An Integrity Constraint Checking Method for Temporal Deductive Databases.

Carme Martín, Jaume Sistac.
Departament de Llenguatges i Sistemes Informàtics.
Universitat Politècnica de Catalunya.
Barcelona - Catalonia.

Abstract

We propose a method for integrity checking in the context of temporal deductive databases. A temporal deductive database is a deductive database that supports some aspect of time, in our case valid time, in which valid time is the time when the fact is true in the modelled reality. Our method augments a database with a set of transition and event rules, which explicitly define the insertions and deletions by database update. Standard SLDNF resolution can then be used to check satisfaction of integrity constraints.

A temporal deductive database has to be consistent, that is, after performing an update the database must satisfy its set of integrity constraints. Using our method the temporal deductive database always is consistent because if a retroactive or proactive update violated some integrity constraint the update would be rejected.

1 Introduction

In the last meeting of temporal database community [5] it became clear that actual temporal database system implementations need an integrity constraint checking method to assure the consistency of the database. We contribute to a solution of this problem with an integrity checking method for temporal deductive databases, since we think that using deductive databases one will obtain all the reasoning advantages of these databases. Our method augments a temporal deductive database with a set of transition and event rules, which explicitly define the insertions and deletions by database update. Standard SLDNF resolution can then be used to check satisfaction of integrity constraints.

[15] distinguished between two measures of time, 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 was stored in the database. In a glossary of temporal databases [4], the concept of temporal database has been kept for a database that supports some aspect of time, in our case this time is valid time, we call them valid time deductive databases and we denote them vt-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 vt-ddb must be consistent, that is, when performing a past, present or futur 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.

Integrity constraint checking is an essential issue, which has been widely studied in relational and deductive databases, but not in the field of temporal deductive databases. In this paper, we present a method for solving the problem of integrity constraint checking in vt-ddbs.

The paper is organised as follows. The next Section defines basic concepts of vt-ddbs and introduces a simple example that will be used throughout the paper. Section 3 presents the key concepts of our method. Section 4 describes our method for integrity constraint checking in vt-ddbs. Section 5 compares our method with previous related work. Finally, Section 6 gives the conclusions and points out future work.

2 Valid time deductive databases

A vt-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 vt-ddb can be defined in this form.

Facts, rules and integrity constraints are formulated in a first order language.

Let $T=\{..., t_0, t_1, ...\}$ be a set of times, at most countably infinite, over which is defined the linear (total) order $<_T$, where $t_i <_T t_j$ means t_i occurs before t_j. The set T is used as the basis for incorporating the temporal dimension into the database. We assume that T is isomorphic to the natural numbers.

We assume that database predicates are either base or derived.

Base predicates and derived predicates contain a last term which is a valid time point ranging over the temporal domain *T*. Integrity constraints use the usual operators =, >, <, ≥, ≤ and ≠ to compare a valid time points and to express static and dynamic constraints.

Our method adopts a closed time interval model based in the time representation of valid time information presented by [14].

A fact is a ground atom (as usual we call an atom ground if it is variable free). The last term of any fact is a time interval ranging over the temporal domain *T*, represented by two time points: valid time-start (t_s) and valid time-end (t_e). These time points correspond to the lower and upper bounds of a time interval.

Each fact has a precise t_s value; however, t_e may not be known. In this case, t_e 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 when the user knows it.

2.1 Deductive Rules

A deductive rule is a formula of the form: $p \leftarrow L_1 \wedge \ldots \wedge L_n$ *with* $n \geq 1$, where *p* is an atom, denoting the conclusion, $L_1 \wedge \ldots \wedge L_n$ are literals representing conditions. Any variables in $p, L_1 \wedge \ldots \wedge L_n$ 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 our method for 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 [1] and, as usual with other deductive databases, we require the vt-ddb before and after any updates to be allowed [7].

2.2 Integrity Constraints

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

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

where each L_i is a literal. Any variables in $L_1 \wedge \ldots \wedge L_n$ 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*, with or without terms, and thus it has the same form as the deductive rules. We call them integrity rules. Then, we rewrite the former denial as:

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

The evolution through valid time of a deductive database can be described by a state for each predicate. According to this evolution scheme, static and dynamic constraints can be distinguished: the former restrict the validity to the current state of the deductive database, while the latter relate the validity to past and future states in addition to the current state.

2.3 Example

Base Predicates. We present two base predicates: *offered(C,T)* expresses that "course *C* has been offered at valid time *T*, and *takes(S,C,T)* expresses that "the student S has been enrolled in course *C* at valid time *T*.

Deductive Rules. We present two deductive rules that defines two derived predicates: *some_enrol(C,T)* expresses that "the course *C* has one or more students at valid time *T*, and *many_students(C,T)* expresses that "the course *C* has two students at valid time *T*, at least".

R.1 *some_enrol(C,T) ← takes(S,C,T).*

R.2 *many_students(C,T) ← takes(S1,C,T) ∧ takes(S2,C,T) ∧ S1 ≠ S2.*

Integrity Constraints. We present two static integrity constraints and two dynamic integrity constraints:

Static constraints: *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".

IC.1 *ic1 ← takes(S,C,T) ∧ ¬offered(C,T)*

IC.2 *ic2 ← offered(C,T) ∧ ¬some_enrol(C,T).*

Dynamic constraints: *ic3* and *ic4*, respectively, enforce the properties that "if a student *S* is enrolled in the course *software engineering*, this student cannot be 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".

IC.3 *ic3 ← takes(S,software engineering,T) ∧ takes(S,information systems,T1) ∧ T1<T.*

IC.4 *ic4 ← offered(C,T) ∧ ¬many_students(C,T) ∧ many_students(C,T1) ∧ T1<T.*

3 Key concepts of our method

In this section, we begin by defining the key concepts of our method for integrity checking in vt-ddb. The concept of event, transition rule and event rule was introduced by [11] in the event model as an approach for the design of information systems from deductive conceptual models, and was improved in [16] to include database and transaction design decisions.

3.1 Events

Let *D* be a deductive database at valid time point t_v, *U* an update and D' the updated deductive database at valid time point t_V. We assume for the moment that U consists of an unspecified set of facts to be inserted and/or deleted.

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:

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

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

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

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

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

In case (2) we say that a **deletion event** occurs in the transition, and we denote it by $\delta p(x,t_V)$.

In case (4) we say that an **insertion event** occurs in the transition, and we denote it by $\iota p(x,t_V)$.

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:

(5) $\forall X,T\ (\iota p(X,T) \leftrightarrow p'(X,T)\ \wedge \neg p(X,T))$.

(6) $\forall X,T\ (\delta p(X,T) \leftrightarrow p(X,T)\ \wedge \neg p'(X,T))$.

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

From the above, we then have the equivalences:

(7) $\forall X,T\ (p'(X,T) \leftrightarrow [p(X,T)\ \wedge \neg \delta p(X,T)] \vee \iota p(X,T))$.

(8) $\forall X,T\ (\neg p'(X,T) \leftrightarrow [\neg p(X,T) \wedge \neg \iota p(X,T)] \vee \delta p(X,T))$.

which relate the new state p' at valid time point T to the old state p at valid time point T and the events induced by the transaction.

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

If p is an inconsistency predicate, then ιp facts 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 facts cannot happen in any transition, since we assume that the vt-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 facts and δp facts represent the external events (given by the update) corresponding to insertions and deletions of facts, respectively. Therefore, we assume from now on that U consists of an unspecified set of insertion and/or deletion of external events.

Note that an event happens at some time instant, while the state of the database requires time intervals to express the changes produced by the transaction.

Therefore, when an insertion event $\iota p(x,t)$ happens in a transaction we really represent: $p_r(x,t,forever)$, and when a deletion event $\delta p(x,t)$ occurs in a transaction we modify $p_r(x,t_s,t_e)$ by $p_r(x,t_s,t\text{-}1)$. We consider that representation has to be transparent to the user and transaction should be closer to reality, for example, when a user offers a course s/he knows when this course starts, but probably s/he does not know when this course will end.

Example. Suppose the transaction: $\{\iota offered(databases,3), \iota takes(ton,databases,3), \delta takes(maria,logic,3)\}$ on the vt-ddb:

$offered_r(logic,2,forever)$
$takes_r(maria,logic,2,forever)$
$takes_r(jaume,logic,2,forever)$
$takes_r(jordi,logic,2,forever)$

If this transaction does not violate any integrity constraint, their representation will be:

$\iota offered(databases,3)$ .... $offered_r(databases,3,forever)$
$\iota takes(ton,databases,3)$ .... $takes_r(ton,databases,3,forever)$
$\delta takes(maria,logic,3)$ .... $takes_r(maria,logic,2,2)$

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 vt-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 vt-ddb in terms of the old state database D at valid time point T of the predicates appearing in the body of the rule, and the events that occur at valid time point T.

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

$(Q_i(X_i,T_i) \wedge \neg \delta Q_i(X_i,T_i)) \vee \iota Q_i(X_i,T_i)$

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

$(\neg Q_i(X_i,T_i) \wedge \neg \iota Q_i(X_i,T_i)) \vee \delta Q_i(X_i,T_i)$

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

We are conscious of the resulting amount of transition rules and we are working in some simplifications strategies to drastically reduce them.

Example. 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:

$ic1_1' \leftarrow takes(S,C,T) \wedge \neg \delta takes(S,C,T) \wedge \neg offered(C,T) \wedge \neg \iota offered(C,T)$.
$ic1_2' \leftarrow takes(S,C,T) \wedge \neg \delta takes(S,C,T) \wedge \delta offered(C,T)$.
$ic1_3' \leftarrow \iota takes(S,C,T) \wedge \neg offered(C,T) \wedge \neg \iota offered(C,T)$.
$ic1_4' \leftarrow \iota takes(S,C,T) \wedge \delta offered(C,T)$.
$ic1' \leftarrow ic_i' \quad i = 1,...,4$

Note that in the case of integrity constraints, $\neg ici_1'$ always holds because the vt-ddb was consistent at state *D* (for example, *takes(S,C,T)* and *¬offered(C,T)* in $ic1_1'$ cannot happen in *D*), and we can simplify 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:

$\forall X,T\ (\iota p(X,T) \leftrightarrow p'(X,T) \wedge \neg p(X,T))$.

And replacing *p'(X,T)* by its equivalent definition intuitively given in 3.2 we get:

(9) $\iota p(X,T) \leftarrow p_i'(X,T) \wedge \neg p(X,T)$ *with i = 1, ..., m*

where *m* is the number of transition rules for *p'(X,T)*.

By replacing $p_i'(X,T)$ with its equivalent definition given in the previous subsection, we get a set of **insertion events rules**. They allow us to deduce which ιp facts (induced insertions) happen in a transition. If *p* is an inconsistency predicate, ιp facts correspond to a violation of the integrity constraint.

Example. The **insertion event rules** for the integrity constraint *ic1* by applying (9), are:

$\iota ic1 \leftarrow ic1_i' \wedge \neg ic1 \ \ i=2..4$

Note that in the case of integrity constraints, $\neg ici$ always holds because the vt-ddb was consistent at state *D* and we can simplify this literal: $\iota ic1 \leftarrow ic1_i' \ \ i=2..4$

3.4 Deletion event rules

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

$\forall X,T\ (\delta p(X,T) \leftrightarrow p(X,T) \wedge \neg p'(X,T))$.

And replacing *p'(X,T)* by its equivalent definition intuitively given in 3.2 we get:

(10) $\delta p(X,T) \leftarrow p(X,T) \wedge \neg p_1'(X,T) \wedge ... \wedge \neg p_i'(X,T) \wedge ... \wedge \neg p_m'(X,T)$

By replacing *p'(X,T)* with its equivalent definition given in a previous subsection, we get a set of **deletion events rules**. They allow us to deduce which δp facts (induced deletions) happen in a transition.

Example. The **deletion event rules** for the derived predicate *some_enrol(C,T)* by applying (10), are:

$\delta some\_enrol(C,T) \leftarrow some\_enrol(C,T) \wedge \neg some\_enrol_1'(C,T) \wedge \neg some\_enrol_2'(C,T)$.

3.5 The augmented database

Let D be a vt-ddb. We denote the augmented vt-ddb, by A(D), to the vt-ddb consisting of D, its transition rules and its event rules.

Our method is based on SLDNF-resolution as it is shown in [12] that if SLDNF resolution is complete for D, then it will also be complete for A(D).

4 A method for integrity constraint checking in valid time deductive databases

The augmented vt-ddb described in the previous section can be used directly to check that a transaction does not produce inconsistencies. We propose an extension of the internal events method for checking integrity constraints in regular deductive databases [12], by including valid time.

4.1 Our method

Let *D* be a vt-ddb, *A(D)* the augmented vt-ddb, and *TR* a transaction consisting of a set of external events at valid time *T*. If *TR* leads to an inconsistency then some of the ιicn facts 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*.

Thus, our method for integrity constraints checking can be entirely based on the use of standard SLDNF resolution. We take as the input set $A(D) \cup TR$ and for each integrity *icn*, $\{\leftarrow \iota icn\}$ as goal. Transaction *TR* leads to an inconsistent vt-ddb state if there is a refutation for some of the above goals. Otherwise, *TR* can be accepted, and the vt-ddb is then updated. Note that in our method the database is updated after verification of constraint satisfaction.

4.2 Examples

Assume the example temporal deductive deductive database with the following facts, and relevant transition and event rules:

Relevant facts.

F.3 $takes_r$*(jordi,databases,2,forever).*

F.5 $takes_r$*(antoni,information systems,3,forever).*

Relevant transition rules.

T.1 $ic1_2' \leftarrow takes(S,C,T) \wedge \neg\delta takes(S,C,T) \wedge \delta offered(C,T)$.

T.5 $ic3_3' \leftarrow \iota takes(S,software\ engineering,T) \wedge takes(S,information\ systems,T1) \wedge \neg\delta takes(S,information\ systems,T1) \wedge T1<T$.

Relevant event rules.

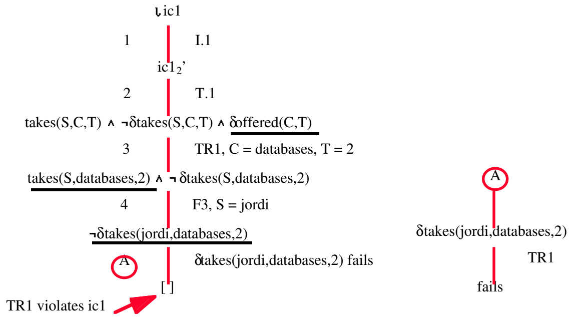
I.1 $\iota ic1 \leftarrow ic1_i'.\ \ i=2..4$

I.2 $\iota ic3 \leftarrow ic3_i'.\ \ i=2..4$

Transaction1.

TR1 is $\{\delta offered(databases,2)\}$

TR1 is an example of retroactive update. The following refutation shows that *TR1* violates the static integrity constraint *ic1* at valid time 2.



In the previous refutation, 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 *D*, respectively, and we go to the transaction or to the vt-ddb to find it, respectively.

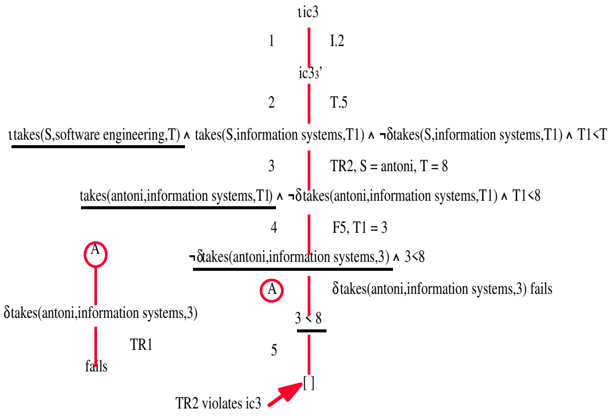
At step A, the selected literal is: *¬δtakes(jordi,databases,2)*. In order to get a successful derivation, SLDNF search space must fail finitely for the subsidiary tree of: *{←δtakes(jordi,databases,2)}*.

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).

Transaction2.

TR2 is {ιoffered(software engineering,8), ιtakes(antoni,software engineering,8)}

TR2 is an example of proactive update. The following refutation shows that *TR2* violates the dynamic integrity constraint *ic3* at valid time 8



In this case, steps 1 and 2 are SLDNF resolution steps using the rules of the augmented vt-ddb.

Step 3 can be resolved by going to the transaction to see whether if the selectionel literal is there.

In step 4 we look for a time *T1* previous to the present one, and we have to inspect the facts to see if we can find it.

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

Finally, in step 5 we resolve an evaluable predicate that takes us the succession of this branch.

4.3 Implementation of our method

Our method is based on the use of the standard SLDNF procedure for consistency checking, and a prototype has thus been implemented in Prolog.

Note that for (5) and (6) we require: $\forall$ X,T (ιp(X,T) $\leftrightarrow$ ¬p(X,T)) and $\forall$ X,T (δp(X,T) $\leftrightarrow$ p(X,T)) and we have a previous step that can reject updates that do not satisfy these conditions.

The most difficult part of this implementation has been the construction of the framework of the dynamic integrity constraints.

Dynamic integrity constraints usually use undefined valid times to refer to past or future states in addition to the current state. For example, in *ic4 ← offered(C,T)* ∧ *many_students(C,T1)* ∧ *¬many_students(C,T)* ∧ *T1>T*, *T* or *T1* appear in the transaction, but it is necessary to find the other one in the vt-ddb to check the integrity constraint.

If one do not have a temporal information that refers to the time we are searching for in the conditions of the integrity constraint, one need an operation to search for the valid times where the predicate holds in all the stored history of our deductive database. Our Prolog program resolves the problem of the search for undefined times by treating the case differently when the undefined valid time appears in the conditions of the dynamic integrity constraint as a positive literal (hold operation) and the case when it appears as a negative literal (does_not_hold operation). More detailed information about these operations can be found in [9].

5 Comparison with other methods

There are a few but not a large number of methods for integrity checking in deductive databases that incorporate time, as you can see in the bibliography of this research area, written by [6].

There are methods, such as [3], that use past temporal logic to formulate the integrity constraints. But these methods do not contain temporal information explicitly as in our case. [20] has a very hard restriction that our method does not have: facts and rules may be added or deleted from the present tense only.

Plexousakis's method [13] formulates integrity constraints in Telos [8], a hybrid language for knowledge representation. The method consists in generating a parameterized simplified structure for each literal of the integrity constraints and the deductive rules that can be affected by an update. The method for finding the simplified form is an extension of Nicolas's method [10], which includes temporal treatment.

To check integrity constraints he uses a dependency graph, which contains the parameterized simplified structures of the literals of the integrity constraints and deductive rules as nodes and the possible violations of the integrity constraints are represented by arrows.

Summarising, Plexousakis's method needs to define a parameterized simplified structure for each literal of an integrity constraint and deductive rule of the database, and it needs to construct a dependency graph for integrity constraints checking. Furthermore, the method needs to use a meta-interpreter from Telos's language to parameterize a simplified structure. In contrast, our method uses only SLDNF proof procedure directly provided in a Prolog system.

6 Conclusions and further work

In this paper we have presented a method for integrity constraints checking in vt-ddbs. Given an update, the transition rules relate the new state to the old state and the events induced by the update. Event rules define explicitly the change induced by the update on the derived predicates.

Our further work consists in completing the method for integrity constraints checking in vt-ddbs with: updates of integrity constraints and deductive rules, including modification, recursive rules, simplifications of the transition and event rules and to increase the efficiency incorporating related work in this area.

Like other temporal deductive database systems as ChronoLog [2] and ChronoBase [17], we would try to incorporate our work in the FOLRE project [19].

Furthermore, our aim is to incorporate transaction time into our method and to work with bitemporal deductive databases. A bitemporal deductive database is a deductive database that supports valid time and transaction time.

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] 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.

[2] Böhlen, M. "Managing temporal knowledge in deductive databases". PhD thesis, Swiss Federal Institute of Technology. Zürich, 1994.

[3] Chomicki, J. "Efficient checking encoding of temporal integrity constraints using bounded history encoding". ACM Transactions on Database Systems. June, 1995, pp 149-186.

[4] 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.

[5] Jensen, C.S.; Snodgrass, R.T..; Böhlen, M. "Current state of temporal data management infrastructure". Proc. of the International Workshop on Temporal Databases. Zürich. (Clifford/Tuzhilin Eds). Springer-Verlag, Sep. 1995. pp 356.

[6] Kline, N. "An update of the temporal database bibliography ". Proc. SIGMOD-RECORD. Vol. 22. Num. 4. 1993.

[7] Lloyd, J.W. "Foundations on logic programming". Second edition. Springer, 1987.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] Olivé, A. "Integrity constraints checking in deductive databases". Proc. of the 17th. Int. Conf on VLDB'91. pp 513-523.

[13] Plexousakis, D. "Integrity constraint and rule maintenance in temporal deductive knowledge bases". Proc. of the 19th. Int. Conf. on VLDB'93. pp 146-157.

[14] Sarda, N.L. "HSQL: A historical query language". In [18], pp 110-140.

[15] Snodgrass, R.; Ahn, I. "Temporal databases". IEEE Computer. Vol 19. Num 9. September, 1986.

[16] Sancho, M.R; Olivé, A. "Deriving transactions specifications from deductive conceptual models of information systems". Proc. of CAiSE'94 conference. 1994, pp 311-324.

[17] Sripada, S.M.. "Design of the ChronoBase temporal deductive database system". Proc. of the ARPA/NSF Int. Conf. on an Infrastructure for Temporal Databases. Arlington, 1993. Edited by Richard Snodgrass.

[18] Tansel, A.U.; Clifford, J.; Gadia, S.; Jajodia, S.; Segev, A.; Snodgrass, R. "Temporal databases: theory, design and implementation". Benjamin/Cummings. 1993.

[19] 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.

[20] Wüthrich, B. "Large deductives databases with constraints". PhD thesis, Swiss Federal Institute of Technology. Zürich, 1991.