The final publication is available at Springer via http://dx.doi.org/10.1007/BFb0031738

Applying Transition Rules to Bitemporal Deductive Databases for Integrity Constraint Checking

Carme Martín and Jaume Sistac

Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Pau Gargallo 5, 08028 Barcelona - Catalonia
e-mail:{martin,sistac}@lsi.upc.es

Abstract. A bitemporal deductive database is a deductive database that supports valid and transaction time. A set of facts to be inserted and/or deleted in a bitemporal deductive database can be done in a past, present or future valid time. This circumstance causes that the maintenance of database consistency becomes more hard. In this paper, we present a new approach to reduce the difficulty of this problem, based on applying transition and event rules, which explicitly define the insertions and deletions given by a database update. Transition rules range over all the possible cases in which an update could violate some integrity contraint. Although, we have a large amount of transition rules, for each one we argue its utility or we eliminate it. We augment a database with this set of transition and event rules and then standard SLDNF resolution can be used to check satisfaction of integrity constraints.

1 Introduction

Two measures of time were distinguished in [21], called valid time and transaction time. Valid time is the time when the fact is true in the modelled reality, while transaction time is the time when the fact is stored in the database. In a consensus glossary of temporal database concepts [8], a deductive database that supports valid time and transaction time is called a bitemporal deductive database and we denote it bt-ddb.

An integrity constraint is a condition that a database is required to satisfy at any time. We deal with static and dynamic constraints formulated in a first order language. A bt-ddb must be consistent, that is, when performing a past, present or future update, that happens at some valid time point, it is necessary to validate whether this update violates some integrity constraint, and if so the update must be rejected. The possibility of past, present and future updates in a bt-ddb causes that the maintenance of database consistency becomes more difficult.

Integrity constraint checking is an essential issue, which has been widely studied in relational and deductive databases (see for example [4] for a comprehensive state of the art survey), but not in the field of temporal deductive databases. The simplest solution to integrity checking would be to evaluate each constraint whenever the database is updated. However, it is usually too costly and highly redundant, since it does not take advantage of the fact that the database satisfies the constraints prior to

the update. In this paper, we present a new approach for consistency maintenance in bt-ddb, that incorporates transaction time to our previous work (see [13] and [14]), based on applying transition and event rules, which explicitly define the insertions and deletions given by a database update. Transition rules range over all the possible cases in which an update could violate some integrity contraint. Although, we have a large amount of transition rules, for each one we argue its utility or we eliminate it. We augment a database with this set of simplified transition rules and event rules and then standard SLDNF resolution can be used to check satisfaction of integrity constraints.

The paper is organised as follows. The next section defines basic concepts of bt-ddbs and introduces a simple example that will be used throughout the paper. Section 3 presents the concepts of events, transition and event rules. Section 4 describes the application of transition rules for integrity constraint checking in bt-ddbs. Particularly, this part shows the utility of the transition rules for the example presented in section 2. Section 5 compares our approach with previous related work. Finally, section 6 gives the conclusions and points out future work.

2 Bitemporal Deductive Databases

A bt-ddb D consists of three finite sets: a set F of facts, a set R of deductive rules, and a set I of integrity constraints. The set of facts is called the extensional database (EDB), and the set of deductive rules is called the intensional database (IDB). A base predicate appears only in the extensional database and possibly in the body of deductive rules. A derived predicate appears only in the intensional database. Every bt-ddb can be defined in this form.

Facts, rules and integrity constraints are formulated in a first order language. We will use names beginning with a lower case letter for predicate symbols and constants and a capital letter for variables.

We consider a temporal domain τ isomorphic to the set of natural numbers, over which is defined the linear (total) order $<_\tau$, where $t_i <_\tau t_j$ means t_i occurs before t_j. The set τ is used as the basis for incorporating the temporal dimensions into the database.

We assume that database predicates are either base or derived. For example, the base predicate *offered*(C,T_V) and the derived predicate *some_enrol*(C,T_V) both contain a last term which is a valid time point ranging over the temporal domain τ. Integrity constraints use the usual operators $=, >, <, \geq, \leq$ and $\neq$ to compare a valid time points and to express static and dynamic constraints. For example: *ic4* $\leftarrow$ *offered*$(C,T_V) \wedge \neg$*many_students*$(C,T_V) \wedge$ *many_students*$(C,T1_V) \wedge T1_V<T_V$. The examples presented here are explained in detail in subsection 2.3.

We adopt a closed time interval model based in the valid time representation presented in [20], adding a transaction time dimension. Including transaction time we ensure that every old state is preserved. If we store only valid time one cannot remember if during a given period one knew another information different from the current one. We are interested in the history of the database and we willing to pay a high cost of the storage of old states. [20] uses two segments to representing current and history data, in which two valid time points are added, named FROM and TO

(defining a valid time interval), valid time start and end in our case. We only use the equivalent to one segment and we add two more time points (transaction time start and end) to represent transaction time and to define a transaction time interval.

A fact is a ground atom. Last terms of any fact are four time points values ranging over the temporal domain τ: valid time-start (t_{vs}), valid time-end (t_{ve}) corresponding to the lower and upper bounds of the valid time interval and transaction time-start (t_{ts}), transaction time-end (t_{te}) corresponding to the lower and upper bounds of the transaction time interval.

Each fact has a precise valid time-start t_{vs} value stored from transaction time-start t_{ts} to *now* (denoting the current time) or to transaction time-end t_{te} when finally the fact cannot be accessible from current time anymore. However, the valid time-end value t_{ve} may not be known. In this case, t_{ve} is given the default value *forever* denoting an artificial time point for the end of time ready to handle future information, but that will change to precise value t_{ve} when the user knows it (see subsection 3.1).

2.1 Deductive Rules

A deductive rule is a formula of the form:

$$p \leftarrow L_1 \wedge ... \wedge L_n \quad with\ n \geq 1,$$

where p is an atom, denoting the conclusion, $L_1 \wedge ... \wedge L_n$ are literals representing conditions. Any variables in p, $L_1 \wedge ... \wedge L_n$ are assumed to be universally quantified over the whole formula. A **derived predicate** p may be defined by means of one or more deductive rules, but for the sake of simplicity, we only show the first case in this paper.

Condition predicates may be ordinary or evaluable. The former are base or derived predicates, while the latter are built-in predicates that can be evaluated without accessing the database.

In this paper we deal with stratified databases [2] and, as usual we require the bt-ddb before and after any updates to be allowed [11].

2.2 Integrity Constraints

An integrity constraint is a closed first order formula that the bt-ddb is required to satisfy. We deal with constraints that have the form of a denial:

$$\leftarrow L_1 \wedge ... \wedge L_n \quad with\ n \geq 1,$$

where each L_i is a literal. Any variables in $L_1 \wedge ... \wedge L_n$ are assumed to be universally quantified over the whole formula.

For the sake of uniformity we associate with each integrity constraint an inconsistency predicate *Icn*, and thus it has the same form as a deductive rule. We call them integrity rules. Then, we rewrite the former denial as:

$$Icn \leftarrow L_1 \wedge ... \wedge L_m \quad with\ m \geq 1.$$

The evolution through time of a deductive database can be described by valid and transaction time intervals for each predicate. According to this evolution scheme, static and dynamic constraints can be distinguished: the former restrict the validity to only one valid time of the bt-ddb, while the latter relate the validity to past and/or future valid times in addition to another one.

2.3 Example

Base Predicates.

offered(C,T_V) expresses that "course *C* is offered at valid time T_V"
takes(S,C,T_V) expresses that "the student S is enrolled in course *C* at valid time T_V".

Deductive Rules.

R.1 $some\_enrol(C,T_V) \leftarrow takes(S,C,T_V)$.
R.2 $many\_students(C,T_V) \leftarrow takes(S1,C,T_V) \wedge takes(S2,C,T_V) \wedge S1 \neq S2$.

some_enrol(C,T_V) expresses that "the course *C* has one or more students at valid time T_V", and *many_students(C,T_V)* expresses that "the course *C* has two students at valid time T_V, at least".

Static Integrity Constraints.

IC.1 $ic1 \leftarrow takes(S,C,T_V) \wedge \neg offered(C,T_V)$
IC.2 $ic2 \leftarrow offered(C,T_V) \wedge \neg some\_enrol(C,T_V)$.

ic1 and *ic2* , respectively, enforce the properties that "a student *S* can only be enrolled in course *C* if course *C* is offered" and "for course *C* to be offered, it must have at least one student enrolled".

Dynamic Integrity Constraints.

IC.3 $ic3 \leftarrow takes(S,software\ engineering,T_V) \wedge takes(S,information\ systems,T1_V) \wedge T1_V \leq T_V$.
IC.4 $ic4 \leftarrow offered(C,T_V) \wedge \neg many\_students(C,T_V) \wedge many\_students(C,T1_V) \wedge T1_V < T_V$.

ic3 and *ic4*, respectively, enforce the properties that "if a student *S* is enrolled in the course *software engineering*, this student cannot be enrolled or cannot have been enrolled in the course *information systems*", "if two or more students were enrolled in a course *C*, this course cannot have less than two students in the future".

3 Events, Transition and Event Rules

In this section, we begin by adapting the concepts of event, transition and event rules that were formalised in [16] for the events model as an approach for the design of information systems from deductive conceptual models, and was applied in [22] to address database and transaction design decisions. In [16] and [22] valid and transaction time are equivalent and the database can only be updated in the current state. In our case, we explicitly distinguish between valid and transaction time and the updates can be done in a past, present or future valid time.

3.1 Events

Let D be a deductive database at transaction time point t_t-1, U an update and D' the updated deductive database at transaction time point t_t (*now*) as one can see in figure 3.1. We assume for the moment that U consists of an unspecified set of facts to be inserted and/or deleted and the bt-ddb can only be updated in the transaction time point *now*.

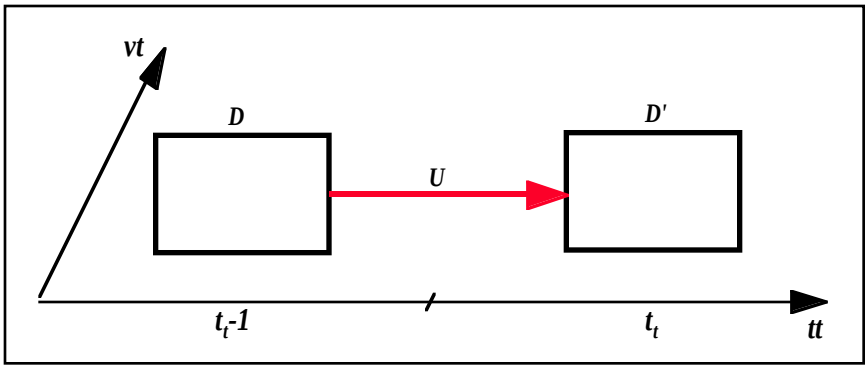

Fig. 3.1.

Let $p(x,t_V)$ be a predicate in D and let $p'(x,t_V)$ denote the same predicate evaluated in D'. Assuming that $p(x,t_V)$ holds in D, where x is a vector of constants, and t_V is a valid time point, two cases are possible:

$p'(x,t_V)$ also holds in D' (both $p(x,t_V)$ and $p'(x,t_V)$ are true). (1)

$p'(x,t_V)$ does not hold in D' ($p(x,t_V)$ is true, but $p'(x,t_V)$ is false). (2)

And assuming that $p'(x,t_V)$ holds in D', two cases are also possible:

$p(x,t_V)$ also holds in D (both $p(x,t_V)$ and $p'(x,t_V)$ are true). (3)

$p(x,t_V)$ does not hold in D ($p'(x,t_V)$ is true, but $p(x,t_V)$ is false). (4)

In case (2) we say that a **deletion event** occurs in the transition at valid time point t_V, we denote it by $\delta p(x,t_V)$ and we store it at transaction time point t_t as shown in figure 3.2.

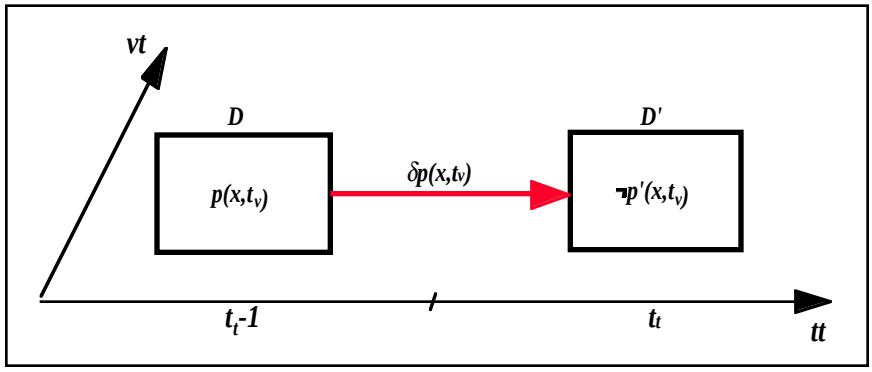

Fig. 3.2.

In case (4) we say that an **insertion event** occurs in the transition at valid time point t_V, we denote it by $\iota p(x,t_V)$ and we store it at transaction time point t_t as shown in figure 3.3.

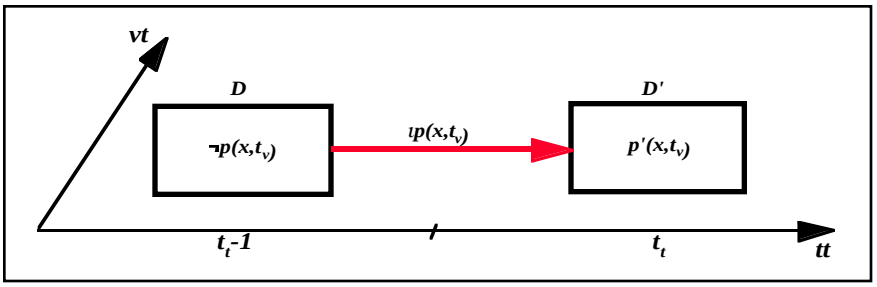

Fig. 3.3.

Formally, we associate an insertion event predicate ιp with each derived or inconsistency predicate p and a deletion event predicate δp with each derived predicate, defined as:

$\forall X,T_V\ (\iota p(X,T_V) \leftrightarrow p'(X,T_V) \wedge \neg p(X,T_V))$. (5)

$\forall X,T_V\ (\delta p(X,T_V) \leftrightarrow p(X,T_V) \wedge \neg p'(X,T_V))$. (6)

where X is a vector of variables and T is a valid time point variable.

From the above, we then have the equivalencies:

$\forall X,T_V\ (p'(X,T_V) \leftrightarrow [p(X,T_V) \wedge \neg\delta p(X,T_V)] \vee \iota p(X,T_V))$. (7)

$\forall X,T_V\ (\neg p'(X,T_V) \leftrightarrow [\neg p(X,T_V) \wedge \neg\iota p(X,T_V)] \vee \delta p(X,T_V))$. (8)

which relate the predicate p' at transaction time point t_t to the predicate p at transaction time point t_t-1 and the events given by the transaction.

If p is a derived predicate, then ιp and δp represent induced insertions and deletions respectively.

If p is an inconsistency predicate, then ιp that occur during the transition will correspond to violations of its integrity constraint. For example, if a given transition induces ιicn, this will mean that such a transition leads to a violation of integrity constraint icn. Note that for inconsistency predicates δp cannot happen in any transition, since we assume that the bt-ddb is consistent before the update, and thus icn is always false.

We also use definitions (5) and (6) above for base predicates. In this case, ιp and δp represent the events given by the update. Therefore, we assume from now on that U consists of an unspecified set of insertion and/or deletion of events given by the update.

Note that an event happens at some time instant, while we require time intervals to express the changes produced by the transaction. Therefore, when an insertion event $\iota p(x,t_V)$ happens in a transaction time t_t we really represent: $p_r(x,t_V,forever,t_t,now)$, and when a deletion event $\delta p(x,t_V)$ occurs in a transaction time t_t we modify $p_r(x,t_{VS},t_{Ve},t_{TS},now)$ by $p_r(x,t_{VS},t_V\text{-}1,t_t,now)$ and $p_r(x,t_{VS},t_{Ve},t_{TS},t_t\text{-}1)$. When a deletion event occurs we do not really remove information; instead we store the fact that it has existed from one valid time to another valid time.

Example. Suppose the update *U* at transaction time 3:

{ιoffered(databases,2), ιtakes(ton,databases,2), δtakes(maria,logic,2)}

on the bt-ddb as shown in figure 3.4.

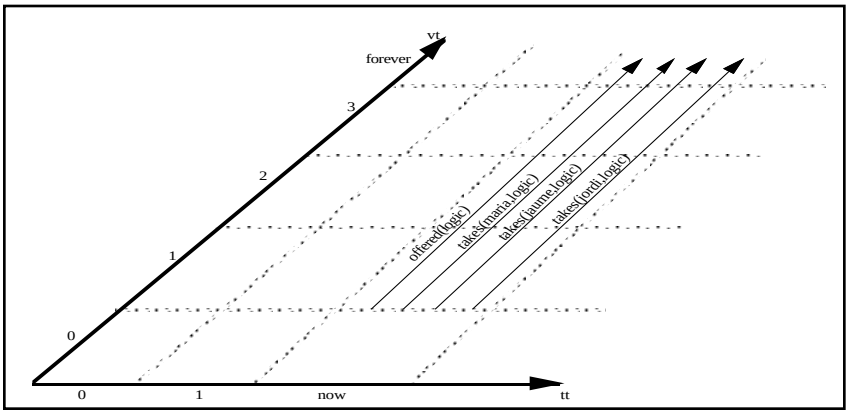

Fig. 3.4.

If this transaction does not violate any integrity constraint, at transaction time 3, we have the bt-ddb as shown in figure 3.5, and their facts will be, including the temporal information that appears in the figure 3.5 axis co-ordinates.

$offered_r$(logic,1,forever,2,now)
$takes_r$(jaume,logic,1,forever,2,now)
$takes_r$(jordi,logic,1,forever,2,now)
$takes_r$(maria,logic,1,forever,2,2)
$takes_r$(maria,logic,1,1,3,now)
$offered_r$(databases,2,forever,3,now)
$takes_r$(ton,databases,2,forever,3,now)

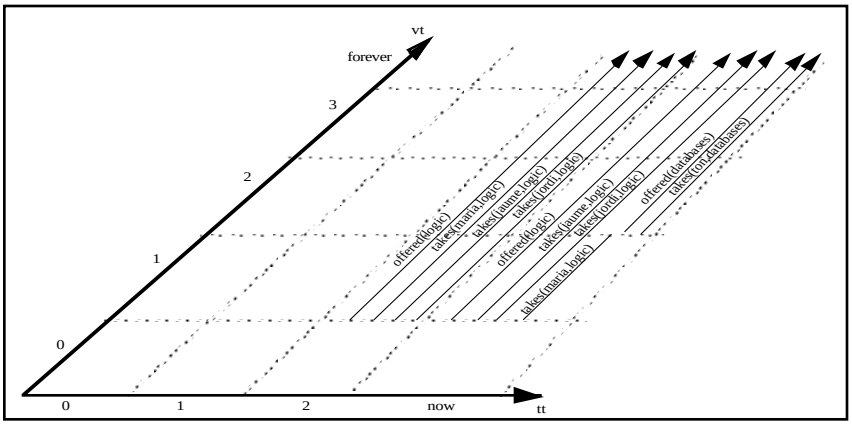

Fig. 3.5.

3.2 Transition Rules

Let $p \leftarrow L_1, ..., L_m$ be a deductive or inconsistency rule. When the rule is to be evaluated in the updated bt-ddb, its form is $p' \leftarrow L_1', ..., L_m'$, where L_i' (i = 1..m) is obtained by replacing the predicate Q of L_i with Q'. Now if we rewrite each literal in the body by its equivalent definition, given in (7) or (8), we get a new rule called a **transition rule**, which defines predicate p' in the updated bt-ddb in terms of transaction time point *now-1* of the predicates appearing in the body of the rule, and the events that occur at transaction time point *now*.

More precisely, if L_i' is an ordinary positive literal $Q_i'(X_i,T_{vi})$ we apply (7) and replace it with:

$(Q_i(X_i,T_{vi}) \wedge \neg\delta Q_i(X_i,T_{vi})) \vee \iota Q_i(X_i,T_{vi})$

and if L_i' is an ordinary negative literal $\neg Q_i'(X_i,T_i)$ we apply (8) and replace it with:

$(\neg Q_i(X_i,T_{vi}) \wedge \neg\iota Q_i(X_i,T_{vi})) \vee \delta Q_i(X_i,T_{vi})$

If L_i is an evaluable predicate, we just replace L_i'(positive or negative) by its current L_i.

It will be easier to refer to the resulting expressions if we denote by:

$O(L_i')$	$= (Q_i(X_i,T_{vi}) \wedge \neg\delta Q_i(X_i,T_{vi}))$	*if* $L_i' = Q_i'(X_i,T_{vi})$
	$= (\neg Q_i(X_i,T_{vi}) \wedge \neg\iota Q_i(X_i,T_{vi}))$	*if* $L_i' = \neg Q_i'(X_i,T_{vi})$
	$= L_i$	*if* L_i *is evaluable*
$N(L_i')$	$= \iota Q_i(X_i,T_{vi})$	*if* $L_i' = Q_i'(X_i,T_{vi})$
	$= \delta Q_i(X_i,T_{vi})$	*if* $L_i' = \neg Q_i'(X_i,T_{vi})$

Both $O(L_i')$ and $N(L_i')$ express conditions for which L_i' is true. $O(L_i')$ corresponds to the case that L_i' holds because L_i was already true in the **O**ld transaction time point *now-1* and has not been deleted, while $N(L_i')$ corresponds to the case that $N(L_i')$ holds because it is **N**ew, induced in the transition, and false before. Note that $O(L_i') \rightarrow L_i$ and $N(L_i') \rightarrow L_i$.

With this notation, the equivalencies (7) and (8) become:

$$\forall X,T_V\ (p'(X,T_V) \leftrightarrow O(p'(X,T_V)) \vee N(p'(X,T_V))). \quad (9)$$

$$\forall X,T_V\ (\neg p'(X,T_V) \leftrightarrow O(\neg p'(X,T_V)) \vee N(\neg p'(X,T_V))). \quad (10)$$

and applying them to each of the L_i' (i = 1...n) literals, we get:

$$p'(X,T_V) \leftarrow \bigwedge_{i=1}^{i=n} [O(L_i') \vee N(L_i') \mid O(L_i')] \quad (11)$$

where the first option is taken if L_i' is an ordinary literal, and the second one if L_i' is evaluable. After distributing $\wedge$ over $\vee$, we get an equivalent set of 2^k transition rules, each of them with the general form:

$$p_j'(X,T_V) \leftarrow \bigwedge_{i=1}^{i=n} [O(L_i') \mid N(L_i')] \qquad \text{with } j = 1, ..., 2^k \quad (12)$$

where k is the number of ordinary literals in the $p'(X,T)$ rule, and

$$p'(X,T_V) \leftarrow p_j'(X,T_V) \qquad \text{with } j = 1, ..., 2^k. \quad (13)$$

We are conscious of the resulting amount of transition rules and we present afterwards in this paper some simplifications to drastically reduce them.

Note that in the case of integrity constraints, $\neg ici_1'$ always holds because the bt-ddb is assumed to be consistent at transaction time point *now-1*. For example, $takes(S,C,T_V)$ and $\neg offered(C,T_V)$ in:

$$ic1_1' \leftarrow takes(S,C,T_V) \wedge \neg \delta takes(S,C,T_V) \wedge \neg offered(C,T_V) \wedge \neg \iota offered(C,T_V),$$

are always false in a transaction time point *now-1*, and thus we can eliminate this transition rule.

3.3 Insertion Event Rules

Let *p* be a derived or inconsistency predicate. Insertion events of *p* were defined in (5) as:

$$\forall X,T_V \ (\iota p(X,T_V) \leftrightarrow p'(X,T_V) \wedge \neg p(X,T_V)).$$

And replacing $p'(X,T)$ by its equivalent definition given in (13) we get:

$$\iota p(X,T_V) \leftarrow p_i'(X,T_V) \wedge \neg p(X,T_V) \qquad with\ i = 1, ..., 2^k. \tag{14}$$

By replacing $p_i'(X,T_V)$ with its equivalent definition given in (12), we get a set of **insertion events rules**. They allow us to deduce which ιp (induced insertions) happen in a transition. If *p* is an inconsistency predicate, ιp facts correspond to a violation of the integrity constraint. Note that in the case of integrity constraints, $\neg ici$ always holds because the bt-ddb was consistent at transaction time point *now-1*, and we can simplify this literal:

$$\iota ici \leftarrow ici_i' \quad i=2, ..., 2^k \tag{15}$$

3.4 Deletion Event Rules

Let *p* be a derived predicate. Deletion events of *p* were defined in (6) as:

$$\forall X,T_V \ (\delta p(X,T_V) \leftrightarrow p(X,T_V) \wedge \neg p'(X,T_V)).$$

And replacing $p'(X,T)$ by its equivalent definition given in (13) we get:

$$\delta p(X,T_V) \leftarrow p(X,T_V) \wedge \neg p_1'(X,T_V) \wedge ... \wedge \neg p_i'(X,T_V) \wedge ... \wedge \neg p_{2^k}'(X,T_V) \tag{16}$$

By replacing $p_i'(X,T_V)$ with its equivalent definition given in (12), we get a set of **deletion events rules**. They allow us to deduce which δp (induced deletions) happen in a transition. Note that in the case of integrity constraints, δicn cannot happen in any transition, since we assume that the bt-ddb is consistent before the update, and thus *icn* is always false.

3.5 The Augmented Database

Let D be a bt-ddb. We denote the augmented bt-ddb by A(D), based in the concept of augmented deductive database defined in [17], to the bt-ddb consisting of D, its transition rules and its event rules.

If SLDNF resolution is complete for D, then it will also be complete for A(D). [17].

4 Applying Transition Rules for Integrity Constraint Checking in Bitemporal Deductive Databases

The augmented bt-ddb described in the previous section can be used directly to check if a transaction produces or not inconsistencies.

Let *D* be a bt-ddb, *A(D)* the augmented bt-ddb, and *TR* a transaction consisting of a set of events at valid time point *T*. If *TR* leads to an inconsistency then some of the *ιicn* will hold in the transition. Using SLDNF proof procedure, *TR* violates integrity constraint *icn* if the goal $\leftarrow \iota icn$ succeeds from input set $A(D) \cup TR$. If every branch of the SLDNF-search space for $A(D) \cup TR \cup \{\leftarrow \iota icn\}$ is a failure branch, then *TR* does not violate *icn*, as show in figure 4.1.

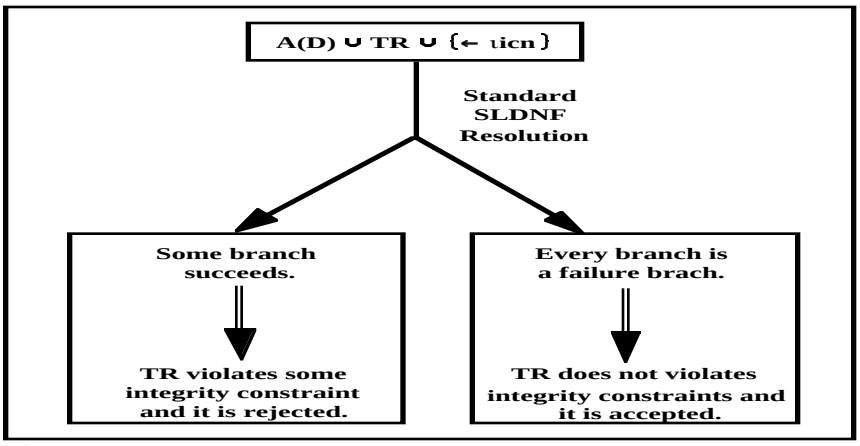

Fig. 4.1.

From the insertion event rule *ιicn* for (15), we then use the transition rules of *icn* to proof the consistency of the bt-ddb. Following, we illustrate the utility of the transition rules for integrity constraints checking, but first we explain the simplifications applied to them. We show some updates that can violate some transition rules obtained for the constraints in the example presented in section 2. Specifically, we select the constraints where we can apply the different types of simplifications. Next subsections simulate (using a discontinuous line) what would happen if the transaction were applied in the bt-ddb. Note that we do not really apply the transaction, we only simulate.

4.1 Simplifications of Transition Rules.

In the following subsections we are going to show the following types of simplifications:

***Inconsistent Rules Simplification*:** When we find $\neg \iota p(X,T) \wedge \iota p(p,T1) \wedge T1<T$ in a transition rule, we can eliminate it because *ιp(p,T1)* is from *T1* to *forever* and that includes *T*.

***Null Effect Rules Simplification*:** When we find $\delta p(X,T) \wedge \iota p(X,T1) \wedge T1<T$ in a transition rule, we can eliminate it because $\iota p(p,T1)$ is from *T1* to *forever* and that includes *T*, it does not matter that $\delta p(X,T)$.

***Mutually Exclusive Rules Simplification*:** When we find $\iota p(X,T) \wedge \delta q(X,T)$ in one transtion rule, and $\delta p(X,T) \wedge \iota q(X,T)$ in another transition rule we can eliminate both of them because if one of them holds then it means the other one happened in a previous transaction time and the database is already inconsistent.

Our simplifications consider an assumption concerning data manipulation: In a transaction we cannot insert and delete the same fact. We have to insert the fact in a transaction at transaction time *now* and then we can delete it in another transaction in a future transaction time. This restriction reduces considerably the difficulty of the update in bt-ddb and it is not too unmanageable to the user.

4.2 Example with Ic1

Consider the integrity constraint:

$$ic1 \leftarrow takes(S,C,T) \wedge \neg offered(C,T).$$

The **transition rules** we obtain after replacing literals and distributing $\wedge$ over $\vee$ are:

$ic1_2' \leftarrow takes(S,C,T) \wedge \neg\delta takes(S,C,T) \wedge \delta offered(C,T).$
$ic1_3' \leftarrow \iota takes(S,C,T) \wedge \neg offered(C,T) \wedge \neg\iota offered(C,T).$
$ic1_4' \leftarrow \iota takes(S,C,T) \wedge \delta offered(C,T).$
$ic1' \leftarrow ic1_i' \quad i = 2,...,4$

We show in figure 4.2 an example of a transaction that violates $ic1_2'$ at transaction time 3. And in figure 4.3 we show its derivation tree.

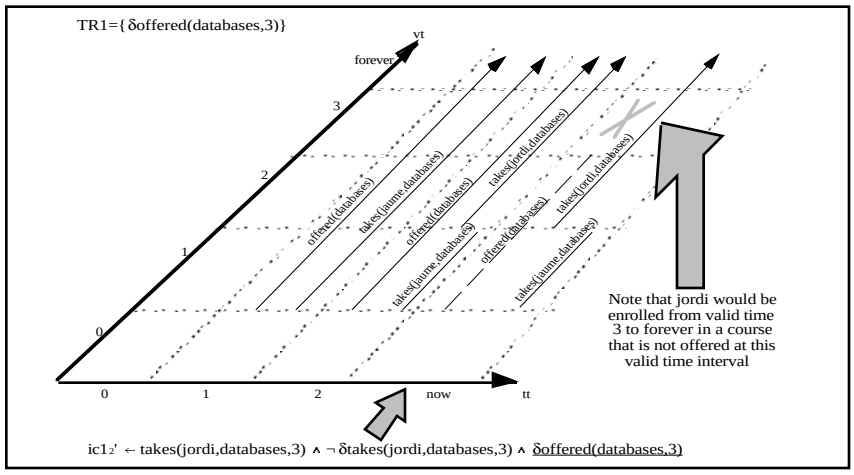

Fig. 4.2.

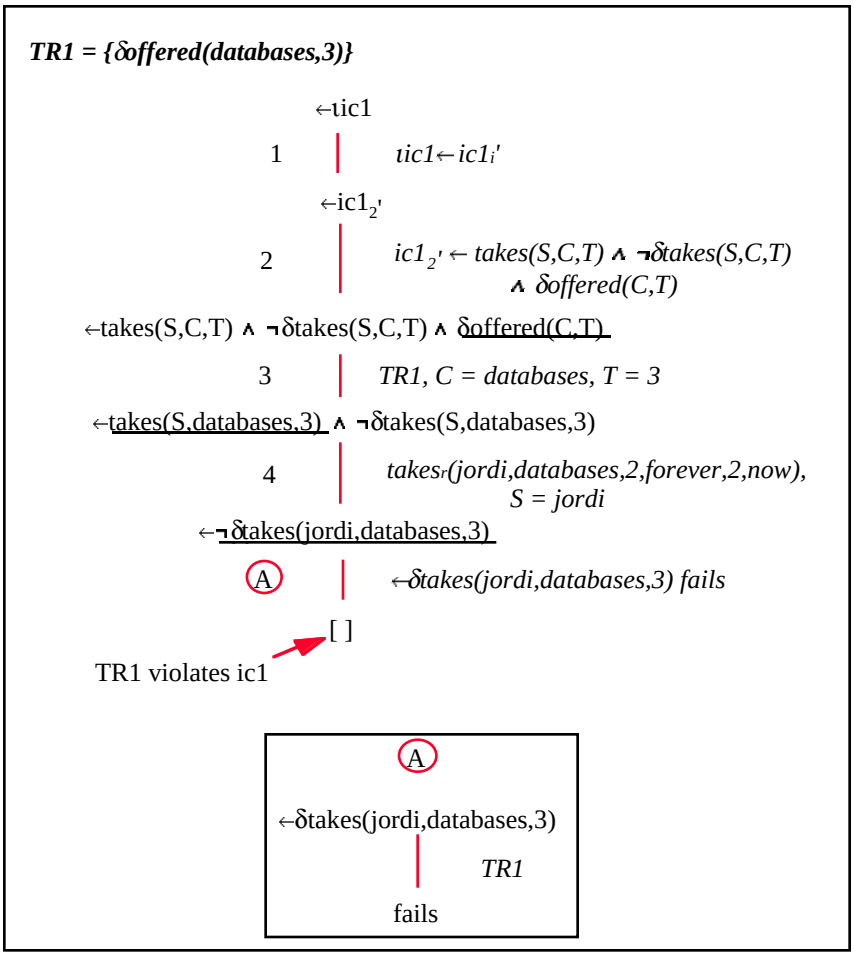

Fig. 4.3.

Steps 1 and 2 are SLDNF resolution steps where rules of A(D) act as input clauses. We may have several rules to resolve with, although only the failure branch that shows the violation of the integrity constraint is shown here.

Note that at steps 3 and 4 the predicate references to the transaction and to the database, respectively, and we go to the transaction or to the bt-ddb to find it, respectively.

At step A, the selected literal is: *¬δtakes(jordi,databases,3)*. In order to get a successful derivation, SLDNF search space must fail finitely for the subsidiary tree of: *{←δtakes(jordi,databases,3)}*.

Note that in the third step we have selected literal *δoffered(C,T)* instead of *takes(S,C,T)*. Given that in most real databases the number of facts is likely to be much greater than the number of events produced in a transition, it seems convenient to use a strategy of selecting first the events (once fully instantiated if they are negative).

We show in figure 4.4 an example of a transaction that violates $ic1_3'$.

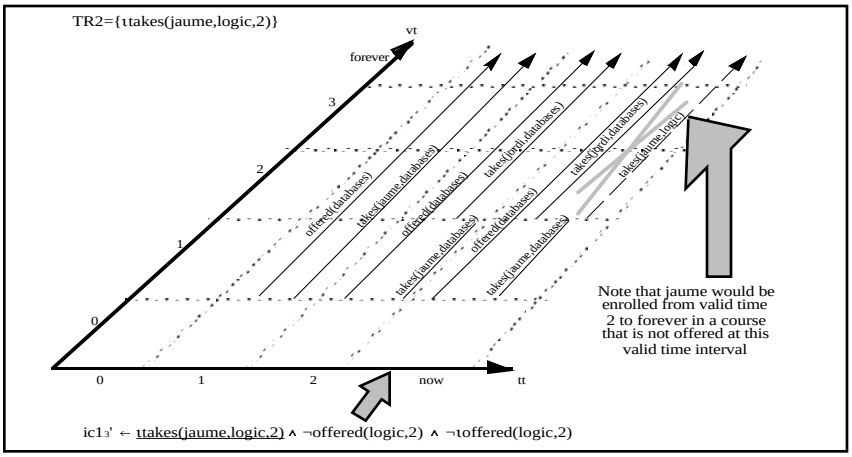

Fig. 4.4.

Note that transition rule $ic1_4' \leftarrow \iota takes(S,C,T) \wedge \delta offered(C,T)$ never holds because the bt-ddb was consistent at transaction time *now-1*, *ic2* requires for a course to be offered it must have at least one student enrolled; if we delete a course *C* at valid time *T*, for *ic2* we must have one student enrolled in *C* at valid time *T* and that is $ic1_2'$. Therefore, we can eliminate $ic1_4'$ which is *mutually exclusive rule* with $ic2_4' \leftarrow \iota offered(C,T) \wedge \delta some\_enrol(C,T)$ and we finally obtain:

$ic1_2' \leftarrow takes(S,C,T) \wedge \neg\delta takes(S,C,T) \wedge \delta offered(C,T).$
$ic1_3' \leftarrow \iota takes(S,C,T) \wedge \neg offered(C,T) \wedge \neg\iota offered(C,T).$
$ic1' \leftarrow ic1_i' \quad i = 2,3$

4.3 Example with Ic4

Consider the integrity constraint:

$ic4 \leftarrow offered(C,T) \wedge \neg many\_students(C,T) \wedge many\_students(C,T1) \wedge T1<T.$

The **transition rules** we obtain after replacing literals and distributing $\wedge$ over $\vee$ are:

$ic4_2' \leftarrow offered(C,T) \wedge \neg\delta offered(C,T) \wedge \neg many\_students(C,T) \wedge$
$\neg\iota many\_students(C,T) \wedge \iota many\_students(C,T1) \wedge T1<T.$
$ic4_3' \leftarrow offered(C,T) \wedge \neg\delta offered(C,T) \wedge \delta many\_students(C,T) \wedge$
$many\_students(C,T1) \wedge \neg\delta many\_students(C,T1) \wedge T1<T.$
$ic4_4' \leftarrow offered(C,T) \wedge \neg\delta offered(C,T) \wedge \delta many\_students(C,T) \wedge$
$\iota many\_students(C,T1) \wedge T1<T.$
$ic4_5' \leftarrow \iota offered(C,T) \wedge \neg many\_students(C,T) \wedge \neg\iota many\_students(C,T) \wedge$
$many\_students(C,T1) \wedge \neg\delta many\_students(C,T1) \wedge T1<T.$

$ic4_6' \leftarrow \iota offered(C,T) \wedge \neg many\_students(C,T) \wedge \neg \iota many\_students(C,T) \wedge$
$\iota many\_students(C,T1) \wedge T1<T.$
$ic4_7' \leftarrow \iota offered(C,T) \wedge \delta many\_students(C,T) \wedge many\_students(C,T1) \wedge$
$\neg \delta many\_students(C,T1) \wedge T1<T.$
$ic4_8' \leftarrow \iota offered(C,T) \wedge \delta many\_students(C,T) \wedge \iota many\_students(C,T1) \wedge T1<T.$
$ic4' \leftarrow ic4_i' \quad i = 2,...,8$

where $\iota many\_students(C,T)$ and $\delta many\_students(C,T)$ are:

$\iota many\_students(C,T) \leftarrow many\_students_1'(C,T) \wedge \neg many\_students(C,T).$
$\iota many\_students(C,T) \leftarrow many\_students_2'(C,T) \wedge \neg many\_students(C,T).$
$\iota many\_students(C,T) \leftarrow many\_students_3'(C,T) \wedge \neg many\_students(C,T).$
$\iota many\_students(C,T) \leftarrow many\_students_4'(C,T) \wedge \neg many\_students(C,T).$
$\delta many\_students(C,T) \leftarrow many\_students(C,T) \wedge \neg many\_students_1'(C,T) \wedge$
$\neg many\_students_2'(C,T) \wedge \neg many\_students_3'(C,T) \wedge$
$\neg many\_students_4'(C,T).$

and where $many\_students_1'(C,T)$, $many\_students_2'(C,T)$, $many\_students_3'(C,T)$ and $many\_students_4'(C,T)$ are:

$many\_students(C,T)_1' \leftarrow takes(S1,C,T) \wedge \neg \delta takes(S1,C,T) \wedge takes(S2,C,T) \wedge$
$\neg \delta takes(S2,C,T) \wedge S1 \neq S2.$
$many\_students(C,T)_2' \leftarrow takes(S1,C,T) \wedge \neg \delta takes(S1,C,T) \wedge \iota takes(S2,C,T) \wedge$
$S1 \neq S2.$
$many\_students(C,T)_3' \leftarrow \iota takes(S1,C,T) \wedge takes(S2,C,T) \wedge \neg \delta takes(S2,C,T) \wedge$
$S1 \neq S2.$
$many\_students(C,T)_4' \leftarrow \iota takes(S1,C,T) \wedge \iota takes(S2,C,T) \wedge S1 \neq S2.$

We show in figure 4.5 an example of a transaction that violates $ic4_3'$.

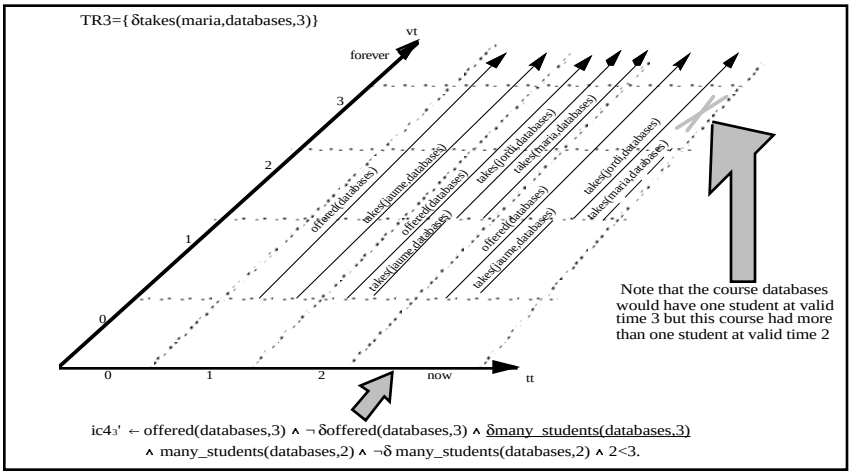

Fig. 4.5.

We show in figure 4.6 an example of a transaction that violates $ic4_5'$.

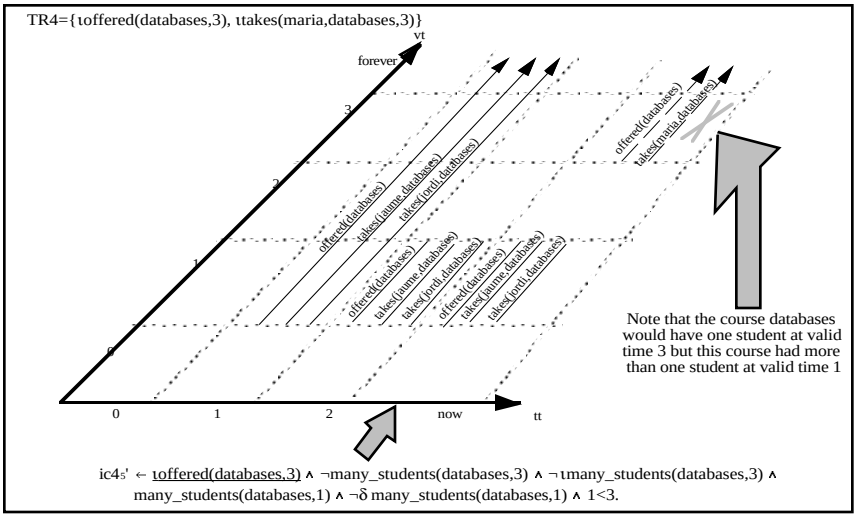

Fig. 4.6.

Note that integrity constraints $ic4_2'$, $ic4_4'$,$ic4_6'$,$ic4_7'$ and $ic4_8'$ can be eliminated.

$ic4_2' \leftarrow offered(C,T) \wedge \neg\delta offered(C,T) \wedge \neg\iota many\_students(C,T) \wedge \neg\iota many\_students(C,T) \wedge \iota many\_students(C,T1) \wedge T1<T$.

and

$ic4_6' \leftarrow \iota offered(C,T) \wedge \neg many\_students(C,T) \wedge \neg\iota many\_students(C,T) \wedge \iota many\_students(C,T1) \wedge T1<T.$

can be eliminated in order to be *inconsistent rules* because if we $\iota many\_students(C,T1)$ in a $T1 <T$, we $\iota many\_students(C,T)$ so when one insert at valid time *T1*, insert from *T1* to *forever*, including *T*.

$ic4_4' \leftarrow offered(C,T) \wedge \neg\delta offered(C,T) \wedge \delta many\_students(C,T) \wedge \iota many\_students(C,T1) \wedge T1<T$.

and

$ic4_8' \leftarrow \iota offered(C,T) \wedge \delta many\_students(C,T) \wedge \iota many\_students(C,T1) \wedge T1<T$.

can be eliminated in order to be *null effect rules* because if we $\iota many\_students(C,T1)$, it does not matter the number of students that one can delete so *ic1* requires one student enrolled at least and we insert another one when $\iota many\_students(C,T1)$ in $T1<T$.

$ic4_7' \leftarrow \iota offered(C,T) \wedge \delta many\_students(C,T) \wedge many\_students(C,T1) \wedge \neg\delta many\_students(C,T1) \wedge T1<T$.

can be eliminated because the bt-ddb was consistent at transaction time *now-1* and *ic1* requires for a student to be enrolled in a course, that this course has to be offered so if we delete students of a course *C* at valid time *T*, for *ic1* we must have this course *C* at valid time *T* and that is $ic4_3'$. Therefore, we can eliminate $ic4_7'$ which is *mutually exclusive rule* with $ic1_4' \leftarrow \iota takes(S,C,T) \wedge \delta offered(C,T)$.

After this simplifications we finally obtain:

$ic4_3' \leftarrow offered(C,T) \wedge \neg\delta offered(C,T) \wedge \delta many\_students(C,T) \wedge$
$many\_students(C,T1) \wedge \neg\delta many\_students(C,T1) \wedge T1<T.$
$ic4_5' \leftarrow \iota offered(C,T) \wedge \neg\iota many\_students(C,T) \wedge \neg\iota many\_students(C,T) \wedge$
$many\_students(C,T1) \wedge \neg\delta many\_students(C,T1) \wedge T1<T.$
$ic4' \leftarrow ic4_i' \qquad i = 3,5$

5 Comparison with Other Methods

Only a few methods for integrity checking in deductive databases incorporate time, as you can see in the bibliography of this research area in [9].

There are methods, such as Chomicki's method [5], [6] or Wüthrich's method [28], that use temporal logic to formulate integrity constraints. But these methods do not contain temporal information explicitly as in our case.

We could have chosen a logic-based Event Calculus, such as [10], [19], [24] or [27], to develop our method, but we think this choice is unimportant, considering that the main thing presented in this paper is a new integrity constraint checking approach for bt-ddb.

Plexousakis's method [18] formulates integrity constraints in a first order language provided by Telos [12], a language for knowledge representation. Telos adopts Allen's [1] interval based time model for representing historical information. The method consists in generating a parameterized simplified structure (PSS) for each literal of the integrity constraints and the deductive rules that can be affected by an update and it needs to construct a dependency graph for integrity constraint checking. The method for finding the simplified form is an extension of Nicolas's method [15], which includes temporal treatment.

Summarising, Plexousakis's method is focussed on temporal integrity constraint simplification with a large number of the thirteen relationships which can exist between two time intervals, defined by Allen [1]. Whereas in our case we do not have Allen's relationships. Furthermore, the method needs to use a meta-interpreter from Telos's language to parameterize a simplified structure. In contrast, we use only SLDNF proof procedure directly provided in a Prolog system.

6 Conclusions and Further Work

In this paper we have presented how and why to apply transition rules for integrity constraints checking in bt-ddbs. Given an update, the transition rules define predicate *p'* in the updated bt-ddb in terms of transaction time point *now-1* of the predicates appearing in the body of the rule, and the events that occur at transaction time point *now*. Event rules define explicitly the changes induced by the update on the derived predicates. We clarify that transition rules are important to recognise all the possible cases that produce violations of integrity constraints.

Our further work consists in completing our approach for integrity constraints checking in bt-ddbs with: updates of integrity constraints and deductive rules,

recursive rules, simplifications of the event rules and more simplifications of transition rules to increase the efficiency incorporating related work in this area, for example [7].

Like other temporal deductive database systems as ChronoLog [3] and ChronoBase [23], we would try to incorporate our work in the FOLRE project [26].

Acknowledgements

The authors would like to thank Antoni Olivé for the support he has given to this work and also P. Costa, D. Costal, E. Mayol, J. A. Pastor, C. Quer, M. R. Sancho, E. Teniente, T. Urpí and specially Michael Gertz for many useful comments and discussions.

This work has been partially supported by the CICYT PRONTIC program project TIC94-0512.

References

1. Allen, J.F. "Maintaining Knowledge about Temporal Intervals". In Communications of the ACM. Vol. 26. Num 11. 1983. pp 832-843.

2. Apt, K.R.; Blair, H.A.; Walker, A. "Towards a Theory of Declarative Knowledge". In Foundations of Deductive Databases and Logic Programming (J.Minker ed.). Morgan Kaufmann. 1988. pp 89-148.

3. Böhlen, M. "Managing Temporal Knowledge in Deductive Databases". PhD thesis, Swiss Federal Institute of Technology. Zürich, 1994.

4. Bry, F.; Manthey, R.; Martens, R. “Integrity Verification in Knowledge Bases”. ECRC Report D.2.1.a, München, April 1990, 26 p.

5. Chomicki, J. "History-less Checking of Dynamic Integrity Constraints". In the 8th Int. Conf. on Data Engineering, IEEE Computer Society Press. Phoenix AZ, February, 1992. pp 557-564.

6. Chomicki, J. "Efficient Checking Encoding of Temporal Integrity Constraints Using Bounded History Encoding". ACM Transactions on Database Systems. June, 1995. pp 149-186.

7. Gertz, M.; Lipeck, U.W. "“Temporal” Integrity Constraints in Temporal Databases". Proc. of the International Workshop on Temporal Databases. Zürich. (Clifford/Tuzhilin Eds). Springer-Verlag, September, 1995. pp 77-92.

8. Jensen, C.S.; Clifford, J.; Gadia, S.K.; Segev, A.; Snodgrass, R.T. "A Glossary of Temporal Database Concepts". Proc. SIGMOD-RECORD. Vol. 21. 1992.

9. Kline, N. "An Update of the Temporal Database Bibliography ". Proc. SIGMOD-RECORD. Vol. 22. Num. 4. 1993.

10. Kowalski, R.; Sergot, M. “A Logic-Based Calculus of Events” New Generation Computing. Vol 4, Num 1. February, 1986. pp 67-95. OHMSHA LTD and Springer-Verlag.

11. Lloyd, J.W. "Foundations on Logic Programming". Second edition. Springer, 1987.

12. Mylopoulos, J.; Borgida, A.; Jarke, M.; Koubarakis, M. "Telos: Representing Knowledge About Information Systems". ACM Transactions on information systems. Vol. 8. Num 4. 1990. pp 324-362.

13. Martín, C.; Sistac, J. "Integrity Constraints Checking in Historical Deductive Databases". Proc. of the 5th. Int. Workshop on the Deductive Approach to Information Systems and Databases. Aiguablava, 1994. pp 299-324.

14. Martín, C.; Sistac, J. "A Method for Integrity Constraint Checking in Temporal Deductive Databases". To appear in Proc. of the 3th. Int. Workshop on Temporal Representation and Reasoning. Florida, 1996.

15. Nicolas, J.M. "Logic for Improving Integrity Checking in Deductive Databases". In Gallaire, H.; Minker, J. Eds. "Logic and databases". Plenum Press. 1978, pp 325-344.

16. Olivé, A. "On the Design and Implementation of Information Systems from Deductive Conceptual Models". Proc. of the 15th. Int. Conf. on VLDB'89, pp 3-11.

17. Olivé, A. "Integrity Constraints Checking in Deductive Databases". Proc. of the 17th. Int. Conf on VLDB'91. pp 513-523.

18. Plexousakis, D. "Integrity Constraint and Rule Maintenance in Temporal Deductive Knowledge Bases". Proc. of the 19th. Int. Conf. on VLDB'93. pp 146-157.

19. Sadri, F.; Kowalski, R. “Variants of the Event Calculus”. Proc. of the 12th Int. Conf. on Logic Programming, 1995. pp 67-81.

20. Sarda, N.L. "HSQL: A Historical Query Language". In [25], pp 110-140.

21. Snodgrass, R.; Ahn, I. "Temporal Databases". IEEE Computer. Vol 19. Num 9. September, 1986.

22. Sancho, M.R; Olivé, A. "Deriving Transactions Specifications from Deductive Conceptual Models of Information Systems". Proc. of CAiSE'94 conference. 1994, pp 311-324.

23. Sripada, S.M.; Möller, P. "The Generalized ChronoBase Temporal Data Model". In K. Apt, F. Turini (eds). Meta-Logics and logic programming, MIT Press, 1995.

24. Sripada, S.M. "Efficient Implementation of the Event Calculus for Temporal Deductive Databases". Proc. of the 12th Int. Conf. on Logic Programming, 1995. pp 99-113.

25. Tansel, A.U.; Clifford, J.; Gadia, S.; Jajodia, S.; Segev, A.; Snodgrass, R. "Temporal Databases: Theory, Design and Implementation". Benjamin/Cummings. 1993.

26. Urpí, T.; Teniente, E.; Pastor, J.A.; Mayol, E.; Martín, C. "FOLRE: Towards a System for the Integrated Treatment of Updates and Rule Enforcement in Deductive Databases”. In Proc. of the Sixth ERCIM Database Research Group Workshop on Deductive and Interoperable Databases. Barcelona. Nov. 1994.

27. Van Belleghem, K.; Denecker, M.; De Schreye, D. “Combining Situation Calculus and Event Calculus”. Proc. of the 12th Int. Conf. on Logic Programming, 1995. pp 83-97.

28. Wüthrich, B. "Large Deductive Databases with Constraints". PhD thesis, Swiss Federal Institute of Technology. Zürich, 1991.