Analyzing Temporal Integrity Constraints to Obtain the Minimum Number of Transition Rules

Carme Martín Escofet

Departament de Llenguatges i Sistemes Informàtics,
Universitat Politècnica de Catalunya.
Jordi Girona Salgado 1-3. E-08034 Barcelona, Catalunya.
Tel: 93 401 79 92 FAX: 93 401 70 14
martin@lsi.upc.es

Abstract

The aim of this paper is to obtain advantages for integrity constraint checking in bitemporal deductive databases, analyzing the temporal integrity constraint behaviour. First at all, a new taxonomy of temporal integrity constraints is introduced to achieve a better understanding of why constraints are required and a best way to define them using first order logic. In order to meet these goals, we have studied temporal integrity constraint taxonomies existing on the temporal database area and other deeply related areas such as multiversion databases. These works have been adapted and further developed to cover the scope of bitemporal deductive databases.

Before checking, we study the transaction behaviour, introducing the concept of coherent transaction. This allows us to reject transactions that are not coherent, without more information requirements.

For integrity constraint checking in bitemporal deductive databases, we derive a set of transition rules for each temporal integrity constraint. Then, Standard Linear Deduction with Negation as Failure Resolution is used to verify that a transaction does not violate any constraint before the transaction is allowed to commit. The set of transition rules obtained is very large; therefore, a carefully analysis of the integrity constraints using the taxonomy introduced above drastically reduces the number of rules. Finally, the paper concludes with an example of the application of this new reduced set of transition rules for integrity constraint checking in bitemporal deductive databases.

1 Introduction

This paper introduces a taxonomy of temporal integrity constraints in bitemporal deductive databases, namely *bt-ddbs* throughout the paper. A bt-ddb is a deductive database that supports valid 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 [22].

In [26], the state-of-the-art is presented, and [11], [7] and [6] are some particular examples. However, most works have used the term "temporal" only because some temporal reference was required in their integrity constraint definitions. Generally, the aim of all these works was not the study of temporal integrity constraints but their use solving a particular problem in a specific behaviour. The most important analysis of temporal integrity constraints in temporal databases carried out up to date is Michael Böhlen's taxonomy [3]. This classification has been recently reviewed by Richard Snodgrass [23]. Integrity constraints are properties that are supposed to be satisfied by all "legal" databases, and are used to incorporate more real-world semantics in a database schema [25]. Following this idea, an interesting study has been done in the multiversion databases area, which is closely related to that of temporal databases. This study helps us to get a better understanding of why temporal integrity constraints are required. This is Anne Doucet *et al'*s taxonomy [8]. The present paper, thus, focuses on the adaptation and development of the above-mentioned classifications so as to cover the scope of bt-ddbs.

Conceptually, a bt-ddb is a deductive database with a bitemporal dimension. A deductive database D consisting 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 extensional database* (EDB) whereas the set of deductive rules is called *intensional database* (IDB) [4]. Facts, rules and integrity constraints have been formulated in first order logic language.

In the bitemporal dimension *time points* belong to a temporal domain τ [24]. τ is isomorphic to the set of natural numbers and is assumed to be ordered such that for two time points t_i and t_j, either t_i is before t_j or t_j is before t_i. Set τ is used as the basis to incorporate temporal dimensions into the deductive database using a valid time start point, a valid time end point, a transaction time start point and a transaction time end point.

Following the semantics of the concept of *now* in temporal databases [5], we add some time point concepts to the temporal domain τ. *Current* denotes the current-time variable for transaction time and *current* is the constant obtained by evaluating the current-time variable in a specified time point. The valid time variable that indicates that a fact is currently valid is

denoted by *Now*. The transaction time variable that indicates that the database still records a fact is denoted by *Until_changed*. In addition, the last concepts we require will be the initial valid time point vt_0 and the initial transaction time point tt_0.

For consistency checking in bt-ddbs, we use an extension of our integrity constraint checking method ([17], [18] and [19]), based on the "Events Method" [21]. It consists in using transition and event rules, which explicitly define the insertions and deletions given by a database update. In [17], we present our method for only a valid time dimension and in [18] for a bitemporal dimension with very restrictive assumptions. In [19], we introduce the idea of our method evolution using a case study of temporal integrity constraints. In [20], we explain the temporal integrity constraint taxonomy used in this paper. Our transition rules range over all the possible cases in which an update could violate a temporal integrity constraint. For this reason, we obtain a big set of transition rules. Therefore, in this paper we apply the obtained results from temporal integrity constraint taxonomy [20] to our transition rules to simplify or eliminate them.

The paper is organized as follows. Next section analyses other existing taxonomies of temporal integrity constraints and explains our new taxonomy in bt-ddbs. Section 3 describes the transaction concepts, which need our checking method. Section 4 introduces the application of our taxonomy for temporal integrity constraint checking in bt-ddbs. Finally, section 5 provides some conclusions.

2 Temporal Integrity Constraints

A temporal 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. For the sake of uniformity, an inconsistency predicate *ic* is associated with each temporal integrity constraint such as in [9] and [16]. Thus, constraints have the same form as a deductive rule:

$$ic \leftarrow l_1 \wedge ... \wedge l_n \qquad with\ n \geq 1.$$

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.

An integrity constraint is made up of basic predicates, derived predicates and comparison predefined predicates. Comparison predefined predicates can be applied to basic and derived

predicates and constants, and they can express temporal conditions. Relationships between valid and transaction time in conditions, as in [14], are not considered. The Allen's thirteen temporal predicates [1] can be defined in terms of start and end points [2]. This is also the way in which they are to be used in the present work. We consider comparison predefined predicates more intuitive than Allen's predicates and they are equivalent.

Obviously, these thirteen temporal predicates can be combined among themselves to express complex conditions. For example, sometimes we are interested in *A during B*, but also in the fact that *A starts B, A ends B* and *A equals B* and that we can use them all to express this. Moreover, in our case we simply put $A.starts \geq B.starts$ and $A.end \leq B.end$.

In this section, firstly, we explain Michael Böhlen's temporal integrity constraint taxonomy [3]. Secondly, we describe Anne Doucet *et al's* taxonomy [8]. To conclude, we present our taxonomy to adapt and further develop these classifications to cover the scope of bt-ddbs.

2.1 Michael Böhlen's Taxonomy

Michael Böhlen's taxonomy [3] classifies temporal integrity constraints depending on the type of the temporal database that is used.

Nontemporal integrity constraints denote traditional integrity constraints, ie. integrity constraints not dealing with time. *Temporal integrity constraints* denote integrity constraints dealing with valid time, transaction time, or both of them. Thus, temporal integrity constraints can be classified into three new groups depending on whether their definition refers to valid time, transaction time or both. Each of these three new groups can be further divided into two new. To extract the tuples, which are valid at a defined time from a relation, the *valid time snapshot operation* is applied. Similarly, to extract the tuples, which are current at a defined time from a relation, the *transaction time snapshot operation* is applied [15]. A *database state* corresponds to the snapshot database [13] defined by the application of the snapshot operation to all database relations. As is described in [3] the temporal integrity constraints are:

- **Transaction Time Integrity Constraints**

Transaction time integrity constraints are defined for transaction time databases. These

constraints are divides into static and dynamic. *Static integrity constraints* are constraints that must hold at every state of a transaction time database while *dynamic integrity constraints* relate and restrict arbitrary states of a transaction time database. *Transition integrity constraints* are a well-known subset of dynamic integrity constraints that put restrictions on succeeding database states.

- **Valid Time Integrity Constraints**

Valid time integrity constraints are defined for valid time databases. They are divided into intrastate and interstate. *Intrastate integrity constraints* enforce the consistency of all snapshots of a valid time database while *interstate integrity constraints* relate and restrict arbitrary snapshots of a valid time database.

- **Bitemporal Integrity Constraints**

Bitemporal integrity constraints are the most complex type of integrity constraints, specifically, interelement integrity constraints and they are defined for bitemporal databases. They are divided into intraelement and interelement. *Intraelement integrity constraints* guarantee the consistency of every state of a bitemporal database while *interelement integrity constraints* relate and constrain different states in a bitemporal database.

2.2 Anne Doucet 3et *al's* Taxonomy

Regarding Anne Doucet et *al's* taxonomy [8], it focuses on semantic characteristics of integrity constraints that are useful when considering for temporal integrity constraints in bt-ddbs.

- **Historical integrity constraints.**

These constraints imply that certain events can only take place if some conditions are verified.

In our taxonomy, these integrity constraints are referred to as *condition integrity constraints* because the *historical* word has valid time connotations in temporal databases.

- **Synchronous integrity constraints.**

These constraints entail that events always happen at the same time.

In our work, these constraints are further divided into *synchronous in valid time*, when valid times are required to be equal, and *synchronous in transaction time*, when transaction times are required to be equal.

- **Periodic integrity constraints.**

In this type of constraints, events happen every week, every month, every year, etc.

Here, these constraints are further divided into *periodic in valid time*, when vt_0 and *Now* are used to express periodicity and *periodic in transaction time*, when tt_0, *Current* and *Until_changed* are used to express periodicity.

- **Dependent on a deadline integrity constraints.**

In these constraints events depend on a deadline.

In our work, we prefer to distinguish between integrity constraints that are *dependent on a deadline in valid time*, when the deadline is expressed with valid time variables, and *dependent on a deadline in transaction time*, when the deadline is expressed with transaction time variables.

2.3 Our Taxonomy

Our taxonomy combines the four temporal criteria of Anne Doucet *et al's* with the Michael Böhlen's taxonomy to get a better understanding of temporal integrity constraints in bt-ddbs. The Michael Böhlen's taxonomy is interesting when only one transaction database or valid time database is used. However, in our opinion, temporal integrity constraints in bitemporal databases can be any of the Michael Böhlen's taxonomy integrity constraints, depending on one of the following: whether they have no time reference, or they only have transaction time references, or they only have valid time references, or they have both time references. Taking into account this situation, our integrity constraint taxonomy comprises only four groups, where we apply seven criteria (adapted from the four of Anne Doucet *et al's*) to know properties that help us to improve our integrity constraint checking method [19]. In figure 2.1, we present the relationship between our four groups and the six basic categories of Michael Böhlen combined with our seven criteria. This table summarizes the possible criteria applicable to each group. The results presented in this figure are going to be explained in the following subsections for each group.

Criteria ↓ / Integrity Constraint Groups	First Group Current	Second Group in Transaction Time			Third Group in Valid Time	Fourth Group Bitemporal
Michael Böhlen Integrity Constraints	Static	Intrastate	Intraelement	Dynamic Transition	Interstate	Interelement
Condition Integrity Constraints	√	√	√	√	√	√
Valid Time Synchronous Integrity Constraints	} √			√		(√)
Transaction Time Synchronous Integrity Constraints					√	(√)
Valid Time Periodic Integrity Constraints					√	√
Transaction Time Periodic Integrity Constraints				√		√
Valid Time Dep. Deadline Integrity Constraints					√	√
Transaction Time Dep. Deadline Integrity Constraints				√		√

Figure 2.1. *Our integrity constraints taxonomy for bt-ddbs.*

A first version of our taxonomy was presented in [20] and its utility for integrity constraint checking in bt-dbs is explained in section 4.

2.3.1 Example.

In order to classify the proposal presented in the current work, a simple example is introduced that will be used throughout the paper. This example is based on a faculty of computer science, where courses are offered and students register in these courses.

The base predicates used are the following:

- *course(C,P,T_{vs},T_{ve},T_{ts},T_{te})* expresses that "*C* is a course at cost *P*, offered by the university from valid time T_{vs} to T_{ve} and from transaction time T_{ts} to T_{te}".
- *student(S,T_{vs},T_{ve},T_{ts},T_{te})* expresses that "*S* is a student at university from valid time T_{vs} to T_{ve} and from transaction time T_{ts} to T_{te}.".
- *registration(S,C,T_{vs},T_{ve},T_{ts},T_{te})* expresses that "student S is enrolled in course *C* from valid time T_{vs} to T_{ve} and from transaction time T_{ts} to T_{te}".

For the sake of simplicity, the simple deductive rule used is the following:

- *classes(C,T_{vs},T_{ve},T_{ts},T_{te})* ← *course(C,P,T_{vs},T_{ve},T_{ts},T_{te})* expresses that "*C* is a course offered by the university from valid time T_{vs} to T_{ve} and from transaction time T_{ts} to T_{te}".

2.3.2 First Group: Current Integrity Constraints.

This first group, namely current, is characterized by making reference only to *current* transaction time and a given valid time. This first group is formed by static, intrastate and intraelement integrity constraints of Michael Böhlen's taxonomy and some dynamic (and transition) integrity constraints excluded from the second group. About Anne Doucet et *al.* criteria, current integrity constraints can only have conditions without time references because if they were periodic or dependent on a deadline in transaction time, then they would refer to a past transaction time point and they would change into the second group. In addition, if they were periodic or dependent on a deadline in valid time, then they would refer to a past valid time point and they would change into the third group. Furthermore, this group can have valid time and transaction time synchronisms, but only when both are referred at the same time.

Formally, a current integrity constraint will be:

ic $\leftarrow$ p(X,T_{vs},T_{ve},T_{ts},T_{te}) [$\wedge$ q(Y,T_{vs},T_{ve},T_{ts},T_{te}) ...] $\wedge$ {conditions / valid and transaction time synchronisms}.

Where the square brackets [] indicate that it is possible to add more than namely options. Where the curly brackets {} indicate that it is possible to choose one or more than namely options between the curly brackets.

Graphically, we show current integrity constraints in figure 2.2.

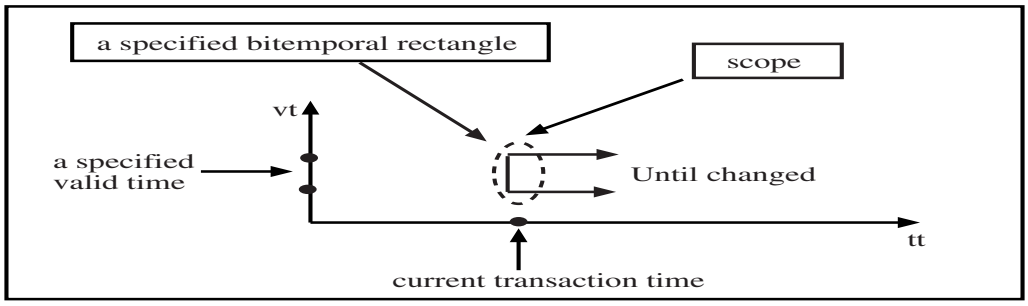


Figure 2.2. *Current integrity constraint scope.*

Example

An example of a current integrity constraint could be *ic1*:

▫ *ic1* $\leftarrow$ *course(C,P,*T_{vs}*,*T_{ve}*,*T_{ts}*,*T_{te}*)* $\wedge$ *P<10000.*

Ic1 enforces the property that "for course *C* to be offered, the cost *P* must be greater than or equal to 10000".

Before introducing the following group, we analyze some transaction time particularities that could occur in integrity constraints.

2.3.3 Past Transaction Time References

If an integrity constraint definition was a future temporal reference, it would be changed by a past temporal reference (see figure 2.3), to unify the comparison predefined predicates that are used in the following section.

Formally, considering $T1_{te}$ as *Current* transaction time,

$$icx \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge T1_{te} \geq T_{te}$$

$$icy \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge T1_{te} > T_{te}$$

then *icx* and *icy* will be rewritten, respectively, to obtain:

$$icx \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge T_{te} \leq T1_{te}$$

$$icy \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge T_{te} < T1_{te}$$

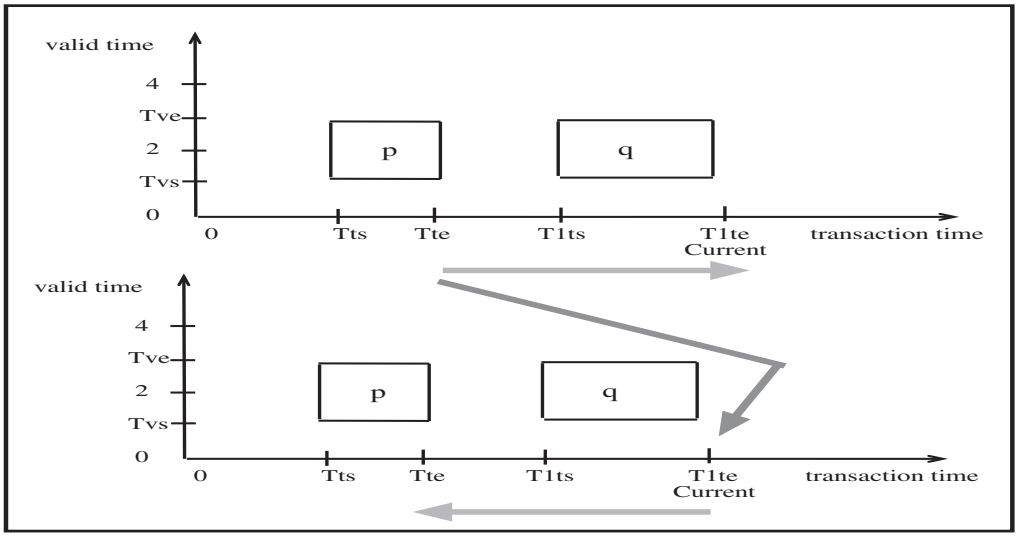


Figure 2.3. *Transformation of future integrity constraints in past integrity constraints.*

2.3.4 Integrity Constraint Condition Analysis

If an integrity constraint refers to a transaction time condition for both transaction time start points and transaction time end points, without periodicity and deadline, always this constraint can be rewritten using several rules. That is the case of the examples of *ica*, for transaction time integrity constraints (second group), and *icb*, for bitemporal integrity constraints (fourth group).

$ica \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge T_{ts} \leq T1_{ts} \wedge T1_{te} \leq T_{te}$

$icb \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T1_{vs},T1_{ve},T1_{ts},T1_{te}) \wedge T_{ts} \leq T1_{ts} \wedge T1_{te} \leq T_{te} \wedge T_{ve} \leq T1_{vs}$

$ica \leftarrow ica_i \quad i = 1,\ldots,4$

$ica_1 \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge T_{ts} < T1_{ts} \wedge T1_{te} < T_{te}$

$ica_2 \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge T_{ts} < T1_{ts} \wedge T1te = Tte$

$ica_3 \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge T_{ts} = T1_{ts} \wedge T1_{te} < T_{te}$

$ica_4 \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge T_{ts} = T1_{ts} \wedge T1_{te} = T_{te}$

$icb \leftarrow icb_i \quad i = 1,\ldots,4$

$icb_1 \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T1_{vs},T1_{ve},T1_{ts},T1_{te}) \wedge T_{ts} < T1_{ts} \wedge T1_{te} < T_{te} \wedge T_{ve} \leq T1_{vs}$

$icb_2 \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T1_{vs},T1_{ve},T1_{ts},T1_{te}) \wedge T_{ts} < T1_{ts} \wedge T1_{te} = T_{te} \wedge T_{ve} \leq T1_{vs}$

$icb_3 \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T1_{vs},T1_{ve},T1_{ts},T1_{te}) \wedge T_{ts} = T1_{ts} \wedge T1_{te} < T_{te} \wedge T_{ve} \leq T1_{vs}$

$icb_4 \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T1_{vs},T1_{ve},T1_{ts},T1_{te}) \wedge T_{ts} = T1_{ts} \wedge T1_{te} = T_{te} \wedge T_{ve} \leq T1_{vs}$

Thus, constraints, as *ica1*, *ica2* and *ica3*, are the integrity constraints of the second group.

Integrity constraints, as *ica4*, are going to be classified in the first group because they should be checked at transaction time *Current* = $T_{ts} = T1_{ts}$.

Integrity constraints, as *icb1*, *icb2* and *icb3*, are going to be in the fourth group.

Finally, integrity constraints, as *icb4*, are included in the third group because we have eliminated the past transaction time references of *icb4*, they are in *icb1*, *icb2* and *icb3*. Now *icb4* only has to be checked at transaction time *Current* = $T_{ts} = T1_{ts}$.

In general, for every integrity constraint with *k* smaller than or equal to transaction time conditions, we classify 2^k-1 integrity constraints in their corresponding group and the number 2^k integrity constraint goes to the previous group.

With this new set of integrity constraints, we manage to check first the integrity constraint of either the first group instead of the second or the third group instead of the fourth. If the integrity constraint is violated with this first check, it is not necessary to continue checking in the other groups.

2.3.5 Second Group: Transaction Time Integrity Constraints.

This second group, namely transaction time, is characterized by referring to a given valid time, and a past and current transaction times. This second group is formed by some dynamic and transition integrity constraints of Michael Böhlen's taxonomy. A transaction integrity constraint could be a condition, synchronous in valid time (the same valid time and different transaction time), periodic in transaction time and dependent on a deadline in transaction time integrity constraints. This is due to fact that if they were synchronous in transaction time, periodic in valid time or dependent on a deadline in valid time, then they would refer to a past valid time point and they would change into the third or fourth group.

Formally, a transaction integrity constraint will be:

ic $\leftarrow$ p(X,T_{vs},T_{ve},$T1_{ts}$,$T1_{te}$) [$\wedge$ q(Y,T_{vs},T_{ve},$T2_{ts}$,$T2_{te}$) ...] $\wedge$ {conditions / valid time synchronism / transaction time periodicity / transaction time deadline}.

Graphically, we show transaction time integrity constraints in figure 2.4.

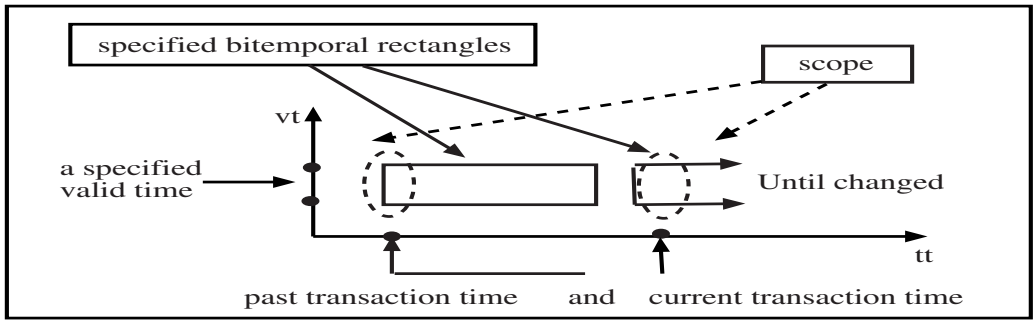


Figure 2.4. *Transaction time integrity constraint scope.*

Example

An example of a transaction time integrity constraint could be *ic2*:

▫ $ic2 \leftarrow student(S,T_{vs},T_{ve},T_{ts},T_{te}) \wedge registration(S,C,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge T_{ts} \leq T1_{ts} \wedge T_{te} < T1_{te}$.

Ic2 enforces the property that "if a student *S* is enrolled in a course *C* then this student *S* has to be university student during the course registration".

2.3.6 Third Group: Valid Time Integrity Constraints.

This third group, namely valid time, is characterized by referring to a given transaction time and different valid times. In this group, we have interstate integrity constraints of Michael Böhlen's taxonomy and bitemporal integrity constraints excluded from the fourth group. A valid time integrity constraint could be a condition, synchronous in transaction time (the same transaction times and different valid times), periodic in valid time or dependent on a deadline in valid time integrity constraint. The reason behind is that if they were synchronous in valid time, periodic in transaction time or dependent on a deadline in transaction time, then they would refer to a past transaction time point and they would change into the second or fourth group.

Formally, a valid time integrity constraint will be:

ic $\leftarrow$ p(X,$T1_{vs}$,$T1_{ve}$,T_{ts},T_{te}) [$\wedge$ q(Y,$T2_{vs}$,$T2_{ve}$,T_{ts},T_{te}) ...] $\wedge$ {conditions / transaction time synchronism / valid time periodicity / valid time deadline}.

Graphically, we show valid time integrity constraints in figure 2.5.

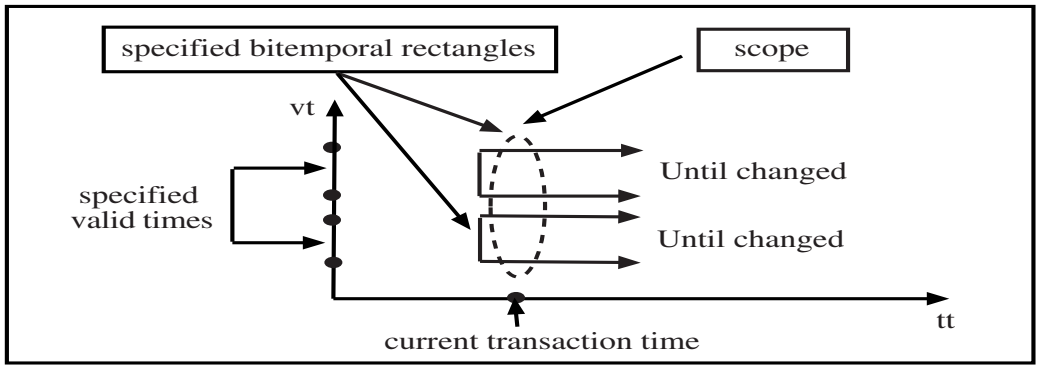


Figure 2.5. *Valid time integrity constraint scope.*

Example

An example of a valid time integrity constraint could be *ic3*:

▫ *ic3* $\leftarrow$ *registration(S,'databases',*T_{vs}*,*T_{ve}*,*T_{ts}*,*T_{te}*)*

$\wedge$ *registration(S,'software engineering',*$T1_{vs}$*,*$T1_{ve}$*,*T_{ts}*,*T_{te}*)* $\wedge$ $T_{vs} \leq T1_{ve}$.

Ic3 enforces the property that "if a student *S* is enrolled in the course *databases*, and this student *S* is also enrolled in the course *software engineering at* the same transaction time, then, student *S* has to be enrolled in *software engineering* at a valid time that is previous to the *databases* registration".

2.3.7 Fourth Group: Bitemporal Integrity Constraints.

This fourth group, namely bitemporal, is characterized by referring to different transaction times and different valid times. Bitemporal integrity constraints that are not in the third group form the fourth group. A bitemporal integrity constraint is the most complex type of integrity constraint and it permits all kind of temporal references in their definition. Then, bitemporal integrity constraints could be a condition, synchronous in valid time but not in transaction time, periodic in valid time or dependent on a deadline in valid time integrity constraint. In addition, they could be synchronous in transaction time but not in valid time, periodic in transaction time or dependent on a deadline in transaction time integrity constraint.

Formally, a bitemporal integrity constraint will be:

ic $\leftarrow$ p(X,$T1_{vs}$,$T1_{ve}$,$T1_{ts}$,$T1_{te}$) [$\wedge$ q(Y,$T2_{vs}$,$T2_{ve}$,$T2_{ts}$,$T2_{te}$) ...] $\wedge$ {conditions / (valid time synchronism or transaction time synchronism) / transaction time periodicity / transaction time deadline / valid time periodicity / valid time deadline}.

Graphically, we show bitemporal time integrity constraints in figure 2.6.

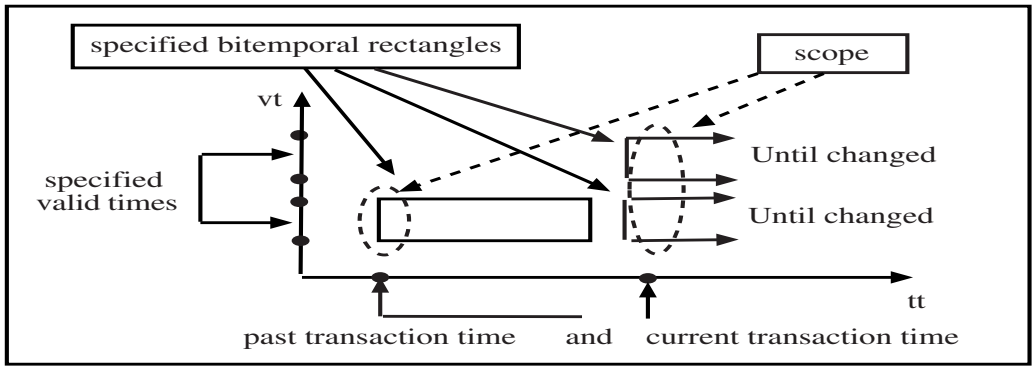


Figure 2.6. *Bitemporal integrity constraint scope.*

Example

An example of a bitemporal time integrity constraint could be *ic4*:

▫ *ic4* $\leftarrow$ *course(C,P,*T_{vs}*,*T_{ve}*,*T_{ts}*,*T_{te}*)* $\wedge$ *course(C,P1,*$T1_{vs}$*,*$T1_{ve}$*,*$T1_{ts}$*,*$T1_{te}$*)* $\wedge$ *P1<P* $\wedge$ $T1_{vs}$<T_{ve} $\wedge$ T_{ts}<$T1_{ts}$-10.

Ic4 enforces the property that "if a course has been offered with a given price, there is only ten days to register both at a reduced price and at a valid time later than originally specified.

3 Transactions

Our aim is to investigate integrity constraint checking in bt-ddbs. A transaction is the logical unit of a database. In our case, a transaction *TR* consists on a set of events: insertions and deletions. Transaction could produce or induce integrity constraint violations in the database. In our previous work ([17], [18] and [19]), we had assumed that the user always introduced only a valid time start point, and the valid time end point was *Now (Until_changed* in this paper). In the present paper, we eliminate this hard restriction. Further, we also eliminate the assumption that in a transaction we cannot insert and delete the same fact. This last restriction is now allowed because we guarantee the coherence of the transaction before checking. These previous assumption eliminations require that we revisit the study of transaction. In the next subsections, we present the reviewed concepts of insertion and deletion, and we also introduce the concept of coherent transaction.

3.1 Insertions

To denote the insertion event we use the symbol: ι. An insertion always happens in the transaction time *Current* and it is going to be valid until a logical deletion was produced. This supposes transaction time can be automatically added when insertion is stored in the bt-ddb. Whereas the insertion operation only has valid time references.

In our computer science faculty example, we can insert a course of *databases* from 4 to 6, and at transaction time 2. The insertion of this fact will be:

ιcourse ('databases', 4, 6), where *Current* time is 2.

In bt-ddbs, there are different situations for an insertion statement described in figure 3.1.

In figure 3.1.a and in 3.1.d the *databases* course never has been offered and we can insert it. In figure 3.1.d., we show the case of an insertion with a specified valid time start, but without to know the valid time end. For example, if we know that the *databases* course begins at 4, but we do not know when it will finish then we can insert the valid time end *Now*:

ιcourse ('databases', 4, Now) or ιcourse ('databases', 4)

Now means that the *databases* course will be offered forever while valid time end cannot be known. When we have a valid time end for the course, we will introduce it instead of *Now*, deleting the old fact and inserting a new one.

In figure 3.1.b the *databases* course has been offered in a different valid time interval and we can insert the new *databases* course without problems.

Finally, in the figure 3.1.c the *databases* course cannot be inserted because this course intersect with the valid time interval given by the insertion.

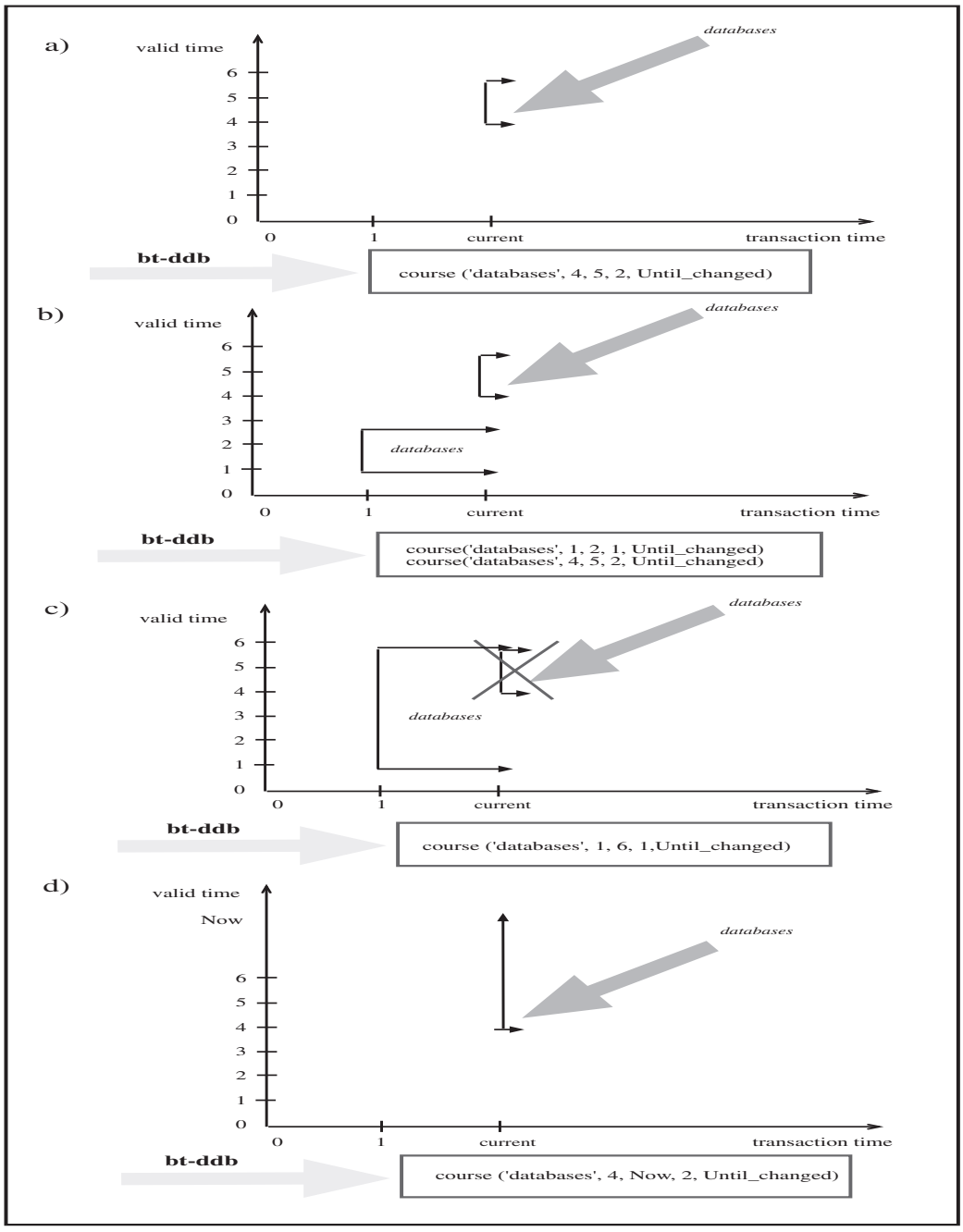


Figure 3.1. *Different situations in bt-ddbs for the insertion of the databases course.*

The intersection between two intervals can be defined by the following deductive rule:

$intersect(T_{ts},T_{te},T1_{ts},T1_{te},T2_{ts},T2_{te}) \leftarrow int_i\,(T_{ts},T_{te},T1_{ts},T1_{te},T2_{ts},T2_{te}) \qquad i = 1,\ldots,4$

$int_1(T_{ts},T_{te},T1_{ts},T1_{te},T2_{ts},T2_{te}) \leftarrow T1_{ts} \geq T2_{ts} \wedge T1_{te} \leq T2_{te} \wedge T_{ts}=T1_{ts} \wedge T_{te}=T1_{te}.$

$int_2(T_{ts},T_{te},T1_{ts},T1_{te},T2_{ts},T2_{te}) \leftarrow T1_{ts} < T2_{ts} \wedge T1_{te} > T2_{te} \wedge T_{ts}=T2_{ts} \wedge T_{te}=T2_{te}.$

$int_3(T_{ts},T_{te},T1_{ts},T1_{te},T2_{ts},T2_{te}) \leftarrow T1_{ts} < T2_{ts} \wedge T1_{te} < T2_{te} \wedge T1_{te} \geq T2_{ts} \wedge T_{ts}=T2_{ts} \wedge T_{te}=T1_{te}.$

$int_4(T_{ts},T_{te},T1_{ts},T1_{te},T2_{ts},T2_{te}) \leftarrow T1_{ts} > T2_{ts} \wedge T1_{te} > T2_{te} \wedge T1_{ts} \leq T2_{te} \wedge T_{ts}=T1_{ts} \wedge T_{te}=T2_{te}.$

In figure 3.2 the four intersection cases are graphically showed.

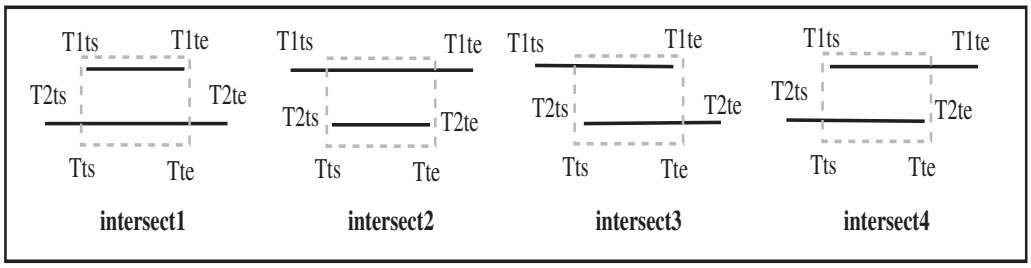


Figure 3.2. *Graphical representation of the intersection between two time intervals.*

3.2 Deletions

To denote the deletion event we use the symbol: δ. A deletion always happens in the transaction time *Current*. This supposes transaction time can be automatically added when deletion is stored in the bt-ddb. Whereas the deletion operation is always a logical deletion. A deletion has only to close the open bitemporal rectangle for the specified element of the bt-ddb.

In our computer science faculty example, we can delete the course of *databases*. The deletion of this fact will be:

δcourse ('databases',4,6), where *Current* time is 3.

In bt-ddbs, there exist different situations for a deletion statement described in figure 3.3.

In figure 3.3.a the *databases* course has been offered from 4 to 6 and then we can delete it. In figure 3.3.b the *databases* course never has been offered and we cannot delete it.

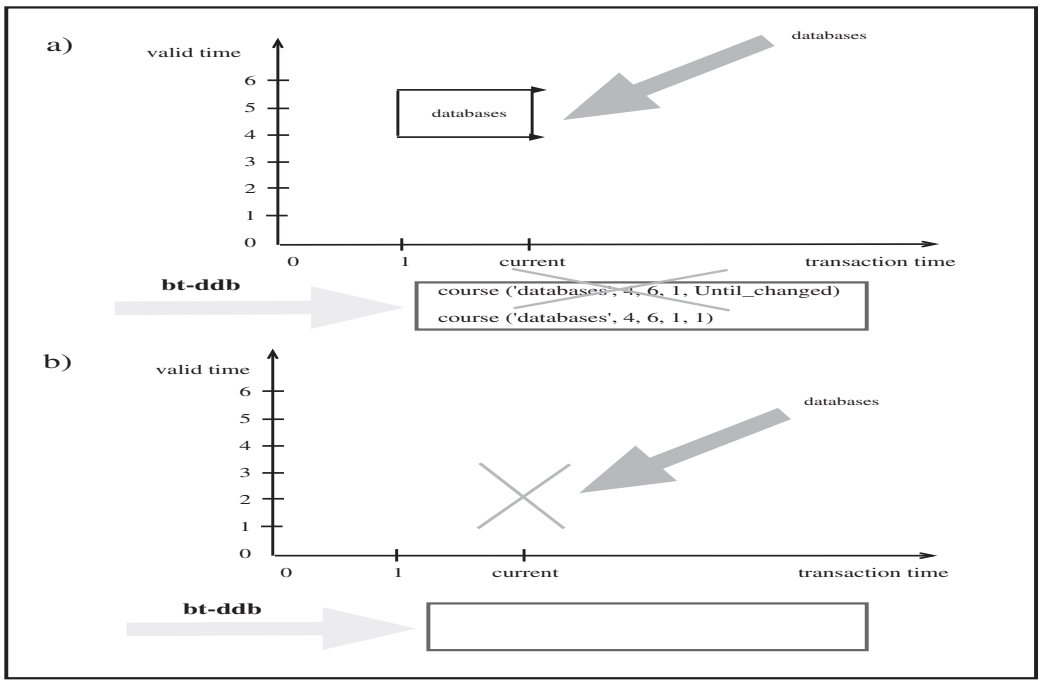


Figure 3.3. *Different situations in bt-ddb for the deletion of the databases course.*

3.3 Coherent Transactions

Our transactions will be a set of insertion and deletion operations. Thus, we consider a modification as a deletion followed by an insertion.

Before to check the consistency would be very interesting to verify the transaction coherence. That is, no operation interactions in the transaction produce problems each other.

In figure 3.4 we show some incoherence transaction examples.

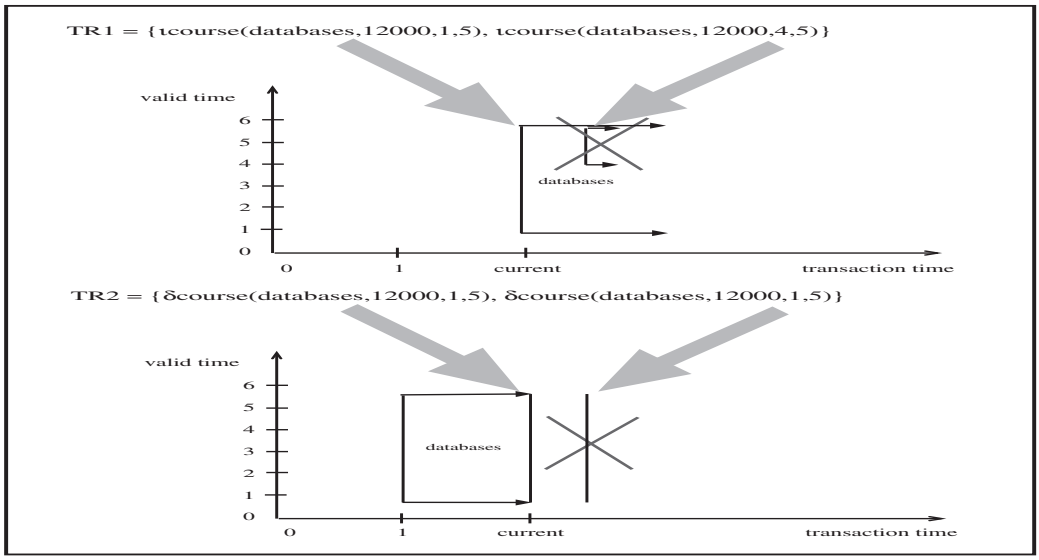


Figure 3.4. *Incoherence transaction examples.*

Formally, let $TR = \{t_1, t_2, ..., t_n\}$ be a transaction, where $t_1, t_2, ..., t_n$ are insertions or deletions. For all the possible transaction constructions, we obtain the following results:

$\forall\ t_i, t_j \in TR \wedge i<j$:

A1) $t_i = \iota(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \iota(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} = t1_{vs} \wedge t_{ve} = t1_{ve}$

If t_j is the same fact insertion t_i then the transaction is incoherent and t_j will be deleted.

A2) $t_i = \iota(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \iota(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} < t1_{vs} \wedge t_{ve} > t1_{ve}$

or $t_i = \iota(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \iota(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} > t1_{vs} \wedge t_{ve} < t1_{ve}$

or $t_i = \iota(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \iota(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} < t1_{vs} \wedge t_{ve} < t1_{ve}$

$\wedge\ t_{ve} \geq t1_{vs}$

or $t_i = \iota(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \iota(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} > t1_{vs} \wedge t_{ve} > t1_{ve}$

$\wedge\ t_{vs} \leq t1_{ve}$

If t_j is an insertion of a fact with valid time intersection with t_i fact insertion then the transaction is incoherent and t_i and t_j will be deleted because someone is mistaken but we do not know which one.

B1) $t_i = \iota(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \delta(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} = t1_{vs} \wedge t_{ve} = t1_{ve}$

If t_j is a deletion of a fact that t_i is inserting then the transaction is incoherent and t_i and t_j will be deleted.

B2) $t_i = \iota(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \delta(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} < t1_{vs} \wedge t_{ve} > t1_{ve}$

or $t_i = \iota(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \delta(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} > t1_{vs} \wedge t_{ve} < t1_{ve}$

or $t_i = \iota(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \delta(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} < t1_{vs} \wedge t_{ve} < t1_{ve}$

$\wedge\ t_{ve} \geq t1_{vs}$

or $t_i = \iota(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \delta(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} > t1_{vs} \wedge t_{ve} > t1_{ve}$

$\wedge\ t_{vs} \leq t1_{ve}$

If t_j is a deletion of a fact with valid time intersection with t_i fact insertion then the transaction is incoherent and t_j will be deleted.

C1) $t_i = \delta(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \iota(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} = t1_{vs} \wedge t_{ve} = t1_{ve}$

This is the case of a null modification. Thus the transaction is incoherent and t_i and t_j will be deleted.

C2) $t_i = \delta(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \iota(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} < t1_{vs} \wedge t_{ve} > t1_{ve}$

or $t_i = \delta(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \iota(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} > t1_{vs} \wedge t_{ve} < t1_{ve}$

or $t_i = \delta(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \iota(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} < t1_{vs} \wedge t_{ve} < t1_{ve}$

$\wedge t_{ve} \geq t1_{vs}$

or $t_i = \delta(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \iota(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} > t1_{vs} \wedge t_{ve} > t1_{ve}$

$\wedge t_{vs} \leq t1_{ve}$

This is the modification case and the transaction is coherent.

D1) $t_i = \delta(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \delta(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} = t1_{vs} \wedge t_{ve} = t1_{ve}$

If t_j is the same fact deletion t_i then the transaction is incoherent and t_j will be deleted.

D2) $t_i = \delta(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \delta(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} < t1_{vs} \wedge t_{ve} > t1_{ve}$

or $t_i = \delta(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \delta(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} > t1_{vs} \wedge t_{ve} < t1_{ve}$

or $t_i = \delta(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \delta(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} < t1_{vs} \wedge t_{ve} < t1_{ve}$

$\wedge t_{ve} \geq t1_{vs}$

or $t_i = \delta(a_1, ..., a_n, t_{vs}, t_{ve}) \wedge t_j = \delta(a_1, ..., a_n, t1_{vs}, t1_{ve}) \wedge t_{vs} > t1_{vs} \wedge t_{ve} > t1_{ve}$

$\wedge t_{vs} \leq t1_{ve}$

If t_j is a deletion of a fact with valid time intersection with t_i fact deletion then the transaction is incoherent and t_i and t_j will be deleted because someone is mistaken but we do not know which one.

4 Temporal Integrity Constraint Checking

The essence of our integrity constraint checking method for bt-ddbs is the temporal transition rule, as in [17] and [18]. However, the complex temporal integrity constraint definitions that are used now [20] require an extension of our previous work. A temporal transition rule describes all the possible cases in which a temporal integrity constraint can be violated by transaction statements.

In section 3, we have explained how to express insertions and deletions in a transaction. An insertion $\iota p(x,t_{vs},t_{ve})$ has a given valid time but we also know the transaction time. The event that includes valid and transaction time is:

$\iota^* p(x,t_{vs},t_{ve},t_{ts},t_{te})$ where t_{ts}=*Current* and t_{te}=*Until_changed*

A deletion $\delta p(x,t_{vs},t_{ve})$ also requires transaction time:

$\delta^* p(x,t_{vs},t_{ve},t_{ts},t_{te})$ where t_{ts}=*Current* and t_{te}=*Until_changed*

The ι^* and δ^* predicates are introduced and described in detail in [19].

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 we get a new rule called a **transition rule**. This rule defines predicate p' in the updated bt-ddb in terms of transaction time point *Current- Δt* of the predicates appearing in the body of the rule and the events (transaction insertions, transaction deletions, induced insertions of derived predicates and induced deletions of derived predicates) that occur at transaction time point *Current*.

More precisely, if l_i' is an ordinary positive literal $q_i'(X_i,T_{vs},T_{ve},Current,Until_changed)$ we replace it with:

$(q_i(X_i,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg \delta^* q_i(X_i,T_{vs},T_{ve},Current,Until_changed))$

$\vee\ \iota^* q_i(X_i,T_{vs},T_{ve},Current,Until_changed)$

And if l_i' is an ordinary negative literal $q_i'(X_i,T_{vs},T_{ve},Current,Until_changed)$ we replace it with:

$(\neg q_i(X_i,T_{vs},T_{ve},T_{ts},T_{ve}) \wedge \neg \iota^* q_i(X_i,T_{vs},T_{ve},Current,Until_changed))$

$\vee \delta^* q_i(X_i,T_{vs},T_{ve},Current,Until_changed)$

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:

$O(l_i')$ $= (q_i(X_i,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg \delta^* q_i(X_i,T_{vs},T_{ve},Current,Until_changed))$ *if* $l_i' = q_i'(X_i,T_{vs},T_{ve},Current,Until_changed)$

$= (\neg q_i(X_i,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg \iota^* q_i(X_i,T_{vs},T_{ve},Current,Until_changed))$ *if* $l_i' = \neg q_i'(X_i,T_{vs},T_{ve},Current,Until_changed)$

$= l_i$ *if* l_i *is evaluable*

$N(l_i')$ $= \iota^* q_i(X_i,T_{vs},T_{ve},Current,Until_changed)$ *if* $l_i' = q_i'(X_i,T_{vs},T_{ve},Current,Until_changed)$

$= \delta^* q_i(X_i,T_{vs},T_{ve},Current,Until_changed)$ *if* $l_i' = \neg q_i'(X_i,T_{vs},T_{ve},Current,Until_changed)$

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 *Current- Δt* 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 $N(l_i') \rightarrow \neg l_i$.

With this notation, the equivalence become:

$\forall X,T_{VS},T_{Ve}\ (p'(X,T_{VS},T_{Ve},Current,Until_changed) \leftrightarrow$

$O(p'(X,T_{VS},T_{Ve},Current,Until_changed)) \vee N(p'(X,T_{VS},T_{Ve},Current,Until_changed))$.

$\forall X,T_{VS},T_{Ve}\ (\neg p'(X,T_{VS},T_{Ve},Current,Until_changed) \leftrightarrow$

$O(\neg p'(X,T_{VS},T_{Ve},Current,Until_changed)) \vee N(\neg p'(X,T_{VS},T_{Ve},Current,Until_changed)))$.

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

$p'(X,T_{VS},T_{Ve},Current,Until_changed) \leftarrow p_j'(X,T_{VS},T_{Ve},Current,Until_changed)$

With $j = 1, ..., 2^k$, where k is the number of ordinary literals in the p' rule.

We are aware of the resulting amount of transition rules and we present afterwards in this paper some simplifications to drastically reduce them. These new simplifications are different to simplifications presented in [18] because our previous assumptions have been eliminated.

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 minor than *Current*, and thus we can eliminate this transition rule. We only have an exception: when the transaction time is tt_0. In this case, if we have an integrity constraint that only has negative literals then we will check this first transition rule.

Following the taxonomy presented in section 2, we show how temporal transition rules deal with different examples of different types of temporal integrity constraints in bt-ddbs.

In next subsections, for every type of integrity constraint we study all their possible constructions. Case 1 considers integrity constraints with that only have positive literals. Case 2 analyses integrity constraints with positive and negative literals. Finally, case 3 deals with integrity constraints that only have negative literals. We obtain the same results for all three cases when applied within the same group. The difference lies in the type of integrity constraint we analyse. Figure 4.1 shows the result of this study for integrity constraints with two literals and thus, with four transition rules.

Integrity Constraints	First Group Current	Second Group Transaction Time	Third Group Valid Time	Fourth Group Bitemporal
Case1: Only positive literals	ic4	ic2, ic3	ic4	ic2, ic3
Case2: Positive and negative literals	ic4	ic2, ic3	ic4	ic2, ic3
Case3: Only negative literals	ic4	ic2, ic3	ic4	ic2, ic3

Figure 4.1. *Transition rules obtained for every type of integrity constraints.*

Furthermore, analysing the transaction time conditions we can find more transition rule to be eliminated. For instance, the transition rules with the following conditions can be eliminated:

Current ≺ T, Current ≺ Current,

Until_changed ≺ T, Until_changed ≺ Until_changed,

Now ≺ T and Now ≺ Now.

After to study all the possible constructions for an integrity constraint we analyze all the possible values for transaction time variables. Finally we deal with deductive rules in a special subsection and present an application example of our method.

4.1 Current Integrity Constraints

Ic1 is a current integrity constraint. It should be checked at *Current* transaction time and must hold from T_{vs} to T_{ve}. *Ic1* could be violated only if at a *Current* transaction time:

$ic1_2' \leftarrow \iota^* course(C,P,T_{vs},T_{ve},Current,Until_changed) \wedge P \prec 10000.$

$ic1 \leftarrow course(C,P,T_{vs},T_{ve},T_{ts},T_{te}) \wedge P \prec 10000.$

This group always has one transition rule only, the last one, to check the consistency of the database.

Case1. Let be a current integrity constraint with all their literals positives:

$ic \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T_{vs},T_{ve},T_{ts},T_{te}) \wedge conditions.$

Their transition rules are:

$ic_2' \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg\delta^* p(X,T_{vs},T_{ve},Current,Until_changed) \wedge$

$\iota^* q(Y,T_{vs},T_{ve},Current,Until_changed) \wedge conditions.$

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if q is inserted with a transaction time start *Current* then

the transaction time start of p will be definitively different.

$ic_3' \leftarrow \iota^*p(X,T_{vs},T_{ve},Current,Until_changed) \wedge q(Y,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg\delta^*q(Y,T_{vs},T_{ve},Current,Until_changed) \wedge conditions.$

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if p is inserted with a transaction time start *Current* then the transaction time start of q will be definitively different.

$ic_4' \leftarrow \iota^*p(X,T_{vs},T_{ve},Current,Until_changed) \wedge \iota^*q(Y,T_{vs},T_{ve},Current,Until_changed) \wedge conditions.$

This transition rule could violate the integrity constraint because if p and q are inserted with a transaction time start *Current* and a transaction time end *Until_changed* then both transaction times start and end for p and q will be the same. In this case, it should check the conditions of the integrity constraint.

Case2. Let be a current integrity constraint with their literals positives and negatives:

$ic \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg q(Y,T_{vs},T_{ve},T_{ts},T_{te}) \wedge conditions.$

Their transition rules are:

$ic_2' \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg\delta^*p(X,T_{vs},T_{ve},Current,Until_changed) \wedge \delta^*q(Y,T_{vs},T_{ve},Current,Until_changed) \wedge conditions.$

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if q is deleted with a transaction time start *Current* for $\neg q$ then the transaction time start of p will be definitively different.

$ic_3' \leftarrow \iota^*p(X,T_{vs},T_{ve},Current,Until_changed) \wedge \neg q(Y,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg\iota^*q(Y,T_{vs},T_{ve},Current,Until_changed) \wedge conditions.$

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if p is inserted with a transaction time start *Current* then the transaction time start of $\neg q$ will be definitively different.

$ic_4' \leftarrow \iota^*p(X,T_{vs},T_{ve},Current,Until_changed) \wedge \delta^*q(Y,T_{vs},T_{ve},Current,Until_changed) \wedge conditions.$

This transition rule could violate the integrity constraint because if p is inserted and q is deleted with a transaction time start *Current* and a transaction time end *Until_changed* then both transaction times start and end for p and $\neg q$ will be the same. In this case, it should

check the conditions of the integrity constraint.

***Case3*.** Let be a current integrity constraint with all their literals negatives:

$ic \leftarrow \neg p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg q(Y,T_{vs},T_{ve},T_{ts},T_{te}) \wedge$ *conditions.*

Their transition rules are:

$ic_2' \leftarrow \neg p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg \iota * p(X,T_{vs},T_{ve},Current,Until_changed) \wedge$

$\delta * q(Y,T_{vs},T_{ve},Current,Until_changed) \wedge$ *conditions.*

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if *q* is deleted with a transaction time start *Current* then the transaction time start of ¬*p* will be definitively different.

$ic_3' \leftarrow \delta * p(X,T_{vs},T_{ve},Current,Until_changed) \wedge \neg q(X,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge$

$\neg \iota * q(Y,T_{vs},T_{ve},Current,Until_changed) \wedge$ *conditions.*

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if *p* is deleted with a transaction time start *Current* then the transaction time start of ¬*q* will be definitively different.

$ic_4' \leftarrow \delta * p(X,T_{vs},T_{ve},Current,Until_changed) \wedge \delta * q(Y,T_{vs},T_{ve},Current,Until_changed)$

$\wedge$ *conditions.*

This transition rule could violate the integrity constraint because if *p* and *q* are deleted with a transaction time start *Current* and a transaction time end *Until_changed* then both transaction times start and end for ¬*p* and ¬*q* will be the same. In this case, it should check the conditions of the integrity constraint.

4.2 Transaction Time Integrity Constraints

Ic2 is a transaction time integrity constraint. Transaction time *ic2* could be violated only if at a *Current* transaction time:

$ic2 \leftarrow student(S,T_{vs},T_{ve},T_{ts},T_{te}) \wedge registration(S,C,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge T_{ts} \leq T1_{ts} \wedge T_{te} < T1_{te}$

$ic2_2' \leftarrow student(S,T_{vs},T_{ve},T_{ts},T_{te})$

$\wedge \neg \delta * student(S,C,T_{vs},T_{ve},Current,Until_changed)$

$\wedge \iota *$ $registration(S,C,T_{vs},T_{ve},Current,Until_changed) \wedge T_{ts} \leq Current \wedge T_{te} < Until_changed$

Transaction time *ic2* use only its second transition rule, because third never can hold, no time is greater than *Until_changed*.

Case1. Let be a transaction time integrity constraint with all their literals positives:

$ic \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge conditions.$

Their transition rules are:

$ic_2' \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg\delta^{*}p(X,T_{vs},T_{ve},Current,Until_changed) \wedge$

$\iota^{*}q(Y,T_{vs},T_{ve},Current,Until_changed) \wedge conditions.$

This transition rule could violate the integrity constraint because if *q* is inserted with a transaction time start *Current* then the transaction time start of *p* will be definitively different because it is already in the database. In this case, it should check the conditions of the integrity constraint.

$ic_3' \leftarrow \iota^{*}p(X,T_{vs},T_{ve},Current,Until_changed) \wedge q(Y,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge$

$\neg\delta^{*}q(Y,T_{vs},T_{ve},Current,Until_changed) \wedge conditions.$

This transition rule could violate the integrity constraint because if *p* is inserted with a transaction time start *Current* then the transaction times start of *q* will be definitively different because it is already in the database. In this case, it should check the conditions of the integrity constraint.

$ic_4' \leftarrow \iota^{*}p(X,T_{vs},T_{ve},Current,Until_changed) \wedge \iota^{*}q(Y,T_{vs},T_{ve},Current,Until_changed)$

$\wedge conditions.$

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if *p* and *q* are inserted with a transaction time start *Current* and a transaction time end *Until_changed* then both transaction times start and end for *p* and *q* will be the same. This transition rule will be a current transition rule instead of a transaction time transition rule.

Case2. Let be a transaction time integrity constraint with their literals positives and negatives:

$ic \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg q(Y,T_{vs},T_{ve},T1_{ts},T1_{te}) \wedge conditions.$

Their transition rules are:

$ic_2' \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg\delta^{*}p(X,T_{vs},T_{ve},Current,Until_changed) \wedge$

$\delta^{*}q(Y,T_{vs},T_{ve},Current,Until_changed) \wedge conditions.$

This transition rule could violate the integrity constraint because if q is deleted with a transaction time start *Current* for $\neg q$ then the transaction time start of p will be definitively different. In this case, it should check the conditions of the integrity constraint

$ic_3' \leftarrow \iota^*p(X,T_{VS},T_{Ve},Current,Until_changed) \wedge \neg q(Y,T_{VS},T_{Ve},T_{ts},T_{te}) \wedge \neg\iota^*q(Y,T_{VS},T_{Ve},Current,Until_changed) \wedge conditions.$

This transition rule could violate the integrity constraint because if p is inserted with a transaction time start *Current* then the transaction time start of $\neg q$ will be definitively different. In this case, it should check the conditions of the integrity constraint.

$ic_4' \leftarrow \iota^*p(X,T_{VS},T_{Ve},Current,Until_changed) \wedge \delta^*q(Y,T_{VS},T_{Ve},Current,Until_changed) \wedge conditions.$

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if p is inserted and q is deleted with a transaction time start *Current* and a transaction time end *Until_changed* then both transaction times start and end for p and $\neg q$ will be the same. This transition rule will be a current transition rule instead of a transaction time transition rule.

Case3. Let be a transaction time integrity constraint with all their literals negatives:

$ic \leftarrow \neg p(X,T_{VS},T_{Ve},T_{ts},T_{te}) \wedge \neg q(Y,T_{VS},T_{Ve},T_{ts},T_{te}) \wedge conditions.$

Their transition rules are:

$ic_2' \leftarrow \neg p(X,T_{VS},T_{Ve},T_{ts},T_{te}) \wedge \neg\iota^*p(X,T_{VS},T_{Ve},Current,Until_changed) \wedge \delta^*q(Y,T_{VS},T_{Ve},Current,Until_changed) \wedge conditions.$

This transition rule could violate the integrity constraint because if q is deleted with a transaction time start *Current* then the transaction time start of $\neg p$ will be definitively different. In this case, it should check the conditions of the integrity constraint.

$ic_3' \leftarrow \delta^*p(X,T_{VS},T_{Ve},Current,Until_changed) \wedge \neg q(Y,T_{VS},T_{Ve},T_{ts},T_{te}) \wedge \neg\iota^*q(Y,T_{VS},T_{Ve},Current,Until_changed) \wedge conditions.$

This transition rule could violate the integrity constraint because if p is deleted with a transaction time start *Current* then the transaction time start of $\neg q$ will be definitively different. In this case, it should check the conditions of the integrity constraint.

$ic_4' \leftarrow \delta^*p(X,T_{VS},T_{Ve},Current,Until_changed) \wedge \delta^*q(Y,T_{VS},T_{Ve},Current,Until_changed) \wedge conditions.$

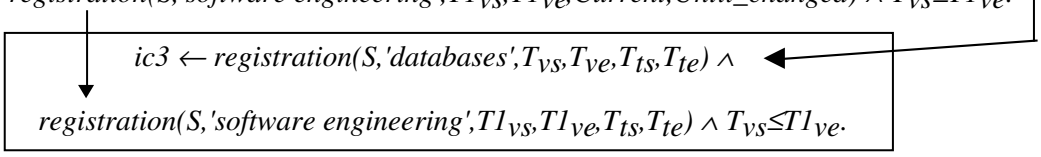
This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if *p* and *q* are deleted with a transaction time start *Current* and a transaction time end *Until_changed* then both transaction time start and end for ¬*p* and ¬*q* will be the same. This transition rule will be a current transition rule instead of a transaction time transition rule.

4.3 Valid Time Integrity Constraints

Ic3 is a valid time integrity constraint. *Ic3* could be violated if at *Current* transaction time:

$ic3_4' \leftarrow \iota^*registration(S,'databases',T_{vs},T_{ve},Current,Until_changed) \wedge$

$\iota^*registration(S,'software\ engineering',T1_{vs},T1_{ve},Current,Until_changed) \wedge T_{vs} \leq T1_{ve}.$



This group always has only one transition rule, the last one, to check the consistency of the database.

Case1. Let be a valid time integrity constraint with all their literals positives:

$ic \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T1_{vs},T1_{ve},T_{ts},T_{te}) \wedge conditions.$

Their transition rules are:

$ic_2' \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg\delta^*p(X,T_{vs},T_{ve},Current,Until_changed) \wedge$

$\iota^*q(Y,T1_{vs},T1_{ve},Current,Until_changed) \wedge conditions.$

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if *q* is inserted with a transaction time start *Current* then the transaction time start of *p* will be definitively different.

$ic_3' \leftarrow \iota^*p(X,T_{vs},T_{ve},Current,Until_changed) \wedge q(Y,T1_{vs},T1_{ve},T_{ts},T_{te}) \wedge$

$\neg\delta^*q(Y,T1_{vs},T1_{ve},Current,Until_changed) \wedge conditions.$

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if *p* is inserted with a transaction time start *Current* then the transaction time start of *q* will be definitively different.

$ic_4' \leftarrow \iota^*p(X,T_{vs},T_{ve},Current,Until_changed) \wedge$

$\iota^*q(Y,T1_{vs},T1_{ve},Current,Until_changed) \wedge conditions.$

This transition rule could violate the integrity constraint because if *p* and *q* are inserted

with a transaction time start *Current* and a transaction time end *Until_changed* then both transaction times start and end for p and q will be the same. In this case, it should check the conditions of the integrity constraint.

Case2. Let be a valid time integrity constraint with their literals positives and negatives:

$ic \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg q(Y,T1_{vs},T1_{ve},T_{ts},T_{te}) \wedge$ *conditions.*

Their transition rules are:

$ic_2' \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg \delta^{*}p(X,T_{vs},T_{ve},Current,Until_changed) \wedge$

$\delta^{*}q(Y,T1_{vs},T1_{ve},Current,Until_changed) \wedge$ *conditions.*

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if q is deleted with a transaction time start *Current* for $\neg q$ then the transaction time start of p will be definitively different.

$ic_3' \leftarrow \iota^{*}p(X,T_{vs},T_{ve},Current,Until_changed) \wedge \neg q(Y,T_{vs},T_{ve},T_{ts},T_{te}) \wedge$

$\neg \iota^{*}q(Y,T_{vs},T_{ve},Current,Until_changed) \wedge$ *conditions.*

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if p is inserted with a transaction time start *Current* then the transaction time start of $\neg q$ will be definitively different.

$ic_4' \leftarrow \iota^{*}p(X,T_{vs},T_{ve},Current,Until_changed) \wedge \delta^{*}q(Y,T_{vs},T_{ve},Current,Until_changed)$

$\wedge$ *conditions.*

This transition rule could violate the integrity constraint because if p is inserted with a transaction time start *Current* and a transaction time end *Until_changed* then both transaction times start and end for p and $\neg q$ will be the same. In this case, it should check the conditions of the integrity constraint.

Case3. Let be a valid time integrity constraint with all their literals negatives:

$ic \leftarrow \neg p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg q(Y,T_{vs},T_{ve},T_{ts},T_{te}) \wedge$ *conditions.*

Their transition rules are:

$ic_2' \leftarrow \neg p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg \iota^{*}p(X,T_{vs},T_{ve},Current,Until_changed) \wedge$

$\delta^{*}q(Y,T1_{vs},T1_{ve},Current,Until_changed) \wedge$ *conditions.*

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if q is deleted with a transaction time start *Current* then the transaction time start of $\neg p$ will be definitively different.

$ic_3 \leftarrow \delta^*p(X,T_{vs},T_{ve},Current,Until_changed) \wedge \neg q(Y,T1_{vs},T1_{ve},T_{ts},T_{te}) \wedge$

$\neg \iota^* q(Y,T1_{vs},T1_{ve},Current,Until_changed) \wedge$ *conditions.*

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if *p* is deleted with a transaction time start *Current* then the transaction time start of ¬*q* will be definitively different.

$ic_4 \leftarrow \delta^*p(X,T_{vs},T_{ve},Current,Until_changed) \wedge$

$\delta^* q(Y,T1_{vs},T1_{ve},Current,Until_changed) \wedge$ *conditions.*

This transition rule could violate the integrity constraint because if *p* and *q* are deleted with a transaction time start *Current* and a transaction time end *Until_changed* then both transaction times start and end for ¬*p* and ¬*q* will be the same. In this case, it should check the conditions of the integrity constraint.

4.4 Bitemporal Integrity Constraints

Ic4 is a bitemporal integrity constraint. *Ic4* could be violated if at *Current* transaction time:

$ic42' \leftarrow \iota^* course(C,P,T_{vs},T_{ve},Current,Until_changed)$

$\wedge\ course(C,P1,T1_{vs},T1_{ve},T1_{ts},T1_{te}) \wedge \neg \delta^* course(C,P1,T1_{vs},T1_{ve},Current,Until_changed)$

$\wedge\ P1 < P \wedge T1_{vs} < T_{ve} \wedge T_{ts} < Current\text{-}10.$

$ic4 \leftarrow course(C,P,T_{vs},T_{ve},T_{ts},T_{te}) \wedge course(C,P1,T1_{vs},T1_{ve},T1_{ts},T1_{te})$

$\wedge\ P1 < P \wedge T1_{vs} < T_{ve} \wedge T_{ts} < T1_{ts}\text{-}10.$

Bitemporal time *ic4* use only his second transition rule, because third never can hold, no time is greater than Current.

Case1. Let be a bitemporal integrity constraint with all their literals positives:

$ic \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge q(Y,T1_{vs},T1_{ve},T1_{ts},T1_{te}) \wedge$ *conditions.*

Their transition rules are:

$ic_2' \leftarrow p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg \delta^* p(X,T_{vs},T_{ve},Current,Until_changed) \wedge$

$\iota^* q(Y,T1_{vs},T1_{ve},Current,Until_changed) \wedge$ *conditions.*

This transition rule could violate the integrity constraint because if *q* is inserted with a

transaction time start *Current* then the transaction time start of p will be definitively different because it is already in the database. In this case, it should check the conditions of the integrity constraint.

$ic_3' \leftarrow \iota^*p(X,T_{VS},T_{Ve},Current,Until_changed) \wedge q(Y,T1_{VS},T1_{Ve},T1_{TS},T1_{Te}) \wedge$

$\neg\delta^*q(Y,T1_{VS},T1_{Ve},Current,Until_changed) \wedge conditions.$

This transition rule could violate the integrity constraint because if p is inserted with a transaction time start *Current* then the transaction time start of q will be definitively different because it is already in the database. In this case, it should check the conditions of the integrity constraint.

$ic_4' \leftarrow \iota^*p(X,T_{VS},T_{Ve},Current,Until_changed) \wedge$

$\iota^*q(Y,T1_{VS},T1_{Ve},Current,Until_changed) \wedge conditions.$

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if p and q are inserted with a transaction time start *Current* and a transaction time end *Until_changed* then both transaction times start and end for p and q will be the same. This transition rule will be a valid time transition rule instead of a bitemporal transition rule.

Case2. Let be a bitemporal integrity constraint with their literals positives and negatives:

$ic \leftarrow p(X,T_{VS},T_{Ve},T_{TS},T_{Te}) \wedge \neg q(Y,T1_{VS},T1_{Ve},T1_{TS},T1_{Te}) \wedge conditions.$

Their transition rules are:

$ic_2' \leftarrow p(X,T_{VS},T_{Ve},T_{TS},T_{Te}) \vee \neg\delta^*p(X,T_{VS},T_{Ve},Current,Until_changed) \wedge$

$\delta^*q(Y,T1_{VS},T1_{Ve},Current,Until_changed) \wedge conditions.$

This transition rule could violate the integrity constraint because if q is deleted with a transaction time start *Current* for $\neg q$ then the transaction time start of p will be definitively different. In this case, it should check the conditions of the integrity constraint.

$ic_3' \leftarrow \iota^*p(X,T_{VS},T_{Ve},Current,Until_changed) \wedge \neg q(Y,T1_{VS},T1_{Ve},T1_{TS},T1_{Te}) \wedge$

$\neg\iota^*q(Y,T1_{VS},T1_{Ve},Current,Until_changed) \wedge conditions.$

This transition rule could violate the integrity constraint because if p is inserted with a transaction time start *Current* then the transaction time start of $\neg q$ will be definitively different. In this case, it should check the conditions of the integrity constraint.

$ic_4' \leftarrow \iota^*p(X,T_{VS},T_{Ve},Current,Until_changed) \wedge$

$\delta^*q(Y,T1_{VS},T1_{VE},Current,Until_changed) \wedge$ *conditions.*

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if *p* is inserted and *q* is deleted with a transaction time start *Current* and a transaction time end *Until_changed* then both transaction times start and end for *p* and ¬*q* will be the same. This transition rule will be a valid time transition rule instead of a bitemporal transition rule.

Case3. Let be a bitemporal integrity constraint with all their literals negatives:

$ic \leftarrow \neg p(X,T_{VS},T_{VE},T_{TS},T_{TE}) \wedge \neg q(Y,T1_{VS},T1_{VE},T1_{TS},T1_{TE}) \wedge$ *conditions.*

Their transition rules are:

$ic_2' \leftarrow \neg p(X,T_{VS},T_{VE},T_{TS},T_{TE}) \wedge \neg \iota^* p(X,T_{VS},T_{VE},Current,Until_changed) \wedge$

$\delta^*q(Y,T1_{VS},T1_{VE},Current,Until_changed) \wedge$ *conditions.*

This transition rule could violate the integrity constraint because if *q* is deleted with a transaction time start *Current* then the transaction time start of ¬*p* will be definitively different. In this case, it should check the conditions of the integrity constraint.

$ic_3' \leftarrow \delta^* p(X,T_{VS},T_{VE},Current,Until_changed) \wedge \neg q(Y,T1_{VS},T1_{VE},T1_{TS},T1_{TE}) \wedge$

$\neg \iota^* q(Y,T1_{VS},T1_{VE},Current,Until_changed) \wedge$ *conditions.*

This transition rule could violate the integrity constraint because if *p* is deleted with a transaction time start *Current* then the transaction time start of ¬*q* will be definitively different. In this case, it should check the conditions of the integrity constraint.

$ic_4' \leftarrow \delta^* p(X,T_{VS},T_{VE},Current,Until_changed) \wedge$

$\delta^*q(Y,T1_{VS},T1_{VE},Current,Until_changed) \wedge$ *conditions.*

This transition rule will be eliminated because it never will violate the integrity constraint. The reason will be that if *p* and *q* are deleted with a transaction time start *Current* and a transaction time end *Until_changed* then both transaction times start and end for ¬*p* and ¬*q* will be the same. This transition rule will be a valid time transition rule instead of a bitemporal transition rule.

4.5 Integrity Constraints with Derived Predicates

In an integrity constraint with derived predicates, we need to introduce the concept of **insertion and deletion event rules**.

Let p be a derived predicate. Insertion events of p were defined as:

$\forall X,T_{vs},T_{ve},T_{ts},T_{te}\ (\iota^{*}p(X,T_{vs},T_{ve},Current,Until_changed) \leftrightarrow$

$p'(X,T_{vs},T_{ve},Current,Until_changed) \wedge \neg p(X,T_{vs},T_{ve},T_{ts},T_{te}))$.

And replacing $p'(X,T_{vs},T_{ve},Current,Until_changed)$ we get:

$\iota^{*}p(X,T_{vs},T_{ve},Current,Until_changed) \leftarrow$

$p_i'(X,T_{vs},T_{ve},Current,Until_changed) \wedge \neg p(X,T_{vs},T_{ve},T_{ts},T_{te})$ with $i = 2, ..., 2^k$.

By replacing $p_i'(X,T_{vs},T_{ve},Current,Until_changed)$ with its equivalent definition, we get a set of **insertion events rules**. They allow us to deduce which ιp (induced insertions) happen in a transition.

Let p be a derived predicate. Deletion events of p were defined as:

$\forall X,T_{vs},T_{ve},T_{ts},T_{te}\ (\delta^{*}p(X,T_{vs},T_{ve},Current,Until_changed) \leftrightarrow$

$p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg p'(X,T_{vs},T_{ve},Current,Until_changed))$.

And replacing $p'(X,T_{vs},T_{ve},Current,Until_changed)$ we get:

$\delta^{*}p(X,T_{vs},T_{ve},Current,Until_changed) \leftarrow$

$p(X,T_{vs},T_{ve},T_{ts},T_{te}) \wedge \neg p_i'(X,T_{vs},T_{ve},Current,Until_changed))$ with $i = 2, ..., 2^k$.

By replacing $p_i'(X,T_{vs},T_{ve},Current,Until_changed)$ with its equivalent definition, we get a set of **deletion events rules**. They allow us to deduce which δp (induced deletions) happen in a transition.

Example.

Let *ic5* be a current integrity constraint with a derived predicate:

$ic5 \leftarrow registration(S,C,T_{vs},T_{ve},T_{ts},T_{ve}) \wedge \neg classes(C,T_{vs},T_{ve},T_{ts},T_{te})$.

Ic5 enforces the property that "a student *S* can be enrolled in a course *C* if this course *C* is offered" with the simple deductive rule, for the sake of simplicity:

$classes(C,T_{vs},T_{ve},T_{ts},T_{te}) \leftarrow course(C,P,T_{vs},T_{ve},T_{ts},T_{te})$.

Ic5 is a current integrity constraint and we obtain the following transition rules, after applying current integrity constraint simplifications.

$ic5' \leftarrow ic54'$

$ic54' \leftarrow \iota^{*}registration(S,C,T_{vs},T_{ve},Current,Until_changed)$

$\wedge \delta^{*}classes(C,T_{vs},T_{ve},Current,Until_changed)$.

In this transition rule, we found a deletion event rule:

$\delta^{*}classes(C,T_{vs},T_{ve},Current,Until_changed) \leftarrow classes(C,T_{vs},T_{ve},T_{ts},T_{te})$

$\wedge \neg classes2'(C,T_{vs},T_{ve},Current,Until_changed)$

where, substituting we obtain:

$\delta^{*}classes(C,T_{vs},T_{ve},Current,Until_changed) \leftarrow course(C,P,T_{vs},T_{ve},T_{ts},T_{te})$

$\wedge \delta^{*}course(C,T_{vs},T_{ve},Current,Until_changed)$

4.6 Application Example of Our Method.

Let *D* be a bt-ddb, *A(D)* the augmented bt-ddb, and *TR* a transaction consisting of a set of events. If *TR* leads to an inconsistency then some of the *ιcn* 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 4.2.

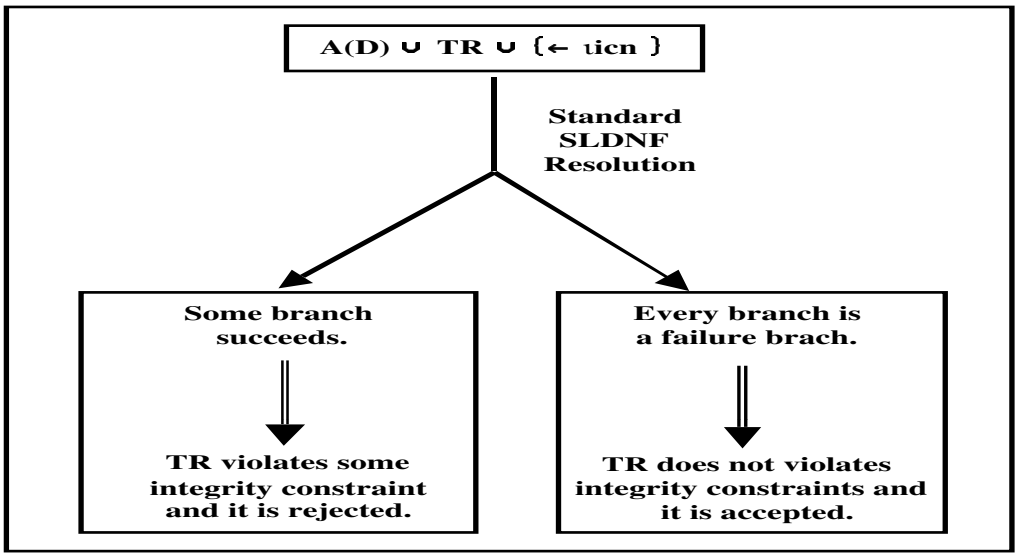


Figure 4.2. *Application of our method.*

Following, we illustrate the utility of the transition rules for integrity constraint checking, with a transaction *TR1* that violates *ic5* in a bt-ddb with the following facts:

course(logic,14000,1,3,2,Until_changed)

student(jordi,1,3,2,Until_changed)

In figure 4.3 we show the general table that summarize the final transition rules obtained after classifying every integrity constraint in its group.

Integrity Constraints	First Group Current	Second Group Transaction Time	Third Group Valid Time	Fourth Group Bitemporal
Case1: Only positive literals	Only last rule	All the rules except the last one	Only last rule	All the rules except the last one
Case2: Positive and negative literals	Only last rule	All the rules except the last one	Only last rule	All the rules except the last one
Case3: Only negative literals	Only last rule	All the rules except the last one	Only last rule	All the rules except the last one

Figure 4.3. *Transition rules used in every group of our taxonomy.*

In figure 4.4 we show the SLDNF-proof procedure of this violation.

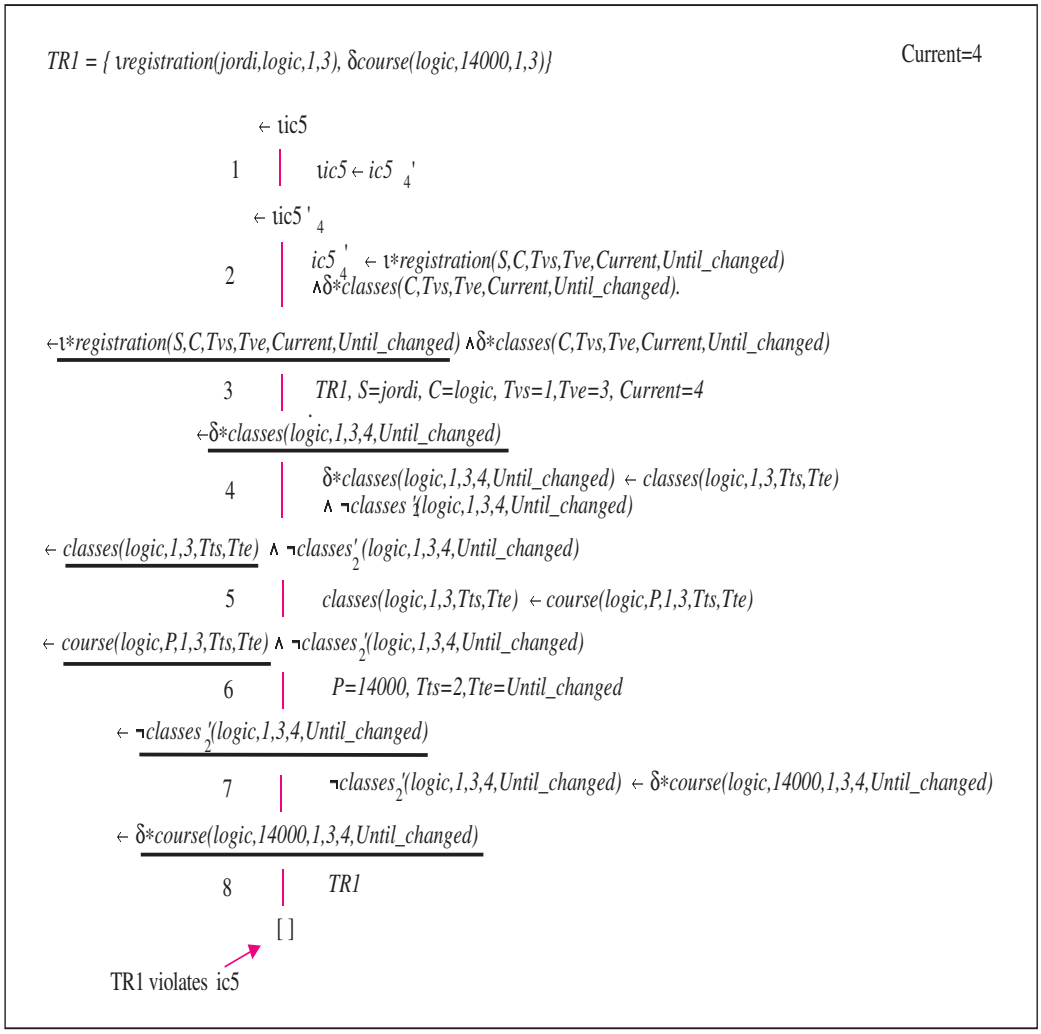


Figure 4.4. *Violation of ic5 for ic54'.*

Steps 1, 2, 4, 5 and 7 are SLDNF-resolution steps where rules of A(D) act as input clauses.

At steps 3 and 8 the predicates refer to the transaction and we go to the transaction to find them. At step 6 the predicate refers to the database and we go to the database to find it.

Note 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).

4.7 Solving Indefinitions for Transaction Time Variables of Predicates.

In a transaction *TR*, the insertion and deletion events could produce or induce predicates, which always have specified a valid time start and a valid time end but the transaction time start and the transaction time end could be undefined and must be deduced with the information obtained from the transaction, the deductive rules and the facts of the bt-ddb.

Let $p(X,T_{VS},T_{Ve},T_{ts},T_{te})$ be a predicate, all the different cases for undefined transaction time will be:

Case1. The predicate $p(X,T_{VS},T_{Ve},T_{ts},T_{te})$ exists in the database:

Example:

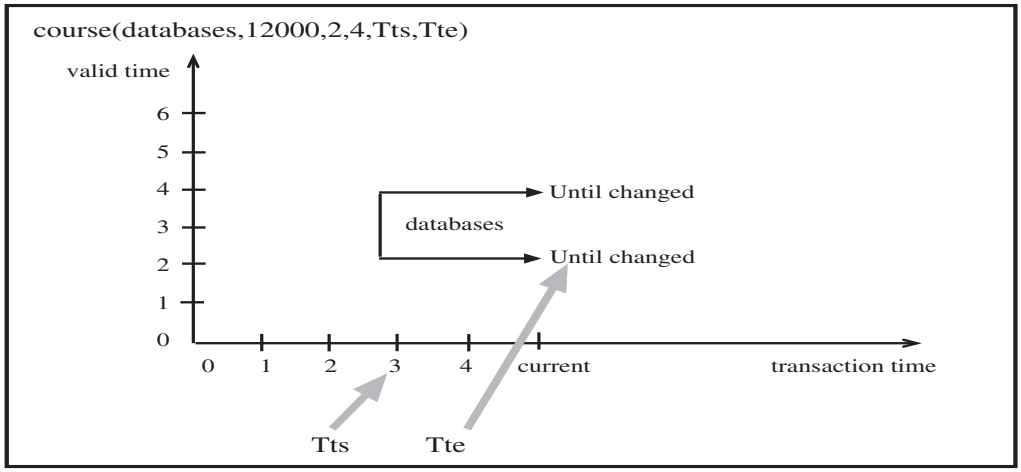


Figure 4.5. *Graphically representation of case 1.*

Case2. The predicate $p(X,T_{VS},T_{Ve},T_{ts},T_{te})$ not exists in the database:

$$T_{ts} = 0 \quad T_{te} = 0$$

Case3. The predicate $p(X,T_{VS},T_{Ve},T_{ts},T_{te})$ exists in the database with different valid times.

Example:

If the valid time intersection is empty:

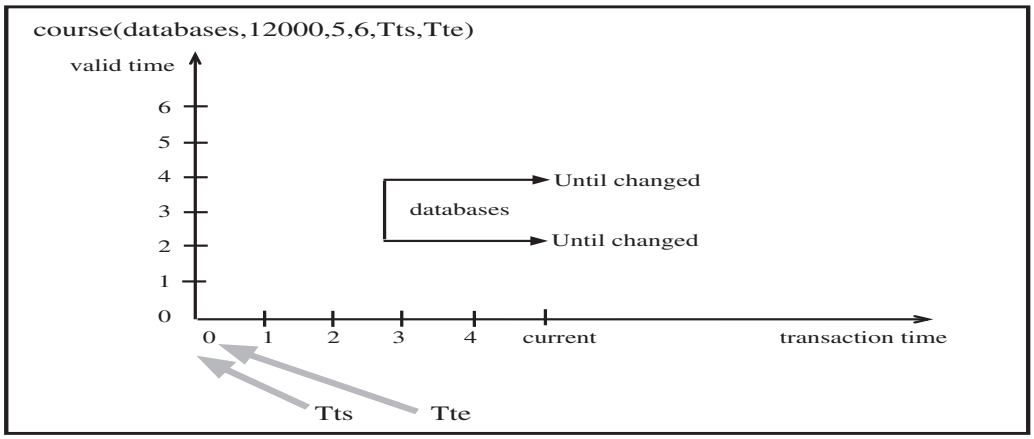


Figure 4.6. *Graphically representation of case 3 for empty intersection.*

If the valid time intersection is not empty, but the valid time of *p* is included in the valid time of the database information, then *p* exists during the transaction time where the intersection is not empty. Otherwise the predicate fails.

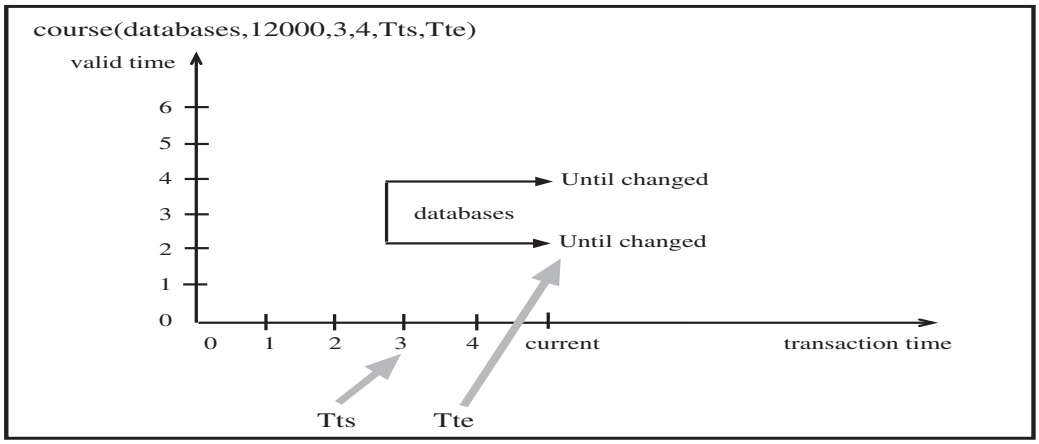


Figure 4.7. *Graphically representation of case 3 for non empty intersection.*

Case4. The predicate $p(X,T_{VS},T_{Ve},T_{ts},T_{te})$ is inserted with a transaction time start *Current* in the database:

Example: ι∗course(databases, 12000, 2, 4, T_{ts}, T_{te})

Current Until_changed

Case5. The predicate $p(X,T_{VS},T_{Ve},T_{ts},T_{te})$ is deleted with a transaction time start *Current* of the database:

Example: course(databases, 12000, 2, 4, 3, Until_changed)

δ∗course(databases, 12000, 2, 4, T_{ts}, T_{te})

Current-1

Let $\neg p(X,T_{vs},T_{ve},T_{ts},T_{te})$ be a predicate, all the different cases for undefined transaction time will be:

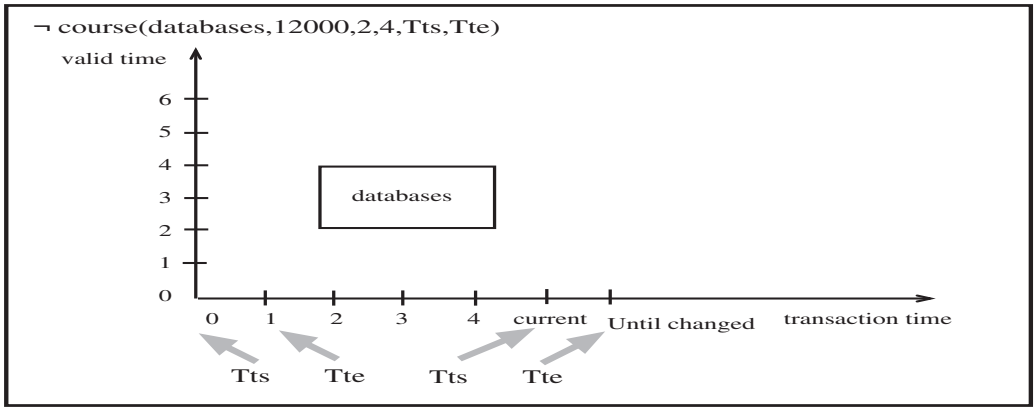


Figure 4.8. *Graphically representation of case 6.*

Case6. The predicate $p(X,T_{vs},T_{ve},T1_{ts},T1_{te})$ exists in the database, so for $\neg p(X,T_{vs},T_{ve},T_{ts},T_{te})$ we obtain:

Example in figure 4.8.

Case7. The predicate $p(X,T_{vs},T_{ve},T1_{ts},T1_{te})$ not exists in the database, so for $\neg p(X,T_{vs},T_{ve},T_{ts},T_{te})$ we obtain: $T_{ts} = 0$ T_{te} = Until_changed

Case8. The predicate $p(X,T1_{vs},T1_{ve},T1_{ts},T1_{te})$ exists in the database with different valid times, so for $\neg p(X,T_{vs},T_{ve},T_{ts},T_{te})$ we obtain:

Example:

If the valid time intersection is empty:

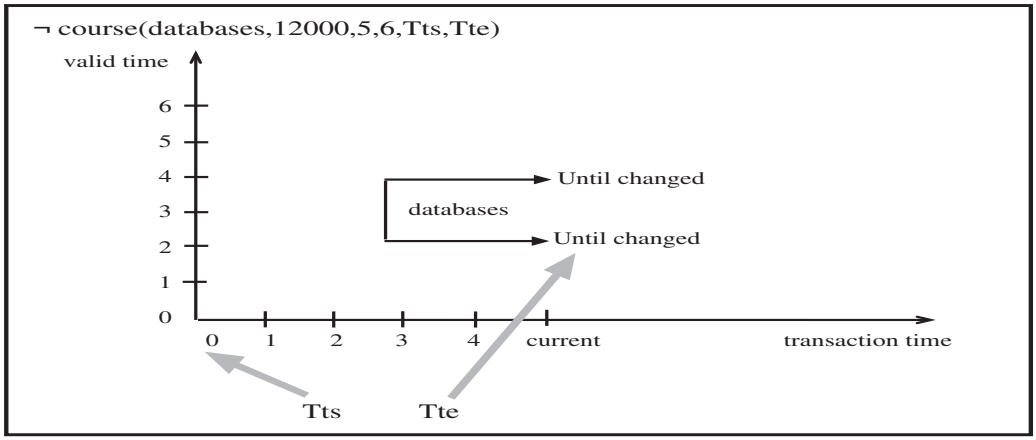


Figure 4.9. *Graphically representation of case 8 for empty intersection*

If the valid time intersection is not empty, but the valid time of p is included in the valid time of the database information, then $\neg p$ exists during the transaction time where the

intersection is empty. Otherwise the predicate fails.

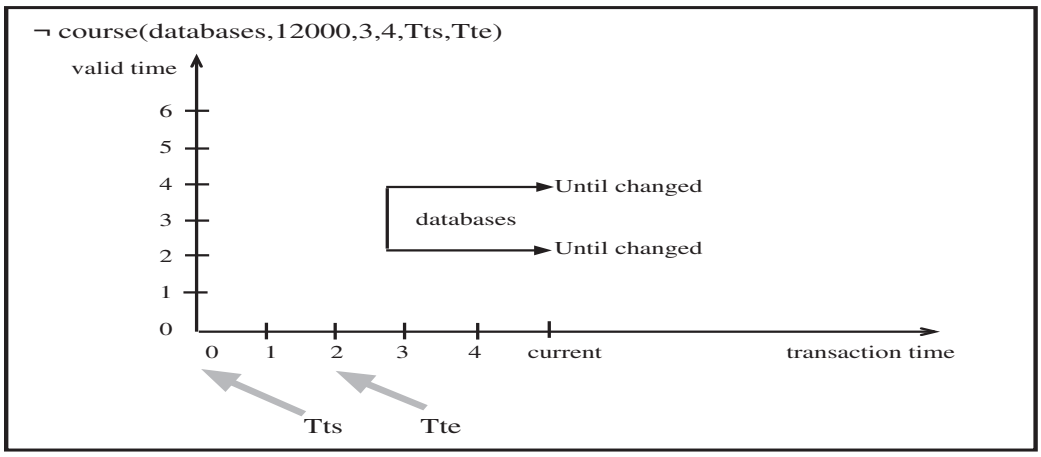


Figure 4.10. *Graphically representation of case 8 for non empty intersection*

Case9. The predicate $p(X,T_{vs},T_{ve},T1_{ts},T1_{te})$ is inserted with a transaction time start *Current* in the database, so for $\neg p(X,T_{vs},T_{ve},T_{ts},T_{te})$ we obtain:

Example:

ι∗course(databases, 12000, 2, 4, T_{ts}, T_{te})

If the predicate not exits in the database: $T_{ts} = 0$ T_{te} = Current-1

If the predicate exits in the database, it will be case 6 (or case 8 for different valid times):

0 2 course(databases, 12000, 2, 4, 3, 7) 8 Current - 1

T_{ts} T_{te} T_{ts} T_{te}

Case10. The predicate $p(X,T_{vs},T_{ve},T1_{ts},T1_{te})$ is deleted with a transaction time start *Current* of the database, so for $\neg p(X,T_{vs},T_{ve},T_{ts},T_{te})$ we obtain:

Example:

δ∗course(databases, 12000, 2, 4, T_{ts}, T_{te})

Current Until_changed

4.8 Application Example with Transaction Time Indefinition.

We finish this section with an example of solving indefinitions for transaction time variables of predicates during the SLDNF-resolution procedure.

We illustrate the utility of the deduction of transaction time, explained in the previous section, for integrity constraint checking, with a transaction *TR2* that violates *ic2* in a bt-ddb with the fact:

course(logic,14000,1,3,2,Until_changed).

In figure 4.11 we show the SLDNF-proof procedure of this violation.

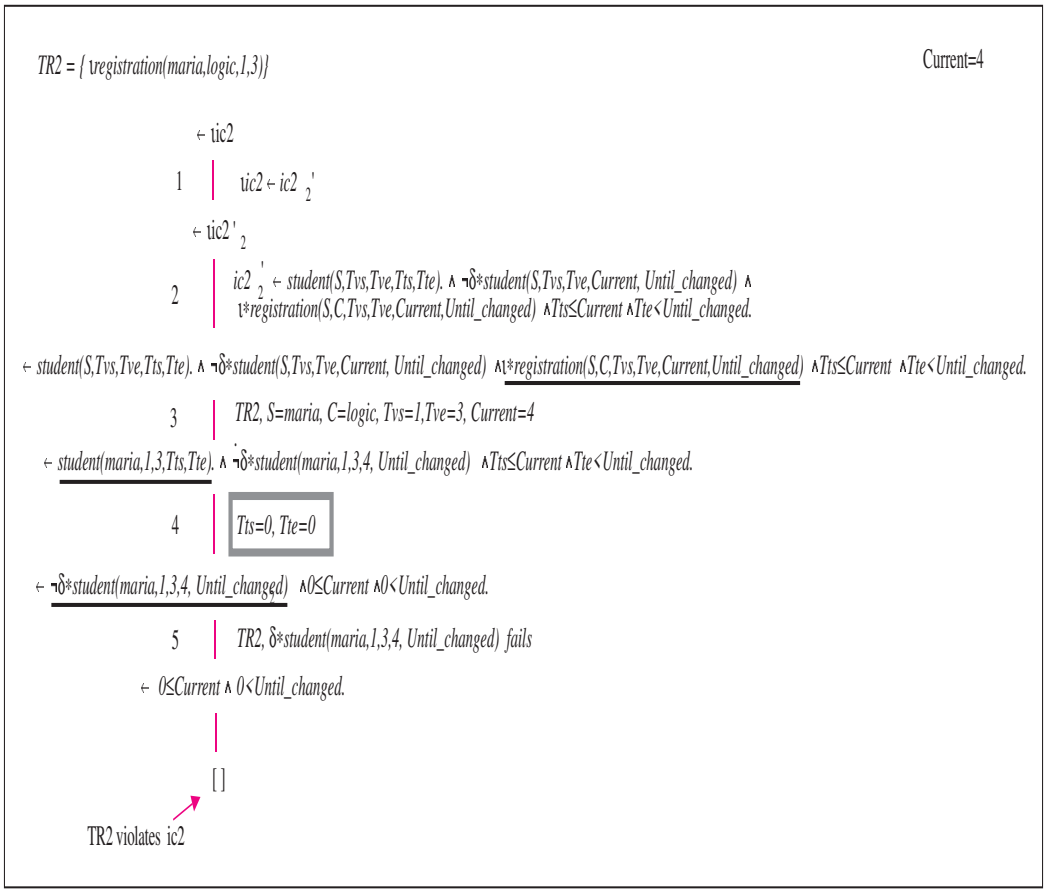


***Figure 4.11**. Violation of ic2 for ic2$_2$'.*

Steps 1 and 2 are SLDNF-resolution steps where rules of A(D) act as input clauses. Steps 3 and 5 refer to the transaction and we go to the transaction to find them. It is in step 4 when we use the deduction of the transaction time for a predicate that not exists in the database.

5 Conclusions

In this paper, a new taxonomy of temporal integrity constraints for bt-ddbs is introduced. Our taxonomy combines the four temporal criteria of Anne Doucet *et al's* with the Michael Böhlen's taxonomy to get a better understanding of temporal integrity constraints in bt-ddbs. The Michael Böhlen's taxonomy is interesting when only one transaction database or valid time database is used. However, we consider that temporal integrity constraints in bt-ddbs can be any of the Michael Böhlen's taxonomy integrity constraints, depending on one of the

following: whether they have no time reference, or they only have transaction time references, or they only have valid time references, or they have both time references. Taking into account this situation, our integrity constraint taxonomy comprises only four groups, where we apply seven criteria (adapted from the four of Anne Doucet *et al's*) to know properties that help us to improve our integrity constraint checking method [19]. Table 2.1 summarizes the relationship between our four groups and the six basic categories of Michael Böhlen combined with our seven criteria.

Regarding transactions, the important concepts of insertion and deletion event and coherent transaction have been explained. The coherence transaction definition allows us to reject transactions before checking.

Finally, the presented taxonomy has already been applied to an extension of our integrity constraint checking method for bt-ddbs ([18] and [19]). It has been mainly due to this application that spare transition rules have been successfully eliminated (as we show in figure 4.3). Some examples have been presented in order to show the usefulness of our method with this new set of transition rules.

Acknowledgments

The author would like to thank Núria Castell for the support he has given to this work and Toni Urpí, Enric Mayol, Joan Antoni Pastor, Àngels Hernández and specially Victoria Arranz for many useful comments and discussions.

References

1. Allen, J.F. "Maintaining Knowledge about Temporal Intervals". In Communications of the ACM. Vol.26. No.11. pp 832-843. 1983.
2. Böhlen, M. "Managing Temporal Knowledge in Deductive Databases". PhD thesis, Swiss Federal Institute of Technology. Zürich, 1994.
3. Böhlen, M. "Valid Time Integrity Constraints". Technical Report 94-30. Computer Science. University of Arizona, 1994.
4. Bancilhom, F.; Ramakrishan, R. "An Amateurs Introduction to Recursive Query Processing Strategies". Proc. of the ACM SIGMOD Int. Conf. on Management of Data. Washington D.C., pp 16-52. 1986
5. Clifford, J.; Dyreson, C.; Isakowitz, T.; Jensen, C.S., Snodgrass, R.T. "On the Semantics of "Now" in Databases". In ACM Transactions on Database Systems. Vol.22. No.2. pp 171-214. June, 1997.

6. Cowley, W.; Plexousakis, D. "Temporal Integrity Constraints with Indeterminacy". Proc. of the 26th. Int. Conf. on VLDB'00. pp 441-450. Cairo, 2000.
7. Chomicki, J.; Toman, D. "Implementing Temporal Integrity Constraints Using an Active DBMS". In the 11th Int. Conf. on Data Engineering, IEEE Computer Society Press. pp 566-582. Taipei, March, 1995.
8. Doucet, A.; Fauvet, M.C.; Garçarski, S.; Jomier, G. and Monties, S. "Using Database Versions to Implement Temporal Integrity Constraints". Proc. of the Int. Workshop on Constraint Databases. pp 219-233. 1997.
9. Das, S.; Williams, H. "A Path Finding Method for Constraint Checking in Deductive Databases". Data & Knownedge Engineering. No. 4. pp 223-224. 1989.
10. Etzion, O.; Jajodia, S.; Sripada, S. "Temporal Databases: Research and Practice". Springer-Verlag, 1998.
11. 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, pp 77-92. September, 1995.
12. Gallaire, H.; Minker, J. “Logic in Data Bases”. Plenum Press. New York, 1978.
13. Jensen, C.S.; Clifford, J.; Gadia, S.K.; Segev, A.; Snodgrass, R.T. "A Glossary of Temporal Database Concepts". In ACM SIGMOD Record. Vol. 21. No. 3. September, 1992. pp QQ-1-QQ-28
14. Jensen, C.S.; Snodgrass, R.T. "Temporal Specialization and Generalization". In TKDE Vol.6. No.6. pp 954-974. 1994.
15. Jensen, C.S.; Soo, M.D.; Snodgrass R.T. “Extending Normal Forms to Temporal Relations". Technical Report 92-17. Computer Science. University of Arizona, 1992.
16. Kowalski, R. “Logic for Data Description”. In [12], pp 77-103.
17. Martín, C.; Sistac, J. "A Method for Integrity Constraint Checking in Temporal Deductive Databases". In Proc. of the 3th. Int. Workshop on Temporal Representation and Reasoning. pp 136-141. Key West, 1996.
18. Martín, C.; Sistac, J. "Applying Transition Rules to Bitemporal Deductive Databases for Integrity Constraint Checking". In Proc. of the Int. Workshop on Logic in Databases. pp 117-134. San Miniato, 1996.
19. Martín, C. "The Events Method for Temporal Integrity Constraint Handling in Bitemporal Deductive Databases”. Report de recerca LSI-98-68-R. Departament de Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya. 1998.
20. Martín, C. "Restriccions d’Integritat Temporals en Bases de Dades Deductives Bitemporals”. Report de recerca LSI-00-80-R (in Catalan). Departament de Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya. 2000.
21. Olivé, A. "Integrity Constraints Checking in Deductive Databases". Proc. of the 17th. Int. Conf on VLDB'91. Barcelona, 1991. pp 513-523.
22. Snodgrass, R.T.; Ahn, I. "Temporal Databases". IEEE Computer. Vol 19. No. 9. 1986. pp 35-42.
23. Snodgrass, R.T. "Developing Time-Oriented Database Applications in SQL". Morgan Kaufmann Publishers. 2000.

24. Tansel, A.U.; Clifford, J.; Gadia, S.; Jajodia, S.; Segev, A.; Snodgrass, R. "Temporal Databases: Theory, Design and Implementation". Benjamin/Cummings. 1993.
25. Wijsen, J. "Temporal FDs on Complex Objects"". ACM Transactions on Database Systems. Vol. 24. No 1. 1999. pp. 127-176.
26. Wu, Y.; Jajodia, S.; Wang, X.S. "Temporal Database Bibliography Update". In [10], pp 338-366.