set of ICs defined for the database $\mathcal{D}$ then $\mathcal{I}_{\mathcal{D}}$ must hold in every state of $\mathcal{D}$ visible by the end-user, otherwise an error is pointed out by the system. Therefore $\mathcal{I}_{\mathcal{D}}$ must hold in every state of $\mathcal{D}$ in which a transaction T is started. This assumption is released for the database states produced by the intermediate steps of the transaction, except for $S_{T,f}$ (see Section 2.5), which are not visible by the end-user. Semantic integrity of data, in fact, can be violated during the execution of a transaction without forcing the transaction to be aborted (a case in which this situation occurs is, for example, that mentioned in the Introduction, concerning the bank-accounts administration).

$$\begin{aligned}
I_1 = \langle\; & \mathit{REQUESTS\_LIST} \times \mathit{REVIEW\_CATALOGUE}, \\
& [\mathit{REQUESTS\_LIST.topic} = \text{`}\mathit{computerscience}\text{'}] \\
& \mathit{or}\ [\mathit{REQUESTS\_LIST.topic} = \text{`}\mathit{astronomy}\text{'}], \\
& [\mathit{REQUESTS\_LIST.review\_title} \in \mathcal{K}_{\mathit{REVIEW\_CATALOGUE.review\_title}}] \\
& \mathit{or}[\mathit{REQUESTS\_LIST.demanded\_year} < 1984]\ \rangle
\end{aligned}$$

$$\begin{aligned}
I_2 = \langle\; & \pi_{\{\mathit{REQUESTS\_LIST.applicant\_name}, \mathit{SUBSCRIBER.name}\}} \\
& \sigma_{[\mathit{REQUEST\_LIST.topic}=\text{'}\mathit{computerscience}\text{'}]}(\mathit{REQUESTS\_LIST} \times \mathit{SUBSCRIBER}), \\
& \mathit{true}, \\
& [\mathit{REQUESTS\_LIST.applicant\_name} \in \mathcal{K}_{\mathit{SUBSCRIBER.name}}]\rangle
\end{aligned}$$

$$\begin{aligned}
I_3 = \langle\; & \sigma_{[\mathit{LOAN\_CATALOGUE.applicant\_code}=\mathit{SUBSCRIBER.subscriber\_code}]}(\mathit{LOAN\_CATALOGUE} \times \\
& \mathit{SUBSCRIBER}), \\
& [\mathit{LOAN\_CATALOGUE.period} > 15], \\
& [\mathit{SUBSCRIBER.category} = \text{`}\mathit{professor}\text{'}]\ \rangle
\end{aligned}$$

$$\begin{aligned}
I_4 = \langle\; & \sigma_{[\mathit{LOAN\_CATALOGUE.applicant\_code}=\mathit{SUBSCRIBER.subscriber\_code}]}(\mathit{LOAN\_CATALOGUE} \times \\
& \mathit{SUBSCRIBER}), \\
& \mathit{true}, \\
& [\mathit{SUBSCRIBER.category} \neq \text{`}\mathit{student}\text{'}]\rangle
\end{aligned}$$

$$\begin{aligned}
I_5 = \langle\; & \mathit{EMPLOYEE}, \\
& [\mathit{EMPLOYEE.position} =' \mathit{manager}'], \\
& [\mathit{EMPLOYEE.salary} > 30,000]\rangle
\end{aligned}$$

Figure 2: Formal definition of the integrity constraints presented in Fig.1

3.2 Simple integrity constraints

In this section we focus on a class of ICs, called *simple* ICs, which satisfy some restrictions both on the form of the domain, and on the form of the condition and precondition.

Definition 3.6 An IC $I = \langle E, P', P'' \rangle$ is said to be *simple* iff:

(a) E is a base relation or a Cartesian product of base relations

(b) for each base relation R in E is
$R \in \mathcal{P}rel(P') \cup \mathcal{P}rel(P'')$

(c) P' is a *simple predicate* or a conjunction of *simple predicates*

(d) P'' is a *simple predicate* or a disjunction of *simple predicates*. □

Note that by condition (b) of Definition 3.6 each base relation in the domain of a *simple* IC is either involved in the precondition or in the condition of the *simple* IC, or in both. Thus the domain of every *simple* IC I is defined only on base relations on which I really expresses a property and therefore is not redundant. Next property follows as a direct consequence of condition (b) of Definition 3.6 and of Property 3.1:

Property 3.2 Let $I = \langle E, P', P'' \rangle$ be a *simple* IC. Then

$$\mathcal{R}el(E) = \mathcal{P}rel(P') \cup \mathcal{P}rel(P''). \ \square$$

In spite of the restrictions *simple* ICs satisfy, the following theorem holds (the proof of this theorem is presented in Appendix B):

Theorem 3.1 *Given a set of ICs $\mathcal{I}$, there exists a set of simple ICs $\mathcal{I}'$ such that $\mathcal{I} \equiv \mathcal{I}'$.* □

Thus *simple* ICs preserve the same expressive power of the whole class of ICs. Without loss of generality we shall therefore consider only *simple* ICs in the remainder of this paper.
To conclude this section, we present in Fig.3 an equivalent formulation in terms of *simple* ICs of the set of ICs shown in Fig.2. Note that:

$\{I_1\} \equiv \{I_{s_{1,1}}, I_{s_{1,2}}\}$

$I_2 \equiv I_{s_2}$

$I_3 \equiv I_{s_3}$

$I_4 \equiv I_{s_4}$

$I_5 \equiv I_{s_5}$

The proof of the previous equivalences is straightforward and is obtained from Lemma B.5, Lemma B.2, and Lemma B.3 in Appendix B (we omit it for brevity).

$I_{s_{1,1}} = \langle$ *REQUESTS_LIST* $\times$ *REVIEW_CATALOGUE*,
[*REQUESTS_LIST.topic* = 'computerscience'],
[*REQUESTS_LIST.review_title* $\in \mathcal{K}_{REVIEW\_CATALOGUE.review\_title}$]
or[*REQUESTS_LIST.demanded_year* < 1984] $\rangle$

$I_{s_{1,2}} = \langle$ *REQUESTS_LIST* $\times$ *REVIEW_CATALOGUE*,
[*REQUESTS_LIST.topic* = 'astronomy'],
[*REQUESTS_LIST.review_title* $\in \mathcal{K}_{REVIEW\_CATALOGUE.review\_title}$]
or[*REQUESTS_LIST.demanded_year* < 1984] $\rangle$

$I_{s_2} = \langle$ *REQUESTS_LIST* $\times$ *SUBSCRIBER*,
REQUESTS_LIST.topic =' computerscience',
[*REQUESTS_LIST.applicant_name* $\in \mathcal{K}_{SUBSCRIBER.name}$]$\rangle$

$I_{s_3} = \langle$ *LOAN_CATALOGUE* $\times$ *SUBSCRIBER*,
[*LOAN_CATALOGUE.applicant_code* = *SUBSCRIBER.subscriber_code*]
and [*LOAN_CATALOGUE.period* > 15],
[*SUBSCRIBER.category* = 'professor'] $\rangle$

$I_{s_4} = \langle$ *LOAN_CATALOGUE* $\times$ *SUBSCRIBER*,
[*LOAN_CATALOGUE.applicant_code* = *SUBSCRIBER.subscriber_code*]
[*SUBSCRIBER.category* $\neq$ 'student']$\rangle$

$I_{s_5} = \langle$ *EMPLOYEE*,
[*EMPLOYEE.position* =' manager'],
[*EMPLOYEE.salary* > 30,000]$\rangle$

Figure 3: Formulation of the set of integrity constraints listed in Fig.1 in terms of *simple* integrity constraints

4 Semantic Rules for Query Transformation

Query transformation can be performed by using syntactical properties of RA operators (as is the case of the rules presented in Appendix A) as well as semantic properties of data. In this section we focus on this second aspect, and propose some rules, called in the following *semantic rules*, which transform a query by modifying the predicate in the selection and by eliminating a join according to the database properties stated by a set of integrity constraints. Join elimination is particularly important since joins are usually very expensive to perform. The modification of the predicate in the selection, in turn, comes out to be useful not only to reduce the predicate to a simpler form but even to introduce clustering indexes in the query, or to recognize that the query is empty without evaluating it.

The semantic rules we present below apply to D_0- and U_{D_0}*-queries* (see Section 2.4.1), however they can be easily extended to arbitrary RA-4 queries because of Property 2.1 and Property 2.2. In addition they only make use of *simple* integrity constraints, but this is not a restrictive assumption because of Theorem 3.1. First in Section 4.1 we introduce a semantic rule for modifying the predicate in the selection of a D_0*-query*, without any concern for join elimination. Then in Section 4.2 we propose three semantic rules for eliminating a join in D_0*-queries*. In Section 4.3 we consider the case of U_{D_0}*-queries*. Finally in Section 4.4 we provide a sample application. The proof of the correctness of the semantic rules presented in this section is given in Section 5. In Section 6 we shall investigate their application within transactions.

4.1 Modification of the predicate in the selection

A semantic rule is presented which transforms a D_0*-query* by modifying the predicate in the selection.

S-Rule 4.1 Let $\mathcal{I}_\mathcal{D}$ be the set of *simple* ICs defined for a database $\mathcal{D}$, and let $Q = \pi_\mathcal{A}\sigma_P(E)$ be a D_0*-query* on $\mathcal{D}$ ($P \neq false$).

If for some $I_i = (E_i, P_i', P_i'')$ in $\mathcal{I}_\mathcal{D}$ the following conditions hold

1. $\mathcal{R}el(E_i) \subseteq \mathcal{R}el(E)$
2. $P = P_1$ *and* P_2, where $P_1 \Rightarrow P_i'$

then Q can be transformed with respect to I_i into the query

$$\underline{Q} = \pi_\mathcal{A}\sigma_{\underline{P}}(E),$$

where

$$\underline{P} = \begin{cases} P_1 & \text{if } P_i'' \Rightarrow P_2 \quad (case1) \\ false & \text{if } P_i'' \Rightarrow not\ P_2 \quad (case2) \\ P\ and\ P_i'' & \text{otherwise} \quad (case3)\ \square \end{cases}$$

The predicates P_1 and P_2 in S-Rule 4.1 are *simple* predicates or conjunctions of *simple* predicates. Note that (*case 1*) of S-Rule 4.1 allows the predicate in the original query to be transformed by eliminating a conjunct, while (*case 3*) allows a new predicate to be added to the original predicate of the query (the addition of a predicate, for instance, may be useful when this predicate is defined on an attribute on which there is an index). Finally (*case 2*) allows to deduce that the query is empty without evaluating it.

We introduce the following definition:

Definition 4.1 An IC I_i is said to be *reducing* for a D_0*-query* Q with respect to a RA predicate P iff Q can be transformed with respect to I_i by using S-Rule 4.1, and the application of S-Rule 4.1 produces the elimination of P from the selection of Q. □

Thus an IC I_i is *reducing* with respect to a D_0*-query* Q whenever I_i can be used to transform Q according to (*case 1*) of S-Rule 4.1.

4.2 Join elimination

Three semantic rules are presented which transform a D_0*-query* by eliminating a join. They concern the case of *equijoins*.

S-Rule 4.2 Let $\mathcal{I}_\mathcal{D}$ be the set of *simple* ICs defined for a database $\mathcal{D}$, and let $Q = \pi_\mathcal{A}\sigma_P(E)$ be a D_0*-query* on $\mathcal{D}$ such that :

(a) $E = R_1 \times R_2$

(b) $\mathcal{A} \subseteq \mathcal{A}ttr(R_1)$

(c) $P = P_1$ *and* $[R_1.C_1 = R_2.C_2]$

and

(d) $P_1 = true$,
or $\mathcal{P}rel(P_1) = \{R_1\}$

If for some $I_i = (E_i, P_i', P_i'')$ in $\mathcal{I}_\mathcal{D}$ the following conditions hold

1. $\mathcal{R}el(E_i) \subseteq \mathcal{R}el(E)$
2. $P_1 \Rightarrow P_i'$
3. either $P_i'' = [R_2.C_2 = c]$
or $P_i'' = P_{i,1}''$ *or* $[R_2.C_2 = c]$, where $P_1 \Rightarrow not\ P_{i,1}''$

then Q can be transformed with respect to I_i into the query

$$\underline{Q} = \pi_\mathcal{A}\sigma_{\underline{P}}(R_1),$$

where

$$\underline{P} = P_1\ and\ [R_1.C_1 = c].\ \square$$

Predicates P_1 and $P_{i,1}''$ in S-Rule 4.2 are *simple* predicates or, respectively, a conjunction and a disjunction of *simple* predicates. Condition 3 in S-Rule 4.2 simply states that whenever P_i'' is not a *simple* predicate the component $P_{i,1}''$ of P_i'' must not hold in order to apply S-Rule 4.2. S-Rule 4.2 asserts in particular that the presence of an integrity constraint forcing a column of a relation R_2 to have a constant value may produce as a result the simplification of a D_0*-query* Q requiring an equijoin between R_2 and some relation R_1. This occurs whenever (1) the column of R_2 involved in the equijoin is the same forced by the integrity constraint to be a constant, (2) the set $\mathcal{A}$ of attributes in the projection of Q only includes attributes of R_1, and (3) the predicate P_1 in the selection of Q is either trivially true or is defined only on attributes of R_1. In this case Q can be transformed into a more efficient query $\underline{Q}$ defined only on relation R_1. Note that the constant value expressed by the integrity constraint for the column C_2 of R_2 is forced in the selection of $\underline{Q}$ on the column of R_1 that is involved in the equijoin in Q.

S-Rule 4.3 Let $\mathcal{I}_\mathcal{D}$ be the set of *simple* ICs defined for a database $\mathcal{D}$, and let $Q = \pi_\mathcal{A}\sigma_P(E)$ be a D_0*-query* on $\mathcal{D}$ such that :

(a) $E = R_1 \times R_2$

(b) $\mathcal{A} \subseteq \mathcal{A}ttr(R_1)$

(c) $P = P_1$ *and* $[R_1.C_1 = R_2.C_2]$

and

(d) $P_1 = true$,
or $\mathcal{P}rel(P_1) = \{R_1\}$

If for some $I_i = (E_i, P_i', P_i'')$ in $\mathcal{I}_\mathcal{D}$ the following conditions hold

1. $\mathcal{R}el(E_i) \subseteq \mathcal{R}el(E)$
2. $P_1 \Rightarrow P_i'$
3. either $P_i'' = [R_1.C_1 \in \mathcal{K}_{R_2.C_2}]$
or $P_i'' = P_{i,1}''$ *or* $[R_1.C_1 \in \mathcal{K}_{R_2.C_2}]$, where $P_1 \Rightarrow$ *not* $P_{i,1}''$

then Q can be transformed with respect to I_i into the query

$$\underline{Q} = \pi_\mathcal{A}\sigma_{P_1}(R_1). \ \square$$

S-Rule 4.3 is similar to S-Rule 4.2, but in this case an inclusion relationship holds between a column of R_1 and a column of R_2. This fact allows to transform Q by simply eliminating the equijoin whenever the hypotheses of S-Rule 4.3 are satisfied by Q.

Next semantic rule extends the previous ones to the case in which some additional constraints on relation R_2 are included in the selection predicate of Q.

S-Rule 4.4 Let $\mathcal{I}_\mathcal{D}$ be the set of *simple* ICs defined for a database $\mathcal{D}$, and let $Q = \pi_\mathcal{A}\sigma_P(E)$ be a D_0*-query* on $\mathcal{D}$ such that:

(a) $E = R_1 \times R_2$

(b) $\mathcal{A} \subseteq \mathcal{A}ttr(R_1)$

(c) $P = P_1$ *and* P_2 *and* $[R_1.C_1 = R_2.C_2]$

and

(d) $P_1 = true$
or $\mathcal{P}rel(P_1) = \{R_1\}$

(e) $\mathcal{P}rel(P_2) = \{R_2\}$.

If for some I_{i_1}, I_{i_2} in $\mathcal{I}_\mathcal{D}$ the following conditions hold

1. I_{i_1} is *reducing* for Q with respect to P_2
2. I_{i_2} satisfies with respect to Q either the conditions of S-Rule 4.2 (*case 1*) or those of S-Rule 4.3 (*case 2*)

then Q can be transformed with respect to $\{I_{i_1}, I_{i_2}\}$ into the query

$$\underline{Q} = \pi_\mathcal{A}\sigma_{\underline{P}}(R_1)$$

where

$$\underline{P} = \begin{cases} P_1 \text{ and } [R_1.C_1 = c] & \text{if } (\textit{case 1}) \text{ is verified} \\ P_1 & \text{if } (\textit{case 2}) \text{ is verified } \square \end{cases}$$

S-Rule 4.4 asserts that a query Q requiring an equijoin between two relations R_1 and R_2 can be transformed into a query $\underline{Q}$ defined only on relation R_1 even in the case in which there is a predicate in the selection of Q that is defined only on attributes of R_2. This occurs when there exists an IC that is *reducing* for Q with respect to P_2 and either S-Rule 4.2 or S-Rule 4.3 can be applied accordingly.

All the semantic rules presented in this section can be easily generalized to the case in which the query Q is defined on the Cartesian product of more than two relations.

4.3 Transforming U_{D_0}*-queries* by using semantic rules

U_{D_0}*-queries* can be transformed by using the following semantic rule:

S-Rule 4.5 Let $\mathcal{I}_\mathcal{D}$ be the set of *simple* ICs defined for a database $\mathcal{D}$, and let Q be a U_{D_0}*-query* on $\mathcal{D}$ such that

$$Q = \bigcup_{i=1}^{n} Q_i$$

for some D_0*-queries* $Q_1, \ldots, Q_n$ $(n > 1)$.
Then Q can be transformed with respect to a subset $\mathcal{I}$ of $\mathcal{I}_\mathcal{D}$ into a query

$$\underline{Q} = \bigcup_{i=1}^{n} \underline{Q}_i \ ,$$

where each $\underline{Q}_i$ is obtained by Q_i by applying one or more of S-Rule 4.1, S-Rule 4.2, S-Rule 4.3 and S-Rule 4.4 or, when no transformation is possible, $\underline{Q}_i$ is the same as Q_i, and $\mathcal{I}$ (possibly empty) is the set of ICs in $\mathcal{I}_\mathcal{D}$ used to transform $Q_1, \ldots, Q_n$ into $\underline{Q}_1, \ldots, \underline{Q}_n$, respectively $\square$

S-Rule 4.5 asserts that any U_{D_0}*-query* can be transformed by separately transforming the D_0*-queries* which are part of it, whenever is possible, according to the semantic rules introduced in Section 4.1 and Section 4.2.

4.4 A sample application

In this section we provide a sample application of the semantic rules introduced above. We give, in particular, an example of equijoin elimination.
In the hypotheses of Figure 1, consider the following query Q:

> Q: *Return the list of all the computer science reviews edited in 1990 that are requested at the I.E.I. library and are present at I.E.I library.*

(Note that a request may be issued for a review which is not at the I.E.I. library.) According to the RA notation (see Section 2) Q can be formalized as follows:

$$Q = \pi_\mathcal{A}\sigma_P(REQUESTS\_LIST \times REVIEW\_CATALOGUE),$$

where

$$\mathcal{A} = \{REQUESTS\_LIST.review\_title\}$$

$$P = [REQUESTS\_LIST.topic = \text{'computerscience'}]$$
$$\text{and } [REQUESTS\_LIST.demanded\_year = 1990]$$
$$\text{and } [REQUESTS\_LIST.review\_title = REVIEW\_CATALOGUE.review\_title]$$

Then Q requires an equijoin between relation $REQUESTS\_LIST$ and relation $REVIEW\_CATALOGU$
Consider the *simple* IC $I_{s1,1}$ of Fig.3, reported below for convenience

$$I_{s1,1} = \langle REQUESTS\_LIST \times REVIEW\_CATALOGUE,$$
$$[REQUESTS\_LIST.topic = \text{'computerscience'}],$$
$$[REQUESTS\_LIST.review\_title \in \mathcal{K}_{REVIEW\_CATALOGUE.review\_title}]$$
$$\text{or } [REQUESTS\_LIST.demanded\_year < 1984]\rangle.$$

It can be easily verified that Q and $I_{s1,1}$ satisfies all the hypotheses of S-Rule 4.3 for P_1, P_i' and $P_{i,1}''$ defined respectively as follows:

$$P_1 = [REQUEST\_LIST.topic =' computerscience']$$
$$\text{and}[REQUEST\_LIST.demanded\_year = 1990],$$

$$P_i' = [REQUEST\_LIST.topic =' computerscience']$$

and

$$P_{i,1}'' = [REQUEST\_LIST.demanded\_year < 1984].$$

Then by applying S-Rule 4.3 the query Q can be transformed with respect to $I_{s1,1}$ into the following query $\underline{Q}$ defined only on relation $REQUEST\_LIST$:

$$\underline{Q} = \pi_{\mathcal{A}}\sigma_{\underline{P}}(REQUESTS\_LIST)$$

where

$$\underline{P} = [REQUEST\_LIST.topic = \text{'computerscience'}]$$
$$\text{and } [REQUEST\_LIST.demanded\_year = 1990].$$

Thus Q can be evaluated by evaluating $\underline{Q}$, without any expensive computation of the equijoin.

5 Formal Proof of the Correctness of the Semantic Rules

In this section we prove the correctness of the semantic rules introduced in Section 4. The problem of proving the correctness of the semantic rules is twofold: on one side it is necessary to ensure that the transformation the semantic rules produce is well-formulated with respect to the integrity constraints they use; on the other side it is necessary to ensure that the application of the semantic rules provides the expected results. To this purpose remark that it is not strictly necessary that the resulting query and the original query be *equivalent* (refer to Section 2.1), but it is sufficient that they produce the same results just in those database states in which the resulting query must be evaluated on the place of the original query (see below).

To prove the correctness of the semantic rules we shall therefore introduce two distinct correctness criteria in this section, each dealing with one face of the problem. First in Section 5.1 we present such criteria, then in Section 5.2 we show that the semantic rules introduced in Section 4 satisfy both correctness criteria, and thus are correct. The correct application of semantic rules to queries occurring within transactions will be separately discussed in Section 6.

5.1 Correctness criteria

In this section we formally define the criteria that will be used to prove the correctness of the semantic rules. First we introduce the following definition, which formalize the concept of *local* equivalence:

Definition 5.1 Let Q and $\underline{Q}$ be two queries on a database $\mathcal{D}$. Then Q and $\underline{Q}$ are said to be *locally equivalent* in some state S_j of $\mathcal{D}$, written $Q \equiv_{S_j} \underline{Q}$, iff Q and $\underline{Q}$ produce exactly the same set of tuples when they are evaluated in state S_j. □

Thus two queries are *locally* equivalent in some database state if and only if they produce the same set of tuples when they are evaluated in such state. Next property follows directly from the definitions:

Property 5.1 *Let Q and $\underline{Q}$ be queries on a database $\mathcal{D}$. Then $Q \equiv \underline{Q}$ if and only if $Q \equiv_{S_j} \underline{Q}$ for each state S_j of $\mathcal{D}$.* □

Thus whenever two queries on a database are equivalent they are *locally* equivalent in every possible state of the database. The vice versa is also true.

The correctness of the semantic rules is established as follows:

Definition 5.2 (*First Correctness Criterion*) Let $\mathcal{D}$ be a database, and let $\mathcal{I}_{\mathcal{D}}$ be the set of *simple* ICs defined for $\mathcal{D}$. A semantic rule correctly determines a transformation $\underline{Q}$ of a query Q on $\mathcal{D}$ iff for each state S_j of $\mathcal{D}$ in which holds the set of ICs $\mathcal{I}$ with respect to which Q is transformed $(\mathcal{I} \subseteq \mathcal{I}_{\mathcal{D}})$ is $Q \equiv_{S_j} \underline{Q}$ □

Definition 5.3 (*Second Correctness Criterion*) Let $\mathcal{D}$ be a database, and let $\mathcal{I}_{\mathcal{D}}$ be the set of *simple* ICs defined for $\mathcal{D}$. A semantic rule which determines a transformation $\underline{Q}$ of a query Q on $\mathcal{D}$ is correctly applied to Q iff whatever is the state S_j of $\mathcal{D}$ in which Q must be evaluated is $Q \equiv_{S_j} \underline{Q}$ □

Thus a semantic rule correctly determines a transformation $\underline{Q}$ of a query Q on a database $\mathcal{D}$ if and only if Q and $\underline{Q}$ are *locally* equivalent in every state of $\mathcal{D}$ that is consistent with the set of integrity constraints used by the semantic rule to transform Q into $\underline{Q}$. In addition a semantic rule is correctly applied to a query Q if and only if Q and the resulting query $\underline{Q}$ are *locally* equivalent in every state of $\mathcal{D}$ in which Q must be evaluated.

5.2 Correctness proofs

In this section we prove that the semantic rules introduced in Section 4 satisfy both the First and the Second Correctness Criterion defined in the previous section.

First we present two properties that will be used within the correctness proofs and follow directly from the properties of the RA operators.

Property 5.2 *Given two D_0-queries $Q_1 = \pi_{\mathcal{A}}\sigma_{P_1}(E)$ and $Q_2 = \pi_{\mathcal{A}}\sigma_{P_2}(E)$ on a database $\mathcal{D}$, if in same state S_j of $\mathcal{D}$ is $P_1 \Leftrightarrow P_2$ for each tuple t in E then $Q_1 \equiv_{S_j} Q_2$* □

Property 5.3 *Given some couples of D_0-queries $Q_i = \pi_{\mathcal{A}_i}\sigma_{P_i}(E_i)$ and $Q_i' = \pi_{\mathcal{A}_i}\sigma_{P_i'}(E_i)$ on a database $\mathcal{D}$ $(n \geq 1, 1 \leq i \leq n)$, if in same state S_j of $\mathcal{D}$ for each $i \in \{1, \ldots n\}$ is $Q_i \equiv_{S_j} Q_i'$ then $\bigcup_{i=1}^{n} Q_i \equiv_{S_j} \bigcup_{i=1}^{n} Q_i'$* □

Property 5.2 asserts that whenever two D_0*-queries* are defined on the same relation they are *locally* equivalent in a database state if the predicates appearing in their selection operators are equivalent in that state. Property 5.3 asserts that whenever some couples of D_0*-queries* are *locally* equivalent in the same database state then the two U_{D_0}*-queries* obtained by collecting together respectively the first elements of these couples and the second ones are again *locally* equivalent in that state.

Below we prove that each semantic rule introduced in Section 4 satisfies the First Correctness Criterion.

Theorem 5.1 *S-Rule 4.1 correctly determines a transformation* $\underline{Q}$ *of* Q.

<u>Proof</u>
Let $I_i = \langle E_i, P_i', P_i'' \rangle$, $Q = \pi_{\mathcal{A}}\sigma_{P_1 \textit{ and } P_2}(E)$ and $\underline{Q} = \pi_{\mathcal{A}}\sigma_{\underline{P}}(E)$ be defined accordingly to S-Rule 4.1. We must show that whenever I_i holds in some database state S_j then $Q \equiv_{S_j} \underline{Q}$. By Property 5.2 it is sufficient to show that in every database state S_j in which $\{| I_i |\}$ is verified $\underline{P} \Leftrightarrow P_1 \textit{ and } P_2$ for each tuple t in E. First we show that in every database state S_j in which $\{| I_i |\}$ is verified $P_1 \Rightarrow P_i''$ for each t in E.

Claim 1 *In every database state* S_j *in which* $\{| I_i |\}$ *is verified*

$$\forall t \in E \quad P_1(t) \Rightarrow P_i''(t)$$

<u>Proof of the claim</u>
When $\{| I_i |\}$ is verified then by definition

$$\forall t \in E_i \quad P_i'(t) \Rightarrow P_i''(t)$$

But by condition 1 of S-Rule 4.1

$$\mathcal{R}el(E_i) \subseteq \mathcal{R}el(E),$$

therefore

$$\mathcal{A}ttr(E_i) \subseteq \mathcal{A}ttr(E).$$

Also by definition

$$\mathcal{P}attr(P_i') \cup \mathcal{P}attr(P_i'') \subseteq \mathcal{A}ttr(E_i).$$

Then whenever $\{| I_i |\}$ is verified

$$\forall t \in E \quad P_i'(t) \Rightarrow P_i''(t)$$

Moreover by condition 2 of S-Rule 4.1

$$\forall t \in E \quad P_1(t) \Rightarrow P_i'(t)$$

Thus whenever $\{| I_i |\}$ is verified

$$\forall t \in E \quad P_1(t) \Rightarrow P_i''(t) \quad Q.E.D. \ \square$$

The proof of the theorem can now be completed by showing that in every database state S_j in which $\{| I_i |\}$ is verified

$$\forall t \in E \quad \underline{P}(t) \Leftrightarrow P_1(t) \textit{ and } P_2(t).$$

case 1 ($\underline{P} = P_1$)

($\Leftarrow$) It trivially follows from the definitions.

($\Rightarrow$) It is sufficient to show that whenever $\{| I_i |\}$ is verified then

$$\forall t \in E \quad P_1(t) \Rightarrow P_2(t).$$

But when $\{| I_i |\}$ is verified it follows by Claim 1 that

$$\forall t \in E \quad P_1(t) \Rightarrow P_i''(t).$$

Also by the hypotheses it follows that

$$\forall t \in E \quad P_i''(t) \Rightarrow P_2(t).$$

Thus whenever $\{| I_i |\}$ is verified

$$\forall t \in E \quad P_1(t) \Rightarrow P_2(t) \ Q.E.D.$$

case 2 ($\underline{P} = \textit{false}$)
It must be shown that whenever $\{| I_i |\}$ is verified the RA predicate $P_1 \textit{ and } P_2$ is false for all the tuples in E. But whenever $\{| I_i |\}$ is verified it follows by Claim 1 that

$$\forall t \in E \quad P_1(t) \Rightarrow P_i''(t).$$

Also by the hypotheses it follows that

$$\forall t \in E \quad P_i''(t) \Rightarrow \textit{not } P_2(t).$$

Thus whenever $\{| I_i |\}$ is verified

$$\forall t \in E \quad P_1(t) \Rightarrow \textit{not } P_2(t) \ Q.E.D.$$

case 3 ($\underline{P} = P_1 \textit{ and } P_2 \textit{ and } P_i''$)

($\Rightarrow$) It trivially follows from the definitions.

($\Leftarrow$) It is a direct consequence of Claim 1.

This concludes the proof of the theorem. $Q.E.D.$ $\square$

Theorem 5.2 *S-Rule 4.2 correctly determines a transformation* $\underline{Q}$ *of* Q.

<u>Proof</u>
Let $I_i = \langle E_i, P_i', P_i'' \rangle$, $Q = \pi_{\mathcal{A}}\sigma_{P_1 \textit{ and } [R_1.C_1 = R_2.C_2]}(R_1 \times R_2)$ and $\underline{Q} = \pi_{\mathcal{A}}\sigma_{P_1 \textit{ and } [R_1.C_1 = c]}(R_1 \times R_2)$ be defined accordingly to S-Rule 4.2. We must show that whenever I_i holds in some database state S_j then $Q \equiv_{S_j} \underline{Q}$.

Claim 1 *In every database state* S_j *in which* I_i *holds*

$$\forall t \in R_1 \times R_2 \quad P_1(t) \Rightarrow [R_2.C_2 = c](t)$$

<u>Proof of the claim</u>
Analogously as shown in Theorem 5.1 it can be proved that whenever $\{| I_i |\}$ is verified

$$\forall t \in R_1 \times R_2 \quad P_1(t) \Rightarrow P_i''(t).$$

Then because of condition 3 of S-Rule 4.2 it follows that whenever $\{| I_i |\}$ is verified

$$\forall t \in R_1 \times R_2 \quad P_1(t) \Rightarrow [R_2.C_2 = c](t)$$

This concludes the proof of the claim $Q.E.D.$ $\square$

Claim 2 *For each database state S_j in which I_i holds*

$$\pi_{\mathcal{A}}\sigma_{P_1 \text{ and } [R_1.C_1=c]}(R_1 \times R_2) \equiv_{S_j} \pi_{\mathcal{A}}\sigma_{P_1 \text{ and } [R_1.C_1=R_2.C_2]}(R_1 \times R_2)$$

Proof of the claim
By Property 5.2 it is sufficient to prove that in every database state S_j in which $[\![I_i]\!]$ is verified

$$\forall t \in R_1 \times R_2 \;\; P_1(t) \text{ and } [R_1.C_1 = c](t) \Leftrightarrow [R_1.C_1 = R_2.C_2](t) \qquad [\alpha]$$

But observe that $[R_2.C_2 = c]$ implies both

$$[R_1.C_1 = c] \Rightarrow [R_1.C_1 = R_2.C_2]$$

and

$$[R_1.C_1 = R_2.C_2] \Rightarrow [R_1.C_1 = c]$$

Thus by Claim 1 whenever $[\![I_i]\!]$ is verified the equivalence $[\alpha]$ is satisfied $Q.E.D.$ □

To conclude the proof of the theorem it is therefore sufficient to show that for each database state S_j in which $[\![I_i]\!]$ is verified

$$\pi_{\mathcal{A}}\sigma_{P_1 \text{ and } [R_1.C_1=c]}(R_1 \times R_2) \equiv_{S_j} \pi_{\mathcal{A}}\sigma_{P_1 \text{ and } [R_1.C_1=c]}(R_1) \qquad [\beta]$$

But the equivalence $[\beta]$ follows directly from the properties of the RA operators, by the hypotheses (b) and (d) of S-Rule 4.2 and by Property 5.1. This conclude the proof of the theorem $Q.E.D.$ □

Theorem 5.3 *S-Rule 4.3 correctly determines a transformation $\underline{Q}$ of Q.*

Proof
Let $I_i = (E_i, P_i', P_i'')$, $Q = \pi_{\mathcal{A}}\sigma_{P_1 \text{ and } [R_1.C_1=R_2.C_2]}(R_1 \times R_2)$ and $\underline{Q} = \pi_{\mathcal{A}}\sigma_{P_1}(R_1)$ be defined accordingly to S-Rule 4.3. We must show that whenever I_i holds in same database state S_j then $Q \equiv_{S_j} \underline{Q}$.

Claim 1 *In every database state S_j in which I_i holds*

$$\forall t \in R_1 \times R_2 \;\; P_1(t) \Rightarrow [R_1.C_1 \in \mathcal{K}_{R_2.C_2}](t).$$

Proof of the claim
Analogously as shown in Theorem 5.1 it can be proved that whenever $[\![I_i]\!]$ is verified

$$\forall t \in R_1 \times R_2 \;\; P_1(t) \Rightarrow P_i''(t)$$

Then because of condition 3 of S-Rule 4.3 it follows that whenever $[\![I_i]\!]$ is verified

$$\forall t \in R_1 \times R_2 \;\; P_1(t) \Rightarrow [R_1.C_1 \in \mathcal{K}_{R_2.C_2}](t)$$

This concludes the proof of the claim $Q.E.D.$ □

Claim 2 *For each database state S_j in which I_i holds*

$$\pi_{Attr(R_1)}\sigma_{P_1}(R_1 \times R_2) \equiv_{S_j} \pi_{Attr(R_1)}\sigma_{P_1 \text{ and } [R_1.C_1=R_2.C_2]}(R_1 \times R_2)$$

Proof of the claim
It is sufficient to show that whenever $[\![I_i]\!]$ is verified

$$t_1 \in \pi_{Attr(R_1)}\sigma_{P_1}(R_1 \times R_2) \Rightarrow t_1 \in \pi_{Attr(R_1)}\sigma_{P_1 \text{ and } [R_1.C_1=R_2.C_2]}(R_1 \times R_2).$$

The implication

$$t_1 \in \pi_{Attr(R_1)}\sigma_{P_1}(R_1 \times R_2) \Leftarrow t_1 \in \pi_{Attr(R_1)}\sigma_{P_1 \text{ and } [R_1.C_1=R_2.C_2]}(R_1 \times R_2)$$

in fact is trivially true.
Let $\underline{t} \in \sigma_{P_1}(R_1 \times R_2)$. Then $\underline{t} = (t_1 * t_2)$ for some $t_1 \in R_1$ and $t_2 \in R_2$[9]. Note that by construction $t_1 \in \pi_{Attr(R_1)}\sigma_{P_1}(R_1 \times R_2)$. We must show that whenever $[\![I_i]\!]$ is verified also $t_1 \in \pi_{Attr(R_1)}\sigma_{P_1 \text{ and } [R_1.C_1=R_2.C_2]}(R_1 \times R_2)$. But when $[\![I_i]\!]$ is verified by Claim 1 it follows that

$$\forall t \in R_1 \times R_2 \;\; P_1(t) \Rightarrow [R_1.C_1 \in \mathcal{K}_{R_2.C_2}](t)$$

Then there exists $\underline{t}_2$ in R_2 such that $t_1(R_1.C_1) = \underline{t}_2(R_2.C_2)$[10]. Thus whenever $[\![I_i]\!]$ is verified

$$(t_1 * \underline{t}_2) \in \sigma_{P_1 \text{ and } [R_1.C_1=R_2.C_2]}(R_1 \times R_2),$$

and then

$$t_1 \in \pi_{Attr(R_1)}\sigma_{P_1 \text{ and } [R_1.C_1=R_2.C_2]}(R_1 \times R_2) \; Q.E.D. \; \square$$

To complete the proof of the theorem observe that by Claim 2 it follows that

$$\pi_{\mathcal{A}}\pi_{Attr(R_1)}\sigma_{P_1}(R_1 \times R_2) \equiv_{S_j} \pi_{\mathcal{A}}\pi_{Attr(R_1)}\sigma_{P_1 \text{ and } [R_1.C_1=R_2.C_2]}(R_1 \times R_2) \qquad [\gamma]$$

Also, when $[\![I_i]\!]$ is verified, by Rule A.1 reported in Appendix A it follows that because of hypothesis (b) of S-Rule 4.3 and of Property 5.1

$$\pi_{\mathcal{A}}\pi_{Attr(R_1)}\sigma_{P_1 \text{ and } [R_1.C_1=R_2.C_2]}(R_1 \times R_2) \equiv_{S_j} Q$$

and

$$\pi_{\mathcal{A}}\pi_{Attr(R_1)}\sigma_{P_1}(R_1 \times R_2) \equiv_{S_j} \pi_{\mathcal{A}}\sigma_{P_1}(R_1 \times R_2);$$

moreover by Rule A.4 reported in Appendix A it follows that because of hypothesis (d) of S-Rule 5.3 and of Property 5.1

$$\pi_{\mathcal{A}}\sigma_{P_1}(R_1 \times R_2) \equiv_{S_j} \underline{Q}$$

Due to the equivalence $[\gamma]$ this concludes the proof of the theorem $Q.E.D.$ □

Theorem 5.4 *S-Rule 4.4 correctly determines a transformation $\underline{Q}$ of Q.*

Proof
Let I_{i_1}, I_{i_2}, $Q = \pi_{\mathcal{A}}\sigma_{P_1 \text{ and } P_2 \text{ and } [R_1.C_1=R_2.C_2]}(R_1 \times R_2)$ and $\underline{Q} = \pi_{\mathcal{A}}\sigma_{\underline{P}}(R_1)$ be defined accordingly to S-Rule 4.4. We must show that whenever$\{I_{i_1}, I_{i_2}\}$ holds in some database state S_j then $Q \equiv_{S_j} \underline{Q}$. Let $Q' = \pi_{\mathcal{A}}\sigma_{P_1 \text{ and } [R_1.C_1=R_2.C_2]}(R_1 \times R_2)$. Observe that Q can be transformed into Q' by applying S-Rule 4.1 with respect to I_{i_1}. Then in every database state S_j in which I_{i_1} holds, by Theorem 5.1 is $Q \equiv_{S_j} Q'$. Observe that Q', in turn, can be transformed with respect to I_{i_2} into the query $\underline{Q}$ by applying either S-Rule 4.2 or S-Rule 4.3, depending on the conditions satisfied by I_{i_2}. Thus in every database state S_j in which I_{i_2} holds either by Theorem 5.2 (*case 1*) or by Theorem 5.3 (*case 2*) is $Q' \equiv_{S_j} \underline{Q}$. Therefore in every database state S_j in which both I_{i_1} and I_{i_2} hold is $Q \equiv_{S_j} \underline{Q}$ $Q.E.D.$ □

[9]Given two tuples $t_1 \in R_1$ and $t_2 \in R_2$ by $(t_1 * t_2)$ we denote the *concatenation* of t_1 and t_2 (see [Ullm 82]).
[10]We are using a non positional notation for relations here (see, e.g., [Maie 83]), then a tuple is seen as a mapping from attribute names to values.

Theorem 5.5 *S-Rule 4.5 correctly determines a transformation $\underline{Q}$ of Q.*

Proof
It follows as a corollary of Theorem 5.1, Theorem 5.2, Theorem 5.3 and Theorem 5.4, because of Property 5.3 □

The proof of the correctness of the semantic rules introduced in Section 4 with respect to the First Correctness Criterion is therefore concluded. Now we must show that each semantic rule satisfy also the Second Correctness Criterion. But observe that whenever semantic rules are applied to queries that occur outside a transaction the fact that semantic rules satisfy the First Correctness Criterion is sufficient to ensure that they also satisfy the Second Correctness Criterion. In fact semantic integrity is preserved by the database, and thus all the integrity constraints used by a semantic rule to transform the original query surely hold in the database state in which the evaluation of the query is performed. Therefore the following property holds as a corollary of Theorem 5.1, Theorem 5.2, Theorem 5.3, Theorem 5.4 and Theorem 5.5.

Property 5.4 *Let $\mathcal{D}$ be a database, and let $\mathcal{I}_{\mathcal{D}}$ be the set of simple ICs defined for $\mathcal{D}$. Let Q be a query occurring outside a transaction, and let $\underline{Q}$ be a transformation of Q obtained by applying one or more of S-Rule 4.1, S-Rule 4.2, S-Rule 4.3, S-Rule 4.4 and S-Rule 4.5. Then for each state S_j of $\mathcal{D}$*

$$Q \equiv_{S_j} \underline{Q} \ \square$$

Thus for every query Q occurring outside a transaction any semantic rule which determines some transformation $\underline{Q}$ of Q is correctly applied to Q (remark that by Property 5.4 and by Property 5.1 it is also $Q \equiv \underline{Q}$). This concludes the proof of the correctness of the semantic rules introduced in Section 4.

6 Applying Semantic Rules within Transactions

In this section we investigate the problem of applying the semantic rules we have introduced in Section 4 within transactions. We shall focus on *compiled* transactions (transactions from now on).

As we have seen in Section 5, the fact that semantic rules are well-formulated ensures that the obtained query and the original query produce exactly the same results in every database state which is consistent with the integrity constraints defined for the database. However, differently from the case of queries occurring outside transactions, this property is not sufficient to guarantee that semantic rules are correctly applied when they are used for transforming queries that occur within transactions. Transactions, in fact, are allowed to violate semantic integrity during intermediate steps of processing (see Section 3.1). Thus it may happen that the evaluation of a query in a transaction is performed in a database state which is not consistent with all or some of the integrity constraints defined for the database. Therefore restrictions on the usage of semantic rules within transactions must be introduced to avoid to transform a query into another by applying a semantic rule when the semantic rule makes use of integrity constraints that can be invalidated during the execution of the transaction and can be still not valid when the evaluation of the query is required. An example in which errors are produced because no restriction on the usage of semantic rules within transactions is observed is shown in Section 6.1. The problem of correctly applying semantic rules within transactions is discussed in next two sections. First in Section 6.2 the problem is formalized, then in Section 6.3 sufficient conditions for a correct application of semantic rules within transactions are presented.

```
Begin transaction
$new\_pos$: tuple ($name$: string, $position$: string, $salary$: float);
$emp\_name$: string;
while there are still triples to read do
begin
read($new\_pos$);
if ($new\_pos.salary = 0$) then
$\Psi_{(((EMPLOYEE.position, new\_pos.position)), EMPLOYEE.name=new\_pos.name)}(EMPLOYEE)$
else
$\Psi_{(((EMPLOYEE.position, new\_pos.position), (EMPLOYEE.salary, new\_pos.salary)), EMPLOYEE.name=new\_pos.name)}$
$(EMPLOYEE)$;
end

$\pi_{\{EMPLOYEE.name\}}(\sigma_{EMPLOYEE.position='manager' \text{ and } EMPLOYEE.salary \leq 30000}(EMPLOYEE))$

while there are tuples result of the query do
begin
read($emp\_name$);
$\Psi_{(((EMPLOYEE.salary, 31000), ((EMPLOYEE.status, 'T')), EMPLOYEE.name=emp\_name)}(EMPLOYEE)$;

end

End transaction
```

Figure 4: Transaction example

6.1 A sample of incorrect application

Consider the database illustrated in Fig. 1. Consider a transaction which receives as input a set of employee promotions. Each employee promotion consists of a triple of the form

$$\langle name, position, salary \rangle.$$

A triple specifies that the employee, whose name is specified in the field *name*, must receive the position specified in the field *position*. The field *salary* of the triple contains the new salary to be assigned to the employee. The value 0 is used as a special value and denotes that the new salary has not been established yet, and therefore the salary is left unchanged. However, since it may happen that an employee becomes a manager and its new salary is not specified, the transaction is designed so that in this case a minimum default salary of 31,000 is assigned, if the current salary of the employee is lower or equal to 30,000 (recall that the IC I_5 specifies that the salary of a manager must be greater than 30,000). Moreover, the *status* attribute of the tuple (in relation $EMPLOYEE$) concerning the employee is set to T to to denote that the current salary is temporarily set (and later on, the salary may need to be revised). A possible organization for the transaction is informally specified in Fig. 4.

The transaction is organized into three steps. The first updates all employees by modifying the positions of the employees according to the triples received as input. If the salary in a triple is different from 0, the salary of the corresponding employee is updated accordingly; otherwise is left unchanged. The second step retrieves all employees that are manager having a salary lower than 30,000. It may happen that the previous updates have left some tuples of relation $EMPLOYEE$ inconsistent with respect IC I_5 (note a transaction may violate some ICs during its execution, as long as the final state is left consistent). The purpose of this select operation is to determine which tuples are inconsistent and to assign them a minimum default value of 31,000, so that the database is left in a consistent state after the transaction commit. For each tuple retrieved by the select operation, an update is executed.

A problem may however happen if semantic query optimization techniques are applied to this transaction. Suppose that the predicate in the select operation, that is [*position* = *manager and salary* $\leq$ 30000], is transformed by using the IC I_5 on the basis of rule S-Rule 4.1 (cf. Subsection 4.1). The result of the semantic transformation is $\sigma_{false}(EMPLOYEE)$. IC I_5 asserts that all managers have a salary greater than 30,000. Therefore, a query asking for managers having a salary equal or lower than 30,000 must necessarily return an empty result if IC I_5 is used to transform the query. Thus, if the semantic transformation is applied, the select operation will not retrieve the tuples which must be fixed in order for the database to be consistent with respect to the ICs (this will cause the transaction to be backed out when the ICs validation is performed before transaction commit). If instead no semantic transformation is applied the query will retrieve all tuples that are inconsistent, if any, and correctly update them so that the database state, as modified by the transaction, is correct with respect to IC I_5.

Note that the expected correct behavior is the one where the select operations retrieves the inconsistent tuples, that is, the one obtained when no semantic transformation is performed. The reason why the semantic transformation produces an incorrect behavior is that uses an IC which is temporarily not valid. Since transactions must be allowed to violate some ICs during their execution, conditions must be defined preventing semantic transformations using invalid ICs to be applied to queries within a transaction. In this particular example, the select operation should not be transformed by using IC I_5.

6.2 Correct application of semantic rules within transactions: formalization of the problem

In this section we focus on the problem of correctly applying semantic rules within transactions.

Let $\mathcal{I_D}$ be the set of ICs defined for a database $\mathcal{D}$, and let T be a transaction on $\mathcal{D}$ containing some queries. In general transaction T may contain also modify operations, possibly interleaved with queries. Then if $O_1, \ldots, O_f$ ($f \geq 1$) are, in order, all the occurrences of modify operations in T, some queries may occur in T before O_1, some after O_f and some others between each couples of modify operations O_{j-1}, O_j ($1 < j \leq f$). Correspondingly the evaluation of the query in T is distributed on the database states in $\mathcal{S}tate(\mathcal{D},T)$ (see Section 2.5). That is, some queries are evaluated in state $S_{T,0}$, some in state $S_{T,f}$, and some in the other states produced in $\mathcal{D}$ by the intermediate steps of T, as illustrated in Fig. 5. Note that a query Q may occur more than one time in T (before and after some modify operations), and thus Q can be evaluated in different states of $\mathcal{D}$. In the following we shall denote the set of states in which a query Q is evaluated during the execution of T by $\mathcal{S}tate_Q(\mathcal{D},T)$ (clearly $\mathcal{S}tate_Q(\mathcal{D},T) \subseteq \mathcal{S}tate(\mathcal{D},T)$).

Assume that at compile time a query Q in T can be transformed into the query $\underline{Q}$ by applying some semantic rule with respect to the set of ICs $\mathcal{I}$ ($\mathcal{I} \subseteq \mathcal{I_D}$), and let $\underline{T}$ be the transaction obtained by T by replacing Q with $\underline{Q}$ (see Figure 6(a), where we assume that Q occurs two times in T). Then in order to apply the semantic rule to Q at compile time, and thus to execute transaction $\underline{T}$ on the place of T at run time, it must be guaranteed that $\underline{Q}$ produces exactly the same answers as Q would produce (see Fig. 6(b)), as is established by the Second Correctness Criterion introduced in Section 5. For convenience we report below the Second Correctness Criterion specifically formulated for queries that occur within transactions:

Definition 6.1 *(Second Correctness Criterion for Queries occurring within Transactions)* Let $\mathcal{I_D}$ be the set of integrity constraints defined for a database $\mathcal{D}$, and let T be a transaction on

Figure 5: States of the database during the execution of the transaction T

Figure 6: Applying semantic rules within transactions

$\mathcal{D}$. A semantic rule which determines a transformation $\underline{Q}$ of a query Q in T with respect to the set of ICs $\mathcal{I}$ ($\mathcal{I} \subseteq \mathcal{I_D}$) is correctly applied to Q iff

$$\forall S_{T,j} \in \mathcal{S}tate_Q(\mathcal{D},T) \quad Q \equiv_{S_{T,j}} \underline{Q} \ \square \qquad [\delta]$$

It easily follows from the definitions that the First Correctness Criterion (see Section 5) is not sufficient to guarantee that condition [δ] is satisfied by Q and $\underline{Q}$. The states in $\mathcal{S}tate_Q(\mathcal{D},T)$, in fact, may be not consistent with $\mathcal{I}$, except for state $S_{T,0}$, which is the state in which the execution of T starts (refer to Section 3.1). For every other state $S_{T,j}$ in $\mathcal{S}tate_Q(\mathcal{D},T)$ ($j \geq 1$) the validity of $\mathcal{I}$ clearly depends both on the content of $S_{T,0}$ and on the sequence of modify operations $O_1, .., O_j$. Therefore some restrictions on the usage of semantic rules within transactions must be introduced in order to satisfy condition [δ]. We shall discuss this problem in Section 6.3. Before we need some definitions that are introduced in the following.

6.2.1 Involving and Triggering Operations

Let $\mathcal{I_D}$ be the set of integrity constraints defined for a database $\mathcal{D}$. Then the execution of a modify operation on $\mathcal{D}$ may invalidate one or more integrity constraints defined for $\mathcal{D}$. We introduce the following definition:

Definition 6.2 Let $I_i = \langle E_i, P_i', P_i'' \rangle$ be in $\mathcal{I_D}$, and let O be a modify operation on $\mathcal{D}$. Then

(a) O is said to be *involving* with respect to I_i (or, equivalently, I_i is said to be *involved* by O) iff O changes some relation in $\mathcal{R}el(E_i)$.

(b) O is said to be *triggering* with respect to I_i (or, equivalently, I_i is said to be *triggered* by O) iff O is *involving* with respect to I_i and I_i can be violated by the execution of O. $\square$

Clearly if a modify operation O that is *involving* but not *triggering* with respect to some IC I_i is executed in a database state in which I_i holds then I_i holds also in the database state produced by the execution of O. Furthermore, as we have shown in [Bert 88 - Lemma 4.1], if I_i does not hold in the database state in which O is executed then the execution of O may restore the database in a state consistent with I_i.

It is therefore important to recognize the cases in which a modify operation is *involving* but not *triggering* with respect to some IC, because in this case the semantic integrity of the database is preserved. We have investigated this problem in [Bert 88], where we have presented some syntactical conditions for recognizing this kind of operations at compile time. We shall assume that whenever a modify operation O that is *involving* with respect to an IC does not satisfy the conditions presented in [Bert 88] then O is *triggering* with respect to the IC. In the following we shall denote by $\mathcal{T}rig(\mathcal{I_D},T)$ the set of ICs in $\mathcal{I_D}$ that are triggered by some modify operation in T.

6.2.2 Integrity Enforcement Actions

An *Integrity Enforcement Action* (shortly IEA) represents the check of an integrity constraint in some database state. The IEA of the IC I_i in the database S_j will be denoted in the sequel by $EA(I_i, S_j)$, and the outcome of this IEA will be denoted by $Out(EA(I_i, S_j))$. We shall assume that $Out(EA(I_i, S_j))$ is either abort or ok. In particular:

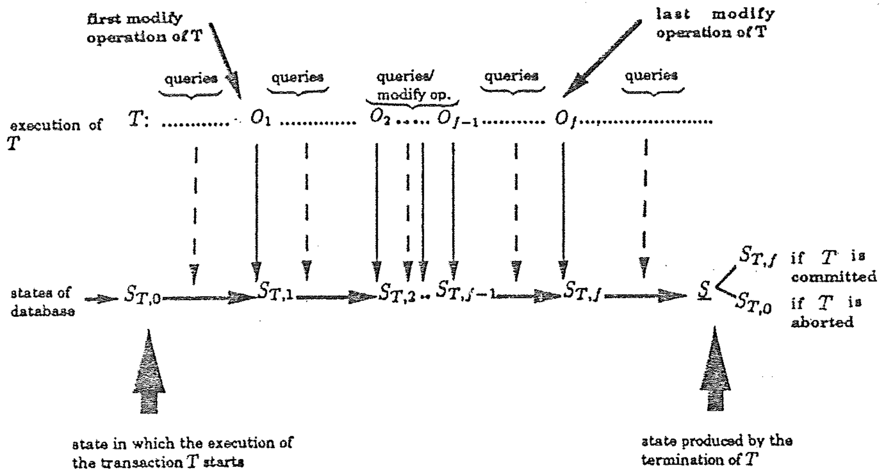
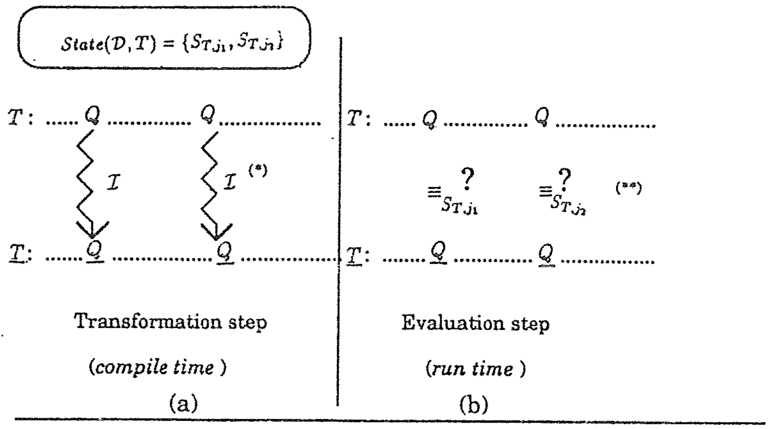

first modify operation of T
last modify operation of T
queries
queries
queries/ modify op.
queries
queries
execution of T
T: .... O_1 .... O_2 ...... O_{f-1} .... O_f ....
states of database
$S_{T,0}$
$S_{T,1}$
$S_{T,2}$
$S_{T,f-1}$
$S_{T,f}$
$\underline{S}$
$S_{T,f}$ if T is committed
$S_{T,0}$ if T is aborted
state in which the execution of the transaction T starts
state produced by the termination of T

Fig.5


$State(\mathcal{D},T) = \{S_{T,j_1}, S_{T,j_2}\}$
T: ...... Q .... Q ....
$\mathcal{I}$
$\mathcal{I}$ (*)
$\underline{T}$: ...... $\underline{Q}$ .... $\underline{Q}$ ....
Transformation step
(compile time)
(a)
T: ...... Q .... Q ....
$\overset{?}{\equiv}_{S_{T,j_1}}$
$\overset{?}{\equiv}_{S_{T,j_2}}$
(**)
$\underline{T}$: ...... $\underline{Q}$ .... $\underline{Q}$ ....
Evaluation step
(run time)
(b)

(*) Q is transformed into $\underline{Q}$ by using the semantic properties od data expressed by the integrity constraints in $\mathcal{I}$

(**) the equivalence is guaranteed only if $\mathcal{I}$ holds

Fig.6

- $out(EA(I_i, S_j))$ is ok if the IC I_i holds in the database state S_j
- $out(EA(I_i, S_j))$ is abort if the IC I_i does not hold in the database state S_j.

We shall also assume in the sequel that no compensation actions are executed in the case of integrity failure. Therefore the enforcement of a given IC will be independent of the enforcement of all other ICs.

6.2.3 Enforcing States and Inherently Enforcing States

The states produced on a database $\mathcal{D}$ by the execution of a transaction can be characterized on the basis of their properties with respect to the set of *simple* ICs $\mathcal{I}_\mathcal{D}$ defined for $\mathcal{D}$, as follows:

Definition 6.3 Let T be a transaction on a database $\mathcal{D}$, and let O_j be a modify operation in T. Let I_i be an IC in $Trig(\mathcal{I}_\mathcal{D}, T)$, and let $S_{T,j}$ be in $State(\mathcal{D}, T)$. Then:

(a) $S_{T,j}$ is said to be *enforcing* with respect to I_i iff either $j = f$ or $Out(EA(I_i, S_{T,j})) = abort$ implies $Out(EA(I_i, S_{T,f})) = abort$

(b) $S_{T,j}$ is said to be *inherently enforcing* with respect to I_i iff whatever is the state $S_{T,0}$ in which the execution of T starts $S_{T,j}$ is *enforcing* with respect to I_i □

Then a state $S_{T,j}$ in $State(T, \mathcal{I}_\mathcal{D})$ is *enforcing* with respect to an IC I_i in $Trig(\mathcal{I}_\mathcal{D}, T)$ if and only if either $S_{T,j}$ is the final state for T (i.e., $j = f$), or whenever I_i is violated in state $S_{T,j}$ then I_i is also violated in state $S_{T,f}$. The state $S_{T,j}$ is *inherently enforcing* with respect to I_i if it is *enforcing* with respect to I_i whatever is the initial state $S_{T,0}$ for T (that is, even if a different initial state $S_{T,0}$ may lead to a different state $S_{T,j}$, $S_{T,j}$ is still *enforcing* with respect to I_i). In other words, *inherently enforcing* states are those states that can be recognized for a transaction as *enforcing* states with respect to some integrity constraint without examining their content. They can be therefore recognized at compile time, on the basis of the form of the transaction and the order among the modify operations included in it, without waiting for the transaction execution. Syntactical conditions for detecting the *inherently enforcing* states with respect to a triggered integrity constraint for a transaction have been presented in [Bert 88].

6.2.4 Integrity Enforcement Schedule and Minimal Integrity Enforcement Schedule

An *Integrity Enforcement Schedule* (shortly, IES) consists of a set of IEAs (see Section 6.2.2), and defines an order for checking the integrity constraints during the execution of a transaction. We have introduced this concept in [Bert 88], to provide a solution to the problems related to the transaction management in the presence of integrity constraints. Delaying the enforcement of integrity constraints to the end of a transaction, in fact, even if theoretically correct, can be expensive, because it may cause a complete rollingback of a transaction. On the other hand, because transactions are allowed to violate semantic integrity during the intermediate steps of processing, it must be guaranteed that after the completion of a transaction the database is again in a consistent state with respect to the integrity constraints, otherwise the transaction must be aborted. The solution proposed by an IES consists in the enforcement of integrity constraints during the execution of a transaction, without waiting for the transaction termination. In the remainder of the paper we shall use the advantages offered by the IESs to individuate conditions allowing a correct application of semantic rules within transactions.

In the sequel we shall denote an IES for a transaction T with respect to a set of ICs $\mathcal{I}$ by $ES(\mathcal{I}, T)$, and its outcome will be denoted by $out(ES(\mathcal{I}, T))$. We shall assume that $out(ES(\mathcal{I}, T))$ is either ok or abort. In particular:

- $out(ES(\mathcal{I}, T))$ is abort
  if $out(EA(I_i, S_j))$ is *abort* for at least one $EA(I_i, S_j)$ in $ES(\mathcal{I}, T)$
- $out(ES(\mathcal{I}, T))$ is ok
  if $out(EA(I_i, S_j))$ is *ok* for every $EA(I_i, S_j)$ in $ES(\mathcal{I}, T)$

Thus when all the IEAs defined for an IES produce as outcome *ok* (that is, when all the integrity constraints enforced by the *Integrity Enforcement Actions* in the IES hold in the pointed out database states) the outcome of the IES is *ok*, otherwise the outcome of the IES is *abort* and the transaction must be aborted.

Refer to [Bert 88 - Definition 4.1] for a formal definition of the correctness of an IES. Briefly, an IES for a transaction is *correct* if and only if its outcome is the same as the outcome of the IES where all the integrity constraints are checked at the end of the transaction execution. Remark that under the hypothesis that no compensation actions be executed in the case of integrity checking failure, the problem of finding a correct IES for a transaction can be restated as the problem of finding the most appropriate database states for the enforcement of each *triggered* integrity constraint.

A special case of IES, called *Minimal Integrity Enforcement Schedule* (shortly, MIES) provides a check of each integrity constraint triggered by the transaction in the first state that is *inherently enforcing* with respect to that integrity constraint, with the obvious advantage of allowing the detection of semantic integrity failure as early as possible during the transaction execution. In fact as soon as an *Integrity Enforcing Action* of an integrity constraint is performed in a state that is *inherently enforcing* with respect to the integrity constraint and returns as outcome *abort* then the checking of the integrity constraint in the final state produced by the execution of the transaction (i.e., for transaction T, state $S_{T,f}$) still returns as outcome *abort* (refer to Definition 6.3). Then by definition also the outcome of the MIES is *abort*, and thus the transaction must be aborted. Therefore the execution of the transaction can be halted in correspondence of the modify operation that produces the *inherently enforcing* state in which the *Integrity Enforcing Action* returning *abort* is performed, without waiting for the transaction termination: all the updates made on the database by carrying on the subsequent part of transaction, in fact, should be erased from the database because the transaction is aborted. Thus their accomplishment can be avoided, with the result of reducing the number of unuseful updates on the database and the related expensive rollingback operations required when the transaction terminates. Clearly the detection of a transaction failure in the first *inherently enforcing* state with respect to some integrity constraint minimizes the number of rollingback operations to be performed on a database.

In [Bert 88] we have proposed an algorithm which returns a MIES with respect to a set of *simple* integrity constraints for a transaction consisting of two distinct phases: a read-phase, where queries on the database are performed, and a modify-phase, which follows the read-phase, where modify operations are carried out on the database. The construction of the MIES (refer to [Bert 88-Algorithm 5.3]) is realized on the basis of the syntactical properties of the transaction and on the form of the *simple* integrity constraints defined for the database, without examining the database content, and thus it can be entirely accomplished at compile time. The MIES obtained by applying such algorithm is a correct *Integrity Enforcement Schedule* for the transaction (see [Bert 88 - Theorem 5.3]), and contains only *Integrity Enforcing Actions* to be executed in same state that is *inherently enforcing* with respect to an integrity constraint triggered by the transaction.

In the following section we shall extend the use of a MIES to the case of transactions containing any arbitrary interleaved sequence of read and modify operations, with the goal of individuating conditions ensuring a correct usage of semantic rules within transactions.

Figure 7: *Inherently enforcing* states and *involving* and *triggering* operations with respect to the integrity constraint I_i

6.3 Sufficient conditions for a correct application of semantic rules within transactions

In this section we present sufficient conditions for correctly applying semantic rules within transaction. Next two properties follow as a direct consequence of the First and the Second Correctness Criterion:

Property 6.1 *If a transaction T contains only queries then every semantic rule can be correctly applied within T.* □

Property 6.2 *If a transaction T contains both queries and modify operations, and queries precede all the modify operations, then every semantic rule can be correctly applied within T.* □

Thus every semantic rule can be correctly applied to transform a query occurring within a transaction if the transaction does not contain any modify operations or if the modify operations included in the transaction follow all the transaction queries.

In the remainder of this section we consider the general case of transactions having an arbitrary form. Assume that T be a transaction containing any arbitrarily interleaved sequence of queries and modify operations. Let T_M be the transaction obtained by T by eliminating all the queries in T, i.e. let T_M be the transaction containing, in order, all the modify operations that are in T and no other operations. Clearly $Trig(\mathcal{I}_\mathcal{D}, T_M)$ is the same as $Trig(\mathcal{I}_\mathcal{D}, T)$. For each $I_i \in Trig(\mathcal{I}_\mathcal{D}, T_M)$ let $ES_i(\{I_i\}, T_M)$ be the MIES for the transaction T_M produced by applying Algorithm 5.3 in [Bert 88] (refer to Section 6.2.4) with respect to the set of ICs $\{I_i\}$. Let $State(ES_i(\{I_i\}, T_M)) = \{S_{T,h_{i,1}}, ..., S_{T,h_{i,n_i}}\}$ $(1 \leq h_{i,1} \leq ... \leq h_{i,n_i} \leq f)$ be the set of *inherently enforcing* states in which I_i must be tested during the execution of T_M according to the MIES ES_i. As illustrated in Fig. 7, every operation O_j in T_M with $j > h_{i,n_i}$ by construction is not *involving* or *involving* but not *triggering* with respect to I_i [11]. Remark also that if O_{j_0} is the first modify operations in T_M that is triggering with respect to I_i then by construction $j_0 \leq h_{i,1}$. Finally observe that for each $h \in \{h_{i,1}, ..., h_{i,n_i}\}$ the execution of T_M is carried on after the execution of the operation O_h only if the database state $S_{T,h}$ is consistent with the IC I_i, otherwise the transaction is immediately aborted (see Section 6.2.4). Therefore if the transaction T_M is not aborted in the state $S_{T,h}$ then the integrity constraint I_i surely holds in all the subsequent states produced by the modify operations following O_h in T_M, until the first modify operation after O_h that is *triggering* with respect to I_i, if any, is not encountered.

Since the MIES ES_i can be defined at compile time, and conditions that are purely syntactical are available for detecting if a modify operation is *involving* but not *triggering* with respect to a given integrity constraint (refer to Section 6.2.1), it is therefore possible to determine at compile time some intervals of states in $State(\mathcal{D}, T_M)$ (see Fig. 8) having the property that either the integrity constraint I_i holds in each state of the interval during the execution of T_M or the transaction T_M is aborted exactly in correspondence of the first state of the interval reached by the database (which is one of the states in $\{h_{i,1}, ..., h_{i,n_i}\}$).

If we consider transaction T on the place of transaction T_M, we can now recognize that the evaluation of some queries in T occur inside the intervals of states emphasized in Fig. 8 (refer to Fig. 5), while the evaluation of some others queries may occur outside such intervals. Clearly

[11] For details concerning this property refer to [Bert 88].

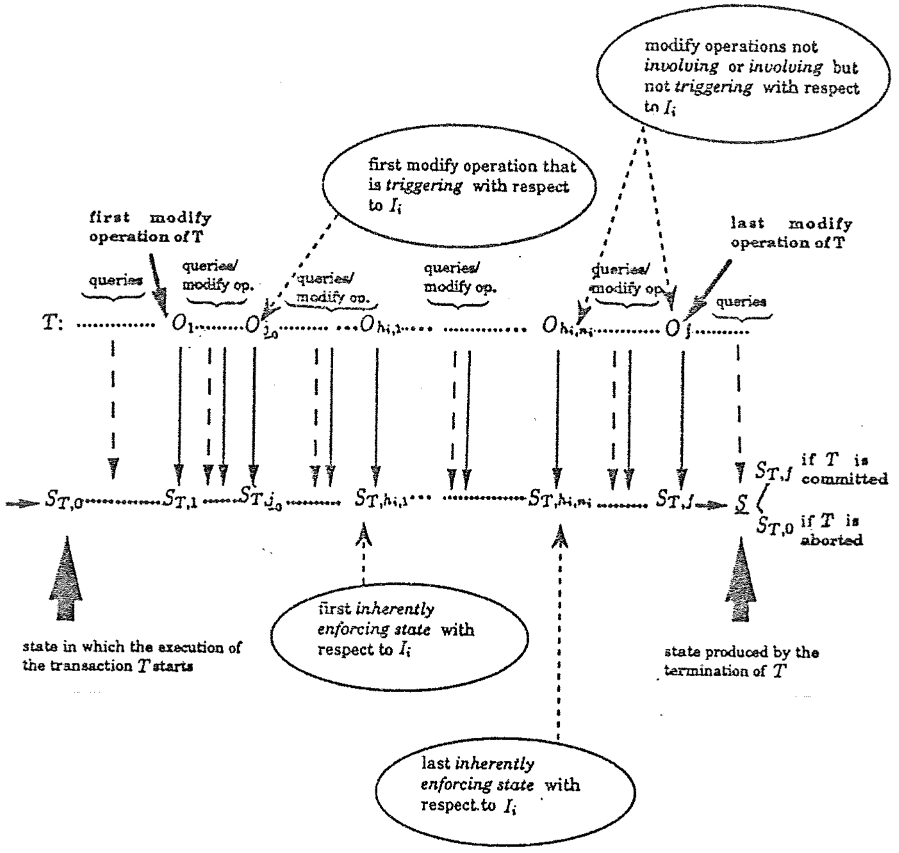

modify operations not *involving* or *involving* but not *triggering* with respect to I_i
first modify operation that is *triggering* with respect to I_i
first modify operation of T
last modify operation of T
queries
queries/ modify op.
queries/ modify op.
queries/ modify op.
queries/ modify op.
queries
T: O_1 O_{j_0} $O_{h_{i,1}}$ $O_{h_{i,n_i}}$ O_f
$S_{T,0}$ $S_{T,1}$ S_{T,j_0} $S_{T,h_{i,1}}$ $S_{T,h_{i,n_i}}$ $S_{T,f}$ S
$S_{T,f}$ if T is committed
$S_{T,0}$ if T is aborted
state in which the execution of the transaction T starts
first *inherently enforcing state* with respect to I_i
state produced by the termination of T
last *inherently enforcing state* with respect to I_i

Fig.7

Figure 8: Special intervals of database states definable at compile time with respect to the integrity constraint I_i

only for the queries whose evaluation is performed in states that belong to the intervals it is guaranteed that the integrity constraint I_i holds during their evaluation (in fact either I_i holds or the execution of T is halted before the evaluation of the query: see above). No assumption concerning the consistency at run time of the database with respect to I_i can instead be made at compile time for those queries that can be evaluated outside these intervals. Therefore to correctly apply at compile time a semantic rule which makes use of the integrity constraint I_i to a query Q in T, it is sufficient to prevent the application of the semantic rule to Q in all the cases in which there is some state in $State_Q(\mathcal{D}, T)$ which is not included within the intervals reported in Fig. 8. This condition is formally defined by the following property.

Property 6.3 *Let T be a transaction on a database $\mathcal{D}$, let $\mathcal{I}_\mathcal{D}$ be the set of simple ICs defined for $\mathcal{D}$, and let $\mathcal{I} \subseteq \mathcal{I}_\mathcal{D}$. For each $I_i \in Trig(\mathcal{I}, T)$, let $ES(\{I_i\}, T)$ be the Minimal Enforcement Schedule derived by applying Algorithm 5.3 in [Bert 88] with respect to I_i, and let $State(ES_i(\{I_i\}, T_M)) = \{S_{T,h_{i,1}}, ..., S_{T,h_{i,n_i}}\}$ $(1 \leq h_{i,1} \leq ... \leq h_{i,n_i} \leq f)$. Consider the following subsets of integers, each one containing a sequence of consecutive numbers representing subsequent indexes of modify operations included in the transaction T (and the corresponding states generated on the database):*

$H_{i,0} = \{0, ..., \underline{j}_0 - 1\}$
$H_{i,1} = \{h_{i,1}, ..., \underline{j}_{h_{i,1}} - 1\}$
....
$H_{i,n_i-1} = \{h_{i,n_i-1}, ..., \underline{j}_{h_{i,n_i-1}} - 1\}$
$H_{i,n_i} = \{h_{i,n_i}, ..., f\}$,

where $\underline{j}_0, \underline{j}_{h_{i,1}}, ..., \underline{j}_{h_{i,n_i-1}}$ are such that $O_{\underline{j}_0}$ is the first modify operation in T that is triggering with respect to I_i (by construction $\underline{j}_0 \leq h_{i,1}$), and for each $s \in \{h_{i,1}, ..., h_{i,n_i-1}\}$ $O_{\underline{j}_{h_{i,s}}}$ is the first modify operation following $O_{h_{i,s}}$ in T that is triggering with respect to I_i (by construction $\underline{j}_{h_{i,s}} \leq h_{i,s+1}$).

Then a semantic rules which transforms a query Q with respect to I_i is correctly applied to Q if:

$$S_Q(\mathcal{D}, T) \subseteq H_{i,0} \cup H_{i,1} ... \cup H_{i,n_i} \quad \square$$

Thus a semantic rule is correctly applied to transform a query Q occurring in a transaction T with respect to the integrity constraint I_i if Q must be evaluated during the execution of T in some database state S_j which either precedes the state $S_{\underline{j}_0}$ produced by the first modify operation that is *triggering* with respect to I_i in T, or is *inherently enforcing* with respect to I_i and is pointed out for the enforcement of I_i in the MIES defined for T_M, or it follows an *inherently enforcing* state for I_i but precedes the subsequent state produced by the next modify operation that is *triggering* with respect to I_i, if any.

When more than one integrity constraint is involved in the application of a semantic rule to a query occurring in a transaction, in order to correctly apply the semantic rule to the query it is sufficient to ensure that the database states in which the query must be evaluated during the execution of the transaction belong to the intersection of the intervals of states that can be identified for each involved integrity constraint as described above. Formally:

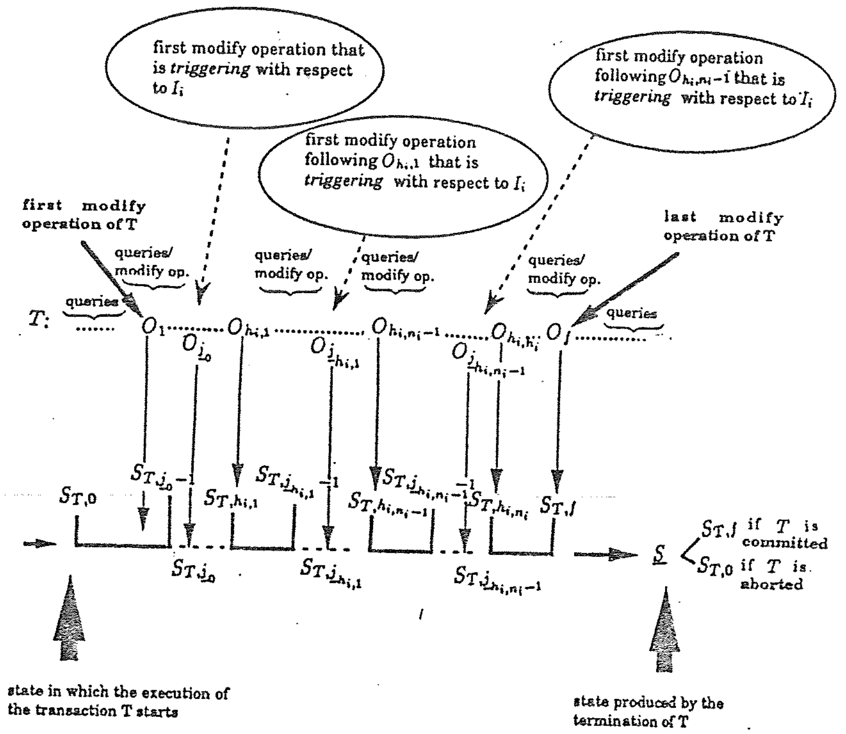

first modify operation that is *triggering* with respect to I_i
first modify operation following $O_{h_{i,1}}$ that is *triggering* with respect to I_i
first modify operation following $O_{h_{i,n_i-1}}$ that is *triggering* with respect to I_i
first modify operation of T
last modify operation of T
queries
queries/ modify op.
queries/ modify op.
queries/ modify op.
queries/ modify op.
queries
T:
O_1 O_{j_0} $O_{h_{i,1}}$ $O_{j_{h_{i,1}}}$ $O_{h_{i,n_i-1}}$ $O_{j_{h_{i,n_i-1}}}$ $O_{h_{i,n_i}}$ O_f
$S_{T,0}$ S_{T,j_0-1} S_{T,j_0} $S_{T,h_{i,1}}$ $S_{T,j_{h_{i,1}}-1}$ $S_{T,j_{h_{i,1}}}$ $S_{T,h_{i,n_i-1}}$ $S_{T,j_{h_{i,n_i-1}}-1}$ $S_{T,j_{h_{i,n_i-1}}}$ $S_{T,h_{i,n_i}}$ $S_{T,f}$
S $S_{T,f}$ if T is committed
$S_{T,0}$ if T is aborted
state in which the execution of the transaction T starts
state produced by the termination of T

Fig.8

Property 6.4 *Let $H_{i,1}, \ldots, H_{i,n_i}$ be defined as in Property 6.3. Then a semantic rules which transforms a query Q with respect to a set of integrity constraints $\mathcal{I}$ is correctly applied to Q if:*

$$\mathcal{S}_Q(\mathcal{D},\mathcal{T}) \subseteq \bigcap_{i \in \mathcal{I}} \left(\bigcup_{j=0}^{n_i} H_{i,j} \right) \square$$

7 Concluding Remarks

In this paper we have discussed the problem of query transformation by using semantic properties of data for queries expressed in terms of Relational Algebra (RA). The particular semantic properties of data used to transform the queries are those expressed by the integrity constraints expressed for the database. The class of queries for which the transformations are provided is a special class of the RA queries. In [Bert 86] it has been demonstrated that any arbitrarily complex query not containing the difference operator is equivalent to the union of this special kind of queries.

First we have given a formal definition of integrity constraints for the RA. An integrity constraint I is defined by a 3-tuple $\langle E, P', P'' \rangle$ expressing the fact that the tuples obtained by applying the predicate P' to the set derived from the expression E satisfy the predicate P''. In other words $P' \Rightarrow P''$ for tuples in the set defined by E.

We have also introduced the concept of *simple* integrity constraints, which are integrity constraints satisfying some restrictions on the structure of the expression E and on the form of the predicates P' and P''. In particular, E is a base relation or a Cartesian product of base relations, P' is a *simple predicate* or a conjunction of *simple predicates*, and P'' is a *simple predicate* or a disjunction of *simple predicates*. As shown in the paper, no loss of generality follows on assuming only *simple* integrity constraints.

Then we have presented query transformation rules which use semantic properties of data expressed by *simple* integrity constraints to transform a given query into an equivalent one. In particular, the first rule allow the predicate of a query to be transformed with the elimination or modification of some of the component predicates, as well as with the addition of new predicates deriving from the semantic properties of data. The other three rules allow the number of relations in a query to be modified, without changing the query semantics, which leads to join elimination. Finally the last rule allow to transform complex queries which also include the *union* RA operator. A formal proof of the correctness of such rules is presented in the paper.

The importance of such rules is that they may affect the time required to process a query. In particular join elimination, clustering index introduction, empty query test become possible, depending on the properties of data. As a result, a query optimization can be gained. Then the transformations defined in the paper become useful in the process of query optimization and can be added to the semantic query optimization techniques.

We have also addressed the problem of correctly applying the semantic rules introduced in this paper within transactions. At best of our knowledge this problem has not been addressed elsewhere in the literature, but it has not a trivial solution. The case of compiled transaction has been investigated, where the query optimization cost can be amortized over several transaction executions. Conditions are presented in the paper which guarantee a correct application of the semantic rules to transform the queries occurring within transactions.

The work presented in this paper can be extended in several ways. First, other semantic rules may be defined. In particular, some generalization of the rules presented in Section 4 can be given. Second, the convenience of the various transformations in terms of query processing cost will be evaluated. Third heuristics will be defined to guide the process of query transformation.

References

[Bert 84] Bertino E. and Apuzzo D., "Integrity Aspects in Database Management Systems", in *Proc. International Trends and Application Conference*, Gaithersburg, MD, May 23-24, 1984.

[Bert 86a] Bertino E. and Musto D., "Transforming Queries into Canonical Forms'", I.E.I. tech. Rep. N. B4-64, 1986.

[Bert 86b] Bertino E. and Musto D., "Semantic Query Transformation", in *Proc. INFORSID Conference*, Abbaye de Fontevraud (France), may 27-30, 1986.

[Bert 88] Bertino E. and Musto D., "Correctness of Semantic Integrity Checking in Database Management Systems", *Acta Informatica*, Vol. 26, 1988, pp. 25-57.

[Brod 78] Brodie M., "Specification and Verification of Database Semantic Integrity", Ph.D. Thesis, Toronto University, March 1978.

[Ceri 83] Ceri S., and Pelagatti G., "Correctness of Query Execution Strategies in Distributed Databases", *ACM Transactions on Database Systems*, Vol. 8, N. 4, Dec. 1983, pp. 577-607.

[Ceri 90] Ceeri S., Gottlob G., Tanca L., "What You Always Wanted to Know about Datalog (and Never Dared to Ask)", *IEEE trans. on Knowledge and Data Engineering*, Vol. 1, No. 1, March 1989, pp.146-166.

[Chak 86] Chakravarty U.S., Fishman D.H., and Minker J., "Semantic Query Optimization in Expert Systems and Database Systems", *Expert Database Systems*, L. Kerschberg (ed.), Benjamin/Cummings Publ., 1986, pp.659-674.

[Codd 70] Codd E.F., "A Relational Model for Large Shared Data Banks", *Comm. ACM*, Vol. 13, N. 6, June 1970, pp. 377-387.

[Daya 81] Dayal U., Goodman N., and Katz R.H., "An Extended Relational Algebra with Control over Duplicate Elimination", Computer Corporation of America, Oct. 1981.

[Date 83] Date C.J., *An Introduction to Database Systems*, Vol.II, Reading: Addison-Wesley Ed. 1983.

[Eswa 75] Eswaran K., Chamberlin D., "Functional Specification of a Subsystem for Data Base Integrity", in *Proc. of the First International Conference on Very Large Data Bases*, Framingham, USA, September 1975.

[Furt 77] Furtado A.L., and Kerschberg L., "An Algebra of Quotient Relations", in *Proc. ACM SIGMOD International Conference*, August 1977, pp. 1-7.

[Gray 82] Gray J.N., McJones P., Blasgen M., Lindsay B., Lorie R., Price T., Putzolu F., Traiger I.L., "The Recovery Manager of System *R* Database Manager", *ACM Comp. Surveys*, Vol. 13, pp. 223-241, 1982.

[Hamm 80] Hammer M., and Zdonik S., "Knowledge-based Query Processing", in *Proc. of the 5th International Conference on Very Large Data Bases*, 1980, pp. 137-147.

[Jark 84] Jarke M., Koch J., "Query Optimization in Database Systems", *ACM Computing Surveys*, Vol. 16, N. 2, June 1984, pp. 110-152.

[King 81] King J., "Quist: a System for Semantic Query Optimization in Relational Databases", in *Proc. of the 6th International Conference on Very Large Data Bases*, 1981, pp. 510-517.

[Maie 83] Maier, D., "The theory of Relational Databases", Computer Science Press, 1983.

[Nico 78] Nicolas J.M., "Logic for Improving Integrity Checking in Relational Databases", *Acta Informatica*, Vol. 18, July 1978.

[Quer 85] "Query Processing in Database Systems", W. Kim, D.S. Reiner, and D, Batory (eds.), Springer-Verlag, 1985.

[Shea 85] Sheard T., Stemple D., "Coping with Complexity in Automated Reasoning about Database Systems", in *Proc. of the 10th International Conference on Very Large Data Bases*, Stockholm (Sweden), August 21-23, 1985, pp. 426-435.

[Shen 87] Shenoy S.T., and Ozsoyoglu Z.M., "A System for Semantic Query Optimizer", in *Proc. ACM-SIGMOD Conference*, San Francisco (Calif.), May 27-29, 1987, pp.181-195.

[Shep 84] Shepherd A., and Kershberg L., "PRISM: a Knowledge Based Approach for Semantic Integrity Specification and Enforcement in Database Systems", in *Proc. ACM SIGMOD Conference*, Boston, Mass., June 18-21, 1984, pp. 307-315.

[Ston 75] Stonebraker M., "Implementation of Integrity Constraints and Views by Query Modification", in *Proc. ACM SIGMOD Conference*, 1975.

[Ullm 82] Ullman J.D., "Database Systems", (Second Edition), Computer Science Press, 1982.

[Wilk 90] K. Wilkinson, P. Lyngbaek, and W. Hasan, "The IRIS Architecture and Implementation", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 2, No. 1, March 1990, pp. 63-75.

[Yu 84] Yu C.T., and Chang C.C., "Distributed Query Processing", *ACM Computing Surveys*, Vol. 16, N. 4, Dec. 1984.

A Equivalence Rules for RA Expressions

In this appendix we present some equivalence rules for RA expressions that are based on syntactical properties of RA operators. For the proof of the correctness of these rules refer to [Ullm 82].

Rule A.1 *Cascade of projections:*
$\pi_{\mathcal{A}}(\pi_{\mathcal{A}'}(E)) \equiv \pi_{\mathcal{A}}(E) \quad$ if $\mathcal{A} \subseteq \mathcal{A}'$

Rule A.2 *Cascade of selections:*
$\sigma_{P_1}(\sigma_{P_2}(E)) \equiv \sigma_{P_1 and P_2}(E)$

Rule A.3 *Union of selections:*
$\sigma_{P_1}(E) \cup \sigma_{P_2}(E) \equiv \sigma_{P_1 or P_2}(E)$

Rule A.4 *Commuting selection and projection:*
$\pi_{\mathcal{A}}(\sigma_P(E)) \equiv \sigma_P(\pi_{\mathcal{A}}(E)) \quad$ if $\mathcal{P}attr(P) \subseteq \mathcal{A}$
$\pi_{\mathcal{A}}(\sigma_P(E)) \equiv \pi_{\mathcal{A}}\sigma_P(\pi_{\mathcal{A}'}(E)) \quad$ if $\mathcal{A}' = \mathcal{A} \cup \mathcal{P}attr(P)$

Rule A.5 *Commuting selection and union:*
$\sigma_P(E_1 \cup E_2) \equiv \sigma_P(E_1) \cup \sigma_P(E_2)$.

B Proof of Theorem 3.1

In this appendix we present the proof of Theorem 3.1. Before we introduce some lemmas that emphasize properties of integrity constraints that will be used to prove Theorem 3.1. The equivalence rules for RA expressions referred within these lemmas are those listed in Appendix A.

Lemma B. 1 (*"$\bigcup$" elimination*)
Let $\mathcal{I} = \{I\} \cup \mathcal{I}_1$, where $I = \langle E_1 \cup E_2, P', P'' \rangle$. Let $\mathcal{I}' = \{I_1, I_2\} \cup \mathcal{I}_1$, where $I_1 = \langle E_1, P', P'' \rangle$, and $I_2 = \langle E_2, P', P'' \rangle$. Then $\mathcal{I} \equiv \mathcal{I}'$.

Proof
It is sufficient to prove that $\{I\} \equiv \{I_1, I_2\}$. Let $E' = \sigma_{P'}(E_1 \cup E_2)$, $E_1' = \sigma_{P'}(E_1)$, and $E_2' = \sigma_{P'}(E_2)$. By Rule A.5 is $E' \equiv E_1' \cup E_2'$. Then $[\| I \|] = \forall t \in E' \; P''(t) = \forall t \in (E_1' \cup E_2') \; P''(t)$, whereas $[\| \{I_1, I_2\} \|] = [\forall t \in E_1' \; P''(t)]$ *and* $[\forall t \in E_2' \; P''(t)]$. Therefore $[\| I \|] \Leftrightarrow [\| \{I_1, I_2\} \|]$. Thus $\{I\} \equiv \{I_1, I_2\}$, and then $\mathcal{I} \equiv \mathcal{I}'$ *Q.E.D.*$\square$

Lemma B. 2 (*"π" elimination*)
Let $\mathcal{I} = \{I\} \cup \mathcal{I}_1$, where $I = \langle \pi_{\mathcal{A}}(E), P', P'' \rangle$. Let $\mathcal{I}' = \{I_1\} \cup \mathcal{I}_1$, where $I_1 = \langle E, P', P'' \rangle$. Then $\mathcal{I} \equiv \mathcal{I}'$.

Proof
It is sufficient to prove that $I \equiv I_1$. Let $E' = \sigma_{P'}(\pi_{\mathcal{A}}(E))$ and let $E_1' = \sigma_{P'}(E)$. By Rule A.4 is $E' \equiv \pi_{\mathcal{A}}(E_1')$, being $\mathcal{P}attr(P') \subseteq \mathcal{A}$ by construction [12]. Then $[\| I \|] = \forall t \in E' \; P''(t) = \forall t \in \pi_{\mathcal{A}}(E_1') \; P''(t)$, whereas $[\| I_1 \|] = \forall t \in E_1' \; P''(t)$. Therefore $[\| I_1 \|] \Leftrightarrow [\| I \|]$, being $\mathcal{P}attr(P'') \subseteq \mathcal{A}$ by construction [13]. Thus $I \equiv I_1$, and then $\mathcal{I} \equiv \mathcal{I}'$ *Q.E.D.*$\square$

[12] This follows from Rule 2 of Definition 2.1 and from condition (c) of Definition 3.1.
[13] It again follows from Rule 2 of Definition 2.1 and from condition (c) of Definition 3.1.

Lemma B. 3 (*"σ" elimination*)
Let $\mathcal{I} = \{I\} \cup \mathcal{I}_1$, where $I = \langle \sigma_P(E), P', P'' \rangle$. Let $\mathcal{I}' = \{I_1\} \cup \mathcal{I}_1$, where $I_1 = \langle E, P$ and $P', P'' \rangle$. Then $\mathcal{I} \equiv \mathcal{I}'$.

Proof
It is sufficient to prove $I \equiv I_1$. Let $E' = \sigma_{P'}(\sigma_P(E))$ and $E'_1 = \sigma_{P \text{ and } P'}(E)$. By Rule A.2 is $E' \equiv E'_1$. Therefore $[\| I_1 \|] \Leftrightarrow [\| I \|]$. Thus $I \equiv I_1$, and then $\mathcal{I} \equiv \mathcal{I}'$ *Q.E.D.* □

Lemma B. 4 (*predicate simplification for disjunction*)
Let $\mathcal{I} = \{I\} \cup \mathcal{I}_1$, where $I = \langle E, P', P'' \rangle$, and P' has the form $[P'_1$ or $P'_2]$ for some RA predicates P'_1 and P'_2. Let $\mathcal{I}' = \{I_1, I_2\} \cup \mathcal{I}_1$, where $I_1 = \langle E, P'_1, P'' \rangle$ and $I_2 = \langle E, P'_2, P'' \rangle$. Then $\mathcal{I} \equiv \mathcal{I}'$.

Proof
It is sufficient to prove $\{I\} \equiv \{I_1, I_2\}$. Let $E' = \sigma_{P'}(E)$, $E'_1 = \sigma_{P'_1}(E)$, and $E'_2 = \sigma_{P'_2}(E)$. By Rule A.3 is $E' \equiv E'_1 \cup E'_2$. Then $[\| I \|] = \forall t \in E' \; P''(t) = \forall t \in (E'_1 \cup E'_2) \; P''(t)$, whereas $[\| \{I_1, I_2\} \|] = [\forall t \in E'_1 \;\; P''(t)]$ *and* $[\forall t \in E'_2 \; P''(t)]$. Therefore $[\| I \|] \Leftrightarrow [\| \{I_1, I_2\} \|]$. Thus $\{I\} \equiv \{I_1, I_2\}$, and then $\mathcal{I} \equiv \mathcal{I}'$ *Q.E.D.* □

Lemma B. 5 (*predicate simplification for conjunction*)
Let $\mathcal{I} = \{I\} \cup \mathcal{I}_1$, where $I = \langle E, P', P'' \rangle$ and P'' has the form $[P''_1$ and $P''_2]$ for some RA predicates P''_1 and P''_2. Let $\mathcal{I}' = \{I_1, I_2\} \cup \mathcal{I}_1$, where $I_1 = \langle E, P', P''_1 \rangle$ and $I_2 = \langle E, P', P''_2 \rangle$. Then $\mathcal{I} \equiv \mathcal{I}'$.

Proof
It is sufficient to prove $\{I\} \equiv \{I_1, I_2\}$. Let $E' = \sigma_{P'}(E)$. Then $[\| I \|] = \forall t \in E' \; [P''_1(t)$ *and* $P''_2(t)]$, whereas $[\| I_1, I_2 \|] = [\forall t \in E' \; P''_1(t)]$ *and* $[\forall t \in E' \; P''_2(t)]$. Therefore $[\| I \|] \Leftrightarrow [\| \{I_1, I_2\} \|]$. Thus $\{I\} \equiv \{I_1, I_2\}$, and then $\mathcal{I} \equiv \mathcal{I}'$ *Q.E.D.* □

Now we present the proof of Theorem 3.1.

Theorem 3. 1 *Given a set of ICs $\mathcal{I}$, there exists a set of simple ICs $\mathcal{I}'$ such that $\mathcal{I} \equiv \mathcal{I}'$.*

Proof
First we prove that given a set of ICs $\mathcal{I}$ for each condition of Definition 3.6 there exists a set of ICs equivalent to $\mathcal{I}$ whose elements satisfy that condition.

Claim 1 *Given a set of ICs $\mathcal{I}$, there exists a set $\mathcal{I}'_1$ equivalent to $\mathcal{I}$ such that each IC in $\mathcal{I}'_1$ satisfies condition* (a) *of Definition 3.6.*

Proof of the claim
It follows from Lemma B.1, Lemma B.2 and Lemma B.3 as a direct consequence of Property 2.1 □

Claim 2 *Given a set of ICs $\mathcal{I}$, there exists a set $\mathcal{I}'_2$ equivalent to $\mathcal{I}$ such that each IC in $\mathcal{I}'_2$ satisfies condition* (b) *of Definition 3.6.*

Proof of the claim
It is sufficient to prove that for any IC in $\mathcal{I}$ not satisfying condition (b) an equivalent IC can be defined that satisfies condition (b). Let $I = \langle E, P', P'' \rangle$ be an IC in $\mathcal{I}$ that does not satisfy such condition. Because of Claim 1 we can assume without loss of generality that $E = R_1 \times \ldots \times R_n$ $(n \geq 1)$, where $R_i \ldots R_n$ are base relations. Let $\underline{I} = \langle \underline{E}, P', P'' \rangle$, where $\underline{E}$ is the Cartesian product of the (base) relations in $\mathcal{P}rel(P') \cup \mathcal{P}rel(P'')$. Then by construction $\mathcal{R}el(\underline{E}) = \mathcal{P}rel(P') \cup \mathcal{P}rel(P'')$. Furthermore by construction $\underline{E} \equiv \pi_{\underline{A}}(E)$, where $\underline{A} = \bigcup_{R_i \in \mathcal{R}el(\underline{E})} \mathcal{A}ttr(R_i)$. Thus by Lemma B.2 $\underline{I} \equiv I$ *Q.E.D.*□

Claim 3 *There exists a set $\mathcal{I}'_3$ equivalent to $\mathcal{I}$ such that each IC in $\mathcal{I}'_3$ satisfies both condition (c) and condition (d) of Definition 3.6.*

Proof of the claim
It follows directly from Lemma B.4 and Lemma B.5. □

To complete the proof of the theorem observe that as a direct consequence of Claim 1, Claim 2 and Claim 3 there exists a set of ICs $\mathcal{I}'$ equivalent to $\mathcal{I}$ whose elements satisfy all the conditions (a), ..., (d) of Definition 3.6. Since all the elements of $\mathcal{I}'$ by construction are *simple* the proof of the theorem is concluded. *Q.E.D.* □