The Ramification Problem in Temporal Databases: a Solution Implemented in SQL

Nikos Papadakis, Dimitris Plexousakis, Grigoris Antoniou, Manolis Daskalakis, Yannis Christodoulou

Department of Computer Science, University of Crete, and
Institute of Computer Science, FORTH, Greece
npapadak,dp,antoniou,mdaskal,jchrist@ics.forth.gr

Abstract. In this paper we elaborate on the handling of the ramification problem in the setting of temporal databases. Starting with the observation that solutions from the literature on reasoning about action are inadequate for addressing the ramification problem, in our prior work we have presented a solution based on an extension of the situation calculus and the work of McCain and Turner. In this paper, we present a tool that connects the theoretical results to practical considerations, by producing the appropriate SQL commands in order to address the ramification problem.

1 Introduction

Many domains about which we wish to reason are dynamic in nature. *Reasoning about action* [10, 8] is concerned with determining the nature of the world (what holds in the world state) after performing an action in a known world state, and has found application in areas such as cognitive robotics [10].

The guarantee of consistency of data that is stored in a database is a very important and difficult problem. The consistency of data is determined by the satisfaction of the integrity constraints in the different databases states (situations). A database state is considered valid (consistent) when all integrity constraints are satisfied. New situations arise as the result of action (transaction) execution. In a new situation (which includes the direct effects of the transactions) the database may be inconsistent because some integrity constraints are violated by means of indirect effects of actions. Thus, it is necessary to produce all indirect effects in order to determine the satisfaction of the integrity constraints. In a large database system with hundreds or thousands of transactions and integrity constraints, it is extremely hard for the designer to know all the effects that actions may have on the consistency of the database. The task can be greatly facilitated by a tool that systematically produces all effects (direct and indirect). The designers will be thus given the ability to realize the effects of their design, correct errors and prevent potential inconsistency problems.

We can assume that a atomic transaction is an action. In this context, the *ramification problem* [7] is concerned with the indirect effects of actions in the presence of constraints. Standard solutions to this problem [2–6] rely on the persistence of fluents, and on the idea that actions have effects on the next situation only.

In our work we consider the ramification problem in a temporal setting. In this context, actions are allowed to have effects which may commence at a time other than the next time point, and the effects may hold only for certain time. For example, assume that if a public employee commits a misdemeanor, then for the next five months she is considered illegal, except if she receives a pardon. When a public employee is illegal, then she must be suspended and cannot take promotion for the entire time interval over which she is considered illegal. Also, when a public employee is suspended, she cannot receive a salary until the end of the suspension period. Each public employee is graded for her work. If she receives a bad grade, then she is considered a bad employee. If she receives a good grade, then she is considered a good employee and she may take a bonus if not suspended. Each public employee receives an increase and a promotion every two and five years, respectively, if not illegal.

We can identify six actions, $misdemeanor$, $take\_pardon$, $good\_grade$, bad_grade, $take\_promotion$ and $take\_increase$ and seven fluents $good\_employee$, $illegal$, $take\_salary$, $take\_bonus$, $position$, $suspended$ and $salary$. The fluent $position(p, l, t_1)$ means that the public worker is at position l for the last t_1 months while $salary(p, s, t_1)$ means that the public worker has been receiving salary s for the last t_1 months. The direct effects of the six actions are expressed in propositional form by the following rules:

$$occur(misdemeanor(p), t) \rightarrow illegal(p, t_1) \wedge \ t_1 \in [t, t+5] \quad (1)$$
$$occur(take\_pardon(p), t) \rightarrow \neg illegal(p, t_1) \wedge t_1 \in [t, \infty) \quad (2)$$
$$occur(bad\_grade(p), t) \rightarrow \neg good\_employee(p, t_1) \wedge t_1 \in [t, \infty) \quad (3)$$
$$occur(good\_grade(p), t) \rightarrow good\_employee(p, t_1) \wedge t_1 \in [t, \infty) \quad (4)$$
$$occur(take\_increase(p), t) \wedge salary(p, s, 24) \rightarrow salary(p, s+1, 0) \quad (5)$$
$$occur(take\_promotion(p), t) \wedge position(p, l, 60) \rightarrow position(p, l+1, 0) \quad (6)$$

where t is a temporal variable and the predicate $occur(misdemeanor(p), t)$ denotes that the action $misdemeanor(p)$ is executed at time t. The former four rules are dynamic and executed every time that the corresponding actions take place. The remaining two are also dynamic but are executed periodically because the corresponding actions take place periodically.

Also we have and the following integrity constraints which give rise to indirect effects of the six actions.

$$illegal(p, t_1) \rightarrow suspended(p, t_1) \quad (7)$$
$$suspended(p, t_1) \rightarrow \neg take\_salary(p, t_1) \quad (8)$$
$$\neg suspended(p, t) \wedge good\_employee(p, t) \ \rightarrow \ take\_bonus(p, t) \quad (9)$$
$$\neg good\_employee(p, t_1) \rightarrow \neg take\_bonus(p, t_1) \quad (10)$$
$$\neg suspended(p, t_1) \rightarrow take\_salary(p, t_1) \quad (11)$$

The rules (7-11) are static and executed every time. This happens because there are effects of the actions which hold for a time interval (e.g. the effect $illegal$ of the action $misdemeanor$ which hold 5 time point after the execution of the action). After the end of the time intervals the effects pause to hold without an action taking place. So, in a temporal setting the main assumptions of previous solutions to the ramification problem are inadequate because they

based in persistent of fluent - nothing change unless an action takes place. Thus, we need new techniques. In [9] the problem has been addressed in cases where actions result in changes in the future (e.g. the promotion of an employee takes effect in two months).

In this paper, we propose a tool which produces sql commands(transcations) which in turn produce the indirect effects of other transactions. These transactions ensure that all integrity constraints will be satisfied after an update take place. The data is stored in a relational database (Oracle). The temporal data are stored in a table as triples (ID, $timestamp\_start$, $timestamp\_end$).

Our approach is based on the situation calculus [7] and the work of McCain and Turner [6]. We extended the approach of [6] by introducing duration to fluents, and by considering temporal aspects. As we have explained in [9], in a temporal database we need to describe the direct and indirect effects of an action not only in the immediately resulting next situation, but possibly for many future situations as well. This means that we need a solution that separates the current effects(dynamic rules) from the future effects (static rules). This is necessary because another action may occur between them which cancels the future effects. The approach of [6] permits this separation.

The paper is organized as follows. Section 2 describes the technical preliminaries, that are based on previous work by the authors [9]. Section 3 presents the algorithm which produces the in sql commands and a description of the system architectures.

2 Technical Preliminaries

Our solution to the ramification problem is based on the situation calculus [7]. However it is necessary to extend the situation calculus to capture the temporal phenomena, as done in the previous work [9]. For each fluent f, an argument L is added, where L is a list of time intervals $[a, b], a < b$. $[a, b]$ represents the time points $\{x | a \leq x < b\}$. The fluent f is true in the time intervals that are contained in the list L.

We define functions $start(a)$ and $end(a)$, where a is an action. The former function returns the time moment at which the action a starts while the latter returns the time moment at which it finishes. Actions are ordered as follows: $a_1 < a_2 < .... < a_n$, when $start(a_1) < start(a_2) < ..... < start(a_n)$. Actions(instantaneous) $a_1, a_2, ..., a_n$ are executed concurrently if $start(a_1) = start(a_2) = ... = start(a_n)$.

The predicate $occur(a, t)$ means that the action a is executed at time moment t. We define functions $start(S)$ and $end(S)$, where S is a situation. The former function returns the time moment at which the situation S starts while the latter returns the time moment at which it finishes.

We define as temporal situation a situation which contains all fluents with the list of time intervals in which the fluent is true. For the rest of the paper we refer to a temporal situation as situation. The function $FluentHold(S, t)$ returns the set of all fluents that are true in the time moment t. For function fluent the $FluentHold(S, t)$ returns the value that the function has at time point t. The situation S is a temporal situation. We define as non-temporal situation S in

a time point t the situation $S = FluentHold(S', t)$. A transformation from a situation to another could happen when the function $FluentHold(S, t)$ returns a different set. [1]

We define two types of rules: (a) dynamic $occur(a, t) \rightarrow f([..])$ which are executed when the action a takes place; and (b) static rules $G_f([a, b]) \rightarrow f([...])$ which are executed at time point t if the fluent f is not true in the time interval $[a, b]$. [2] The execution of a dynamic or static rule has as consequence the transition to a new situation. When an action takes place we want to estimate all direct and indirects effects of the action which has as conclusion a new consistent situation.

We define as *legal* (*consistent*) a situation in which all integrity constraints are satisfied. Also, each function fluent has only one value at each time point.

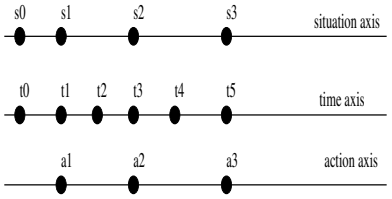


Fig. 1. The correspondence among Time-Actions-Situations

As already mentioned, the previous approaches to solve the ramification problem are inadequate in the case of temporal databases. We overcome these difficulties with the time - actions - situations correspondence that appears in figure 1. There are three parallel axes: the situations axis, the time axis and the actions axis. The database changes into a new situation when an action takes place or a static rule evaluated or cease to hold a fluent.

We base our work on the ideas of McCain and Turner [6] who propose to use *static rules* to capture the indirect effects of actions (based on integrity constraints in the particular domain), and *dynamic rules* to represent the direct effects of actions. In our approach, for each action A there is a dynamic rule of the form $A \rightarrow \bigwedge F_i(L_i')$, where each $F_i(L_i')$ is $f_i(L_i')$ or $\neg f_i(L_i')$ for a fluent f. These rules describe the direct effects of an action. Here is an example:

$occur(misdemeanor(p), t) \rightarrow illegal(p, [[t, t + 5]]))$

In addition, for each fluent f we define two rules, $G(L) \rightarrow f(L)$ and $B(L) \rightarrow \neg f(L)$. $G(L)$ is a fluent formula which, when is true (at list L), causes fluent f to become true at the time intervals contained in the list L (respectively for $B(L)$). These rules encapsulate the indirect effects of an action. Here is an example: $illegal(p, [[2, 7]]) \rightarrow suspended(p, [[2, 7]])$

One cornerstone of our previous work [9] is the production of the static rules from integrity constraints, according to the following ideas. We make use of a binary relation I which is produced from the integrity constraints and encodes

[1] For example if $\{f_1([[7, 9]], \neg f_1([[10, \infty]]), ...\}$ means that at time point 10 we have transform from one situation to another.

[2] $G_f([a, b])$ mean that the fluent formula G_f is true at time interval $[a, b]$.

the dependencies between fluents. The binary relation I is based on the idea that there are two kinds of integrity constraints.

$$(a) \quad G_f \rightarrow K_f \quad (b) \quad G_f \equiv K_f,$$

where G_f and K_f are fluent propositions. For the first kind of constraints, for each $f \in G_f$ and $f' \in K_f$ the pair (f, f') is added in I. For the second kind of constraints, for each $f \in G_f, f' \in K_f$, if f can change its truth value as the direct effect of an action, then (f, f') is added in I. If f' can change its truth value as direct effect of an action then (f', f) is added in I.

The algorithm from producing static rules has been presented in [9]. The main idea is the following: when an integrity constraint is violated, find an executable static rule whose effect is to restore the violation of the constraint. The algorithm is proceedd as follows

1. At each time point t if the situation is S do
   (a) If an action $occur(a, t)$ take place at time point t then
   (b) Execute the dynamic rule which referred in action a
   (c) While there is a static rule r which is executable. Execute the r.

The following theorem has been proved.

Theorem 1. *(a) When a static rule is executable at least one integrity constraint is violated. (b) The algorithm for execution dynamic and static rules always returns a consistent situation.*

3 Transforming the static rules into SQL

In a relational temporal database a fluent of the form $f_1([[a, b], [c, d], [e, f]])$ is stored in a table as triples (ID, *timestamp_start*, *timestamp_end*). The type of ID is Number while the type of *timestamp_start*, *timestamp_end* is date or timestamp. In this assumption the fluent $f_1([[a, b], [c, d], [e, f]])$ will be stored in three rows (triples): $(1, a, b)$,$(1, c, d)$, $(1, e, f)$.

In order to transform and execute a rule of the form $f_1(...) \wedge f_2(...) \rightarrow f_3(...)$ in SQL we must determine the intersection of the time intervals in which the fluents f_1 and f_2 are true. In order to do that we must check the all rows which refer to f_1 with all rows which refer to f_2 and find the all the intersections of the time intervals. For example if there are two rows (1,'26-7-2007','31-7-2007') and (2,'29-7-2007','6-8-2007') then the intersection is the time interval ('29-7-2007','31-7-2007'). This means that the row (3,'29-7-2007','31-7-2007') must be inserted in the table. If we insert in the table the row (1, '4-8-2007','14-8-2007') we now have the rows: (1, '4-8-2007','14-8-2007') and (2,'29-7-2007','6-8-2007'). This means that directly after the insertion of the row (1, '4-8-2007','14-8-2007') we must insert the row (3,'4-8-2007','6-8-2007'), as conclusion of the execution of the above static rule.

In order to capture all cases of relationships between two temporal intervals we must implement all the 13 Allen temporal relations [1]. First we need to describe how we estimate the intersections between two sets of temporal intervals(e.g $f1([[..], [...], ..])f2([[..], ...])$, which refer to two different fluents. As we have already described above, the two sets are stored in a table as triples (ID, *timestamp_start*, *timestamp_end*). The following algorithm estimates the intersection between two sets of temporal intervals (the implementation in SQL).

Algorithm 1

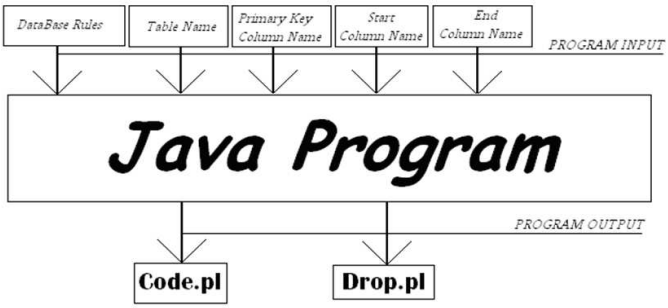
1. Take the first pair (f_i, f_j) of fluents in the left side of the static rule.
2. For each rows $row11, row22$ s.t $row1.ID = i$ and $row2.ID = j$ do
3. s1 := row1.START; e1 := row1.FINISH; s2 := row2.START; e2 := row2.FINISH; changes := FALSE;
4. IF $s1 <$ $s2$ AND $e1 >=$ $s2$ AND $e1 <=$ $e2$ THEN
   resSt := s2; resEnd := e1; changes := TRUE;
5. ELSIF $s1 >=$ $s2$ AND $s1 <=$ $e2$ AND $e1 >=$ $s2$ AND $e1 <=$ $e2$ THEN
   resSt := s1; resEnd := e1; changes := TRUE;
6. ELSIF $s1 <=$ $s2$ AND $e2 <=$ $e1$ THEN
   resSt := s2; resEnd := e2; changes := TRUE;
7. ELSIF $s1 >=$ $s2$ AND $s1 <=$ $e2$ AND $e1 >$ $e2$ THEN
   resSt := s1; resEnd := e2; changes := TRUE;
   END IF;
8. IF changes = TRUE THEN
   INSERT INTO TABLENAME_temporary_table VALUES (currTempTableId, resSt, resEnd) ;
   END IF;

The step 4 is the case of overlaps, the step 5 is the case of during, the step 6 is the cases of equals (when hold the equal in two parts of expression) and during (when does not hold the equal in one of the two parts of the expression). The other relations are implemented in more than one steps of the algorithms. As can be seen from step 8, when a new temporal interval is estimated it is stored in a temporary table. First, we estimate the Allen relations between the time intervals of the two fluents(e.g f_1, f_2). Then, consider these two fluents as one (f_{12}) and repeat the process between this "new" fluent and the next fluent (e.g f_3). Thus we have to execute a static rule of the form $f_1(...) \wedge f_2(...) \wedge ... \rightarrow f_k$, first we execute the algorithm 1 for the first two fluents ($f_1 f_2$). Then we consider the result as a new fluent and repeat the execution of the algorithm 1 with the next fluent. When we have to execute a static rule of the form $[f_1(...) \wedge ....] \vee [f_2(...) \wedge ...] \vee .. \rightarrow f_k$ the only thing we need to do is to execute the algorithm 1 for each part of the disjoin. The architecture of the tool which produce the SQL commands is shown in figure 2. A general description of the java logical unit follows. The input data include:

- the rules of the DB, that is the cross-correlations of fields of the DB, in form $ID1 \wedge ID2(\hat{}IDi) \rightarrow IDx$; where $ID(1, 2, i)$ are the values of the primary key field that is involved in the section for the result and IDx is the value of the primary key field with which will the result be stored in the DB.
- The name of table, as String.
- The name of column of key, any type.
- The name of the starting time column, any type.
- The name of the ending time column, any type.

The output includes:

- Code.pl: a file with SQL commands, which when it is executed in the DB in question will create the essential procedures that will safeguard its integrity.
- Drop.pl: a file with the essential SQL commands, so that the tables and procedures that were created by the program can be erased.



DataBase Rules:	Syntax: id ^ id [^id]* -> id;
Table Name:	String, the name of the database table.
Primary Key:	Name of the column of the table that is the primary key (in any format).
Start:	Name of the column of the table, containing the starting time (in any format).
End:	Name of the column of the table, containing the ending time (in any format).
Code.pl:	Output file, containing pl/sql code, which follows the input rules.
Drop.pl:	Output file, containing pl/sql code that deletes any processes and tables created by code.pl.

Fig. 2. The architecture of the systems

4 Conclusions

In this paper we studied the ramification problem in the setting of relational temporal databases. More specifically, we proposed a solution and a tool which implements the solution in SQL. The key ideas of our approach are: (a)to extend the situation calculus, (b)to establish suitable correspondences between time, actions and situations, (c)to use dynamic rules to capture the direct effects of actions, and static rules to capture the indirect effects of actions, (d)to develop a system which implement the solution in sql for a relation databases.

In future work we intend to extend our system for the case that the updates refer to the past and for the case where a distinction of integrity constraints into strict and soft exists.

References

1. J. F. Allen. Maintaining Knowledge about Temporal Intervals. Communications of the ACM, Vol. 26 No. 11, pp. 832843, 1983.
2. Marc Denecker and Eugenia Ternovska. Inductive situation calculus,Artificial Intelligence archive Volume 171 , Issue 5-6, Pages: 332-360 April 2007
3. Conrad Drescher and Michael Thielscher. Integrating Action Calculi and Description Logics,KI 2007: Advances in Artificial Intelligence, Pages 68-83, August 2007.
4. Laura Giordano1, Alberto Martellib and Camilla Schwindc. Specialization of Interaction Protocols in a Temporal Action Logic, Pages 3-22,LCMAS 2005.
5. A. Fusaoka. Situation Calculus on a Dense Flow of Time,AAAI-1996, pp. 633-638.
6. N. McCain and H. Turner. A causal theory of ramifications and qualifications. Proceedings of IJCAI-95, pp. 1978-1984.
7. J. McCarthy and P.J. Hayes. Some philophical problem from the standpoint of artificial intelligence, Machine Intelligence 4, pp. 463-502. American Elsevier, 1969.
8. Rob Miller and Murray Shanahan. The Event Calculus in Classical Logic - Alternative Axiomatisations, Link. Elect. Art. in Compt. and Inform. Sc. 4(16), 1999.
9. Nikos Papadakis and Dimitris Plexousakis. Action with Duration and Constraints: The Ramification problem in Temporal Databases. International Journal of Artificial Intelligent Tools(IJTAI) pp. 315-353, volume 12, Number 3, September 2003.
10. Raymond Reiter, Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press, 2001