Actions with Duration and Constraints: the Ramification Problem in Temporal Databases

Nikos Papadakis, Dimitris Plexousakis
Department of Computer Science, University of Crete, and
Institute of Computer Science, FORTH, Greece
{npapadak, dp}@csd.uch.gr

Abstract

The ramification problem is a hard and ever present problem in systems exhibiting a dynamic behavior. The area of temporal databases in particular is still lacking satisfactory solutions to the ramification problem. In this paper, we address the ramification problem based on causal relationships that take time into account. We study the problem for both instantaneous actions and actions with duration. The proposed solution advances previous work by considering actions with effects occurring in any of the possible future situations resulting from an action's execution.

1 Introduction

The *ramification problem* is a well-known in AI [12] problem that arises in databases [14], robotics, software engineering [1] and all systems exhibiting a dynamic behavior [18]. In most of these disciplines, the ramification problem has been either ignored or by passed by means of implicit assumptions about the way that things change. We argue that significant benefit can be obtained by studying the problem in the context of systems that represent and reason with a changing world. In this paper, we consider the case of temporal databases, which notably lacks solutions to the ramification problem and its related *frame* and *qualification* problems [12].

We introduce this problem by means of an example. Assume we are interested in maintaining a database that describes a simple circuit, which includes two switches and one lamp (figure 1 (A)). The circuit's behavior is described by a set C comprising the following integrity constraints. First, when the two switches are up, the lamp must be lit. Second, if one switch is down, then the lamp should not be lit. The integrity constraints are expressed as the following formulae employing predicates up and $light$ with the obvious meaning: $C = \{up(s_1) \wedge up(s_2) \supset light, \neg up(s_1) \supset \neg light, \neg up(s_2) \supset \neg light\}$. Action $toggle\_switch$ changes the state of a switch as the following set E of propositions describing the direct effect of the action specify: $E = \{toggle\_switch(s) \supset up(s) \quad if \quad \neg up(s), toggle\_switch(s) \supset \neg up(s) \quad if \quad up(s)\}$.

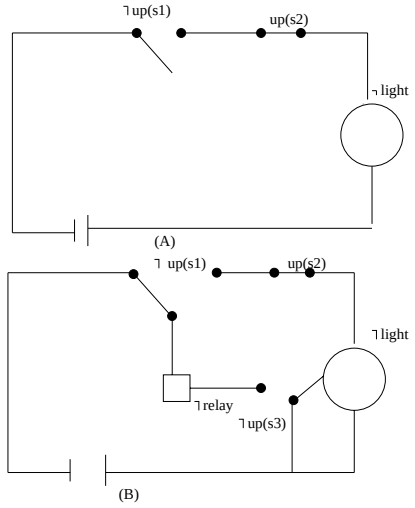

Figure 1. Simple circuits

A situation is called consistent when it satisfies all integrity constraints. Assume that the circuit is in situation $S = \{\neg up(s_1), up(s_2), \neg light\}$. We can easily see that S is consistent. Now assume that we execute the action $toggle\_switch(s_1)$. This action has as its direct effect to change the state of switch s_1 from $\neg up(s_1)$ to $up(s_1)$ leading to the situation $S_1 = \{up(s_1), up(s_2), \neg light\}$. This situation is inconsistent because it violates the first integrity constraint. The reasonable conclusion is that the lamp must be lit. So the final situation is $S_2 = \{up(s_1), up(s_2), light\}$. The change of the condition of the lamp is the indirect effect of the action $toggle\_switch(s_1)$. Notice that the indirect effects occur because of the presence of integrity constraints. The ramification problem refers to the concise description of the indirect effects of an action in the presence of constraints.

The rest of the paper is organized as follows: in section 2, we briefly review solutions which have been proposed for solving the ramification problem in the context

Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'02)
1082-3409/02 $17.00 © 2002 IEEE

of conventional (non-temporal) databases and describe an algorithm that discovers dependencies between fluents. In section 3, we define the ramification problem in temporal databases and we present prevalent previous work relevant to this problem. In section 4 , we propose an extension to the situation calculus for addressing the ramification problem in temporal databases. In section 5, we deal with the ramification problem in temporal databases when actions execute sequentially. We examine two cases, namely the case of instantaneous actions and the case of actions with duration. Finally, in section 6, we address the ramification problem in temporal databases when actions execute concurrently.

2 The Ramification Problem

The majority of proposed solutions are based on the Situation Calculus [12]. The situation calculus is a second-order language that represents the changes which occur in a domain as results of actions. One possible evolution of the world is a sequence of actions and is represented by a first-order term. The situation at which no action has occurred yet is called the initial situation (S_0). A binary function $do(a, S)$ yields the new situation that results from the execution of action a in the situation S. Predicates whose truth value may change from one situation to another are called *fluents*. Similary, functions whose values are situation-dependent are called *functional fluents*.

Among the simplest solutions are those based on the *minimal change* approach [21]. These suggest that, when an action occurs in a situation S, one must find the consistent situation S' which has as few changes from S as possible. In the example of section 1, the minimal change approach cannot distinguish between the two situations $S_1 = \{up(s_1), up(s_2), light\}$ and $S_2 = \{\neg up(s_1), up(s_2), \neg light\}$ as they both are equally close to the original situation S. However, we would like to be able choose S_1 because it is reasonable to light the lamp rather than to toggle switch s_2.

Other proposals are based on the *categorization of fluents* [8, 9]. Fluents are categorized into *primary* (those that can change only as the direct effect of an action) and *secondary* (those that can change both as direct and indirect effects of an action). In the previous example, primary and secondary fluents are $F_p = \{up(s_1), up(s_2)\}$ and $F_s = \{light\}$ respectively. When an action occurs in a situation S, one must find the consistent situation S' which has as few changes as possible from S in the primary fluents. Hence, we choose situation S_1, because it has no changes in the primary fluents whereas S_2 has one ($u(s_1)$). The categorization of fluents solves the ramification problem only when all fluents can be categorized. Sometimes however, some fluents are primary for some action and secondary for others. In that case, the above solutions are not adequate.

The most effective solutions are based on *causal relationships* [2, 10, 11, 3, 19, 20, 21]. Each causal relationship has the form

$$\epsilon \;\; causes \;\; \rho \;\; if \;\; \Phi\,,$$

where ϵ is an action, ρ is the indirect effect and Φ is the context, i.e., a set of fluents that describes the conditions under which the execution of ϵ leads to ρ. In our example, we have four such causal relationships:

$$toggle\_switch(s_1) \;\; causes \;\; light \, if \;\; \neg up(s_1) \wedge up(s_2)$$
$$toggle\_switch(s_2) \;\; causes \;\; light \, if \;\; \neg up(s_2) \wedge up(s_1)$$
$$toggle\_switch(s_1) \;\; causes \;\; \neg light \;\; if \;\; up(s_1)$$
$$toggle\_switch(s_2) \;\; causes \;\; \neg light \;\; if \;\; up(s_2)\,.$$

All the proposed solutions determine the direct and indirect effects of an action that refer to the next consistent situation. As we can observe from the above examples, the change of fluent f's truth value potentially affects the truth-value of some other fluents, while it does not affect others. We define a binary relation I between fluents as follows: if $(f, f') \in I$, then a change in fluent f's value may affect the value of f'. In the above example, $(up(s_1), light) \in I$, whereas $(up(s_1), up(s_2)) \notin I$. The causal relationships are defined only for the pairs of fluents that belong in I. This means that in the example with the single switch there is a causal relationship between the action $toggle\_switch(s_1)$ and fluent $light$, but there is no such relationship between $toggle\_switch(s_1)$ and fluent $up(s_2)$. In the next subsection, we propose an algorithm for the construction of I.

2.1 Fluent Dependencies

Assume that we have two kinds of integrity constraints: (a) $G_f \supset K_f$ and (b) $G_f \equiv K_f$, where G_f and K_f are fluent propositions. The difference between the two kinds is that, for the latter, when $\neg G_f$ holds then $\neg K_f$ also holds, whereas this is not necessarily the case for the former. For the first kind of constraints, for each $f \in G_f$ and $f' \in K_f$ we add the pair $(f, f') \;\; in \;\; I$. Notice that $(f', f) \notin I$ (because $K_f \not\supset G_f$). For the second kind of constraints we make the following hypothesis: The change of the truth value of a fluent belonging to G_f is expected to affect the truth values of some fluents belonging to K_f, while it is not expected to affect the truth values of other fluents which belong to G_f. We make the same hypothesis for the fluents of K_f. Now we can construct the set I. For each pair of fluents f, f', such that $f \in G_f$ and $f' \in K_f$ we add $(f, f') \;\; and \;\; (f', f) \;\; to \;\; I$. Consider the circuit in figure 1 (B). The integrity constraints specifying the behavior of this system are expressed as the following formulae:

$(a) \;\; light \equiv up(s1) \wedge up(s2)$

$(b) \;\; relay \equiv \neg up(s1) \wedge up(s3), \;\; (c) \;\; relay \supset \neg up(s2)$

By applying this procedure the set I is constructed as follows: for constraint (a) we conclude that $(up(s1), light)$, $(up(s2), light)$, $(light, (up(s1))$,$(light, (up(s2))$ must be added in I. From rule (b) we obtain $(up(s1), relay)$,

Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'02)
1082-3409/02 $17.00 © 2002 IEEE

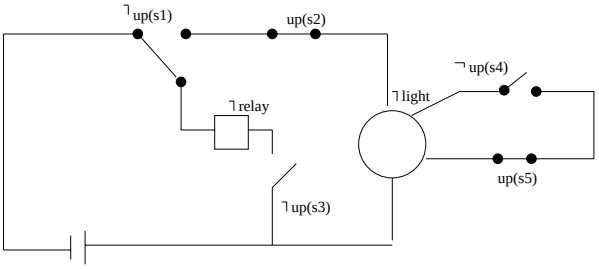

Figure 2. A more complex circuit

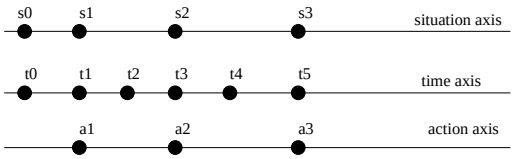

Figure 3. The Time-Situation-Actions Correspondence

$(up(s3), relay)$, $(relay, (up(s1))$,$(relay, (up(s3))$ in I and from rule (c) we obtain $(relay, up(s2))$ in I. Because of our hypothesis, $(up(s1), light)$ $\in$ I, while $(up(s_1), up(s_2))$ $\notin$ I. Assume that the circuit is in the situation that is depicted in figure 1 (B). The action $toggle\_switch(s_1)$ has as indirect effect to light the lamp and not to toggle the switch s_2. We observe that it is not reasonable to include the fluent pairs $(light, (up(s1))$,$(light, (up(s2))$, $(relay, (up(s1))$,$(relay, (up(s3))$ in I. The truth values of fluents $light$ and $relay$ cannot change as the direct effect of an action, so they cannot affect the truth values of other fluents. In order to eliminate them from set I we modify the algorithm as follows:

- For each $f \in G_f, f' \in K_f$, where $G_f \equiv K_f$ is a specified constraint do
  - **{** If f can change its truth value as the direct effect of an action, then add (f, f') in I. If f' can change its truth value as direct effect of an action then add (f', f) in I.

In our example, the above change is right if and only if each of the fluents $light$ and $relay$ appear in one only rule of the form $G_f \equiv K_f$. For example, consider that the circuit in figure 2. The integrity constraints specifying the behavior of this system are expressed as the following formulae:

$(a)\ light \equiv up(s1) \wedge up(s2), (b)\ light \equiv up(s4) \wedge up(s5),$

$(c)\ \ relay \equiv \neg up(s1) \wedge up(s3),\ \ (d)\ \ relay \supset \neg up(s2)$

Applying the procedure described above yields $(up(s1), light), (up(s2), light), (up(s4), light), (up(s5),$ $light), (up(s1), relay), (up(s3), relay), (relay, up(s2)) \in$ I. Assume that the circuit is in the situation depicted in figure 2. Then, after the execution of action $toggle\_switch(s_4)$, because $(up(s4), light)$ $\in$ I, the fluent $light$ changes from $\neg light$ to $light$. Because $(light, up(s1)), (light, up(s2))$ $\notin$ I, the fluents $up(s1), up(s2)$ do not change. This means that the circuit will be in situation $\neg up(s1), up(s2), up(s4), up(s5), \neg up(s3), \neg relay, light$, which violates the rule (a). Assume now that the integrity constraints specifying the behavior of this system are expressed as the following formulae:

$(a)\ \ light \equiv (up(s1) \wedge up(s2)) \vee (up(s4) \wedge up(s5))$

$(b)\ \ relay \equiv \neg up(s1) \wedge up(s3),\ \ (c)\ \ relay \supset \neg up(s2)$

In the above specification of constraints, the fluent $light$ is only in one constraint of type $G_f \equiv K_f$ and the modified algorithm behaves correctly.

3 Temporal Databases and Ramifications

To address the ramification problem in temporal databases we need to incorporate time in the situation calculus. Some works have suggested some ways for inserting time in the situation calculus by drawing a correspondence between situation calculus and a linear time line [13, 5, 15, 16, 4]. This correspondence is defined between real situations and time. This is not absolutely correct because the situation calculus supports parallel histories of situations. The above weakness can be overcome by defining a correspondence between a branching time structure and situations. The branching time structure creates many parallel histories of situations. As we show in the sequel, we extend this approach for addressing the ramification problem in temporal databases. We describe the problem in this context with an example.

Assume that when a driver p drinks alcohol then s/he is considered drunk for the next five hours. In this time span other actions may occur leading to many different situations. In all these situations the fluent $drunk(p)$ must be true. After five hours the fluent $drunk(p)$ must become false and, thus, the database must change into a new situation without any action taking place. The action $drive(p)$ cannot be executed if $drunk(p)$ holds. The causal relationships cannot solve the ramification problem because they determine the direct and indirect effects only for the next situation. The same weakness characterizes all other solutions of the ramification problem in conventional databases.

The above weakness can be alleviated by constructing a correspondence between situations and actions with time. We adopt the correspondence shown in figure 3 [13]. There are three parallel axes: the first is the situations axis, the second is the time axis and the third is the actions axis. For now, we assume that all actions are instantaneous. When an action takes place, the database changes into a new situation.

We assume a discrete model of time in which each timestamp specifies a point in time or moment. Each action occurs at a specific time point. When an action a_2 occurs at time

Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'02)
1082-3409/02 $17.00 © 2002 **IEEE**

point t_3 in a situation S_1, a new situation $S_2 = do(a_2, S_1)$ results. Hence, at each time moment, we must determine the truth value of fluents.

The most prevalent previous works are those by Reiter [17], Reiter and Pinto [15, 16] and by Kakas [6, 7]. Reiter has suggested an extension of the situation calculus in order to encapsulate time and axioms which ensure that in each legal situation all natural actions have been executed. A natural action is an action which executes in a predetermined time moment except if some other action has changed the time of execution. Reiter has extended the fundamental axioms of the situation calculus in order to determine which fluent is true at each time moment. The problem addressed is the frame[1] rather than the ramification problem. However the work of Reiter sets the basis for encapsulating time in the situation calculus. In this paper, we propose a futher extension of the situation calculus based on Reiter's proposal. Kakas [6, 7] proposed the language E which contains a set ϕ of fluents, a set of actions, and a partially ordered set of time points. E employs the following axiom schemas for the description of the world (assume L and F are fluents, T is a time point, A is an action and C is a set of fluents).

$$L \ holds \ at \ T, \quad A \ happens \ at \ T$$
$$C \ intiates \ or \ terminates \ F \ when \ C,$$
$$L \ whenever \ C, \quad A \ needs \ C$$

As we may observe, the third axiom is dynamic because it executes when an action executes, while the last two are static because they execute each time moment. In E one cannot declare effects that persist over a time span as in the aforementioned example where, if someone drinks then s/he is drunk for the subsequent five hours. Also, E cannot represent delayed effects, as e.g., if someone drinks alcohol then s/he becomes drunk after half an hour and remains drunk for the next five hours. The language E is based on the assumption that fluents persist until their truth value is changed. We consider these assumptions rather strong and examine the problem in a strictly more general setting.

4 Extended Situation Calculus

We extend the temporal situation calculus as follows:

- For an action a, functions $start(a)$ and $end(a)$ return the time moment at which the action a starts and the time moment at which it ends respectively.
- For a situation S, functions $start(S)$ and $end(S)$ return the time moments at which situation S begins and ends respectively.
- The functional fluent $f_\alpha(a)$ is defined as $current\_moment - start(a)$, i.e., the duration of execution of action a until the present moment.
- Time is discrete and isomorphic to the natural numbers.
- A fluent f is represented as $f(t')$, meaning that the fluent f is true in the interval $[current\_moment, current\_moment + t']$. $\neg f(t')$ means that f is false in the interval $[current\_moment, current\_moment + t']$. As time progresses, t' is decreased by one time unit.
- Actions are ordered as follows:

  For instantaneous actions $start(a) = end(a)$ and
  $$a_1 < a_2 < .... < a_n \ \ , when$$
  $$start(a_1) < start(a_2) < ..... < start(a_n)$$
  Two actions a_1, a_2 execute concurrently when $start(a_1) = start(a_2)$.

  For actions with duration, $a_1 < a_2$ when $end(a_1) < start(a_2)$. Two actions a_1, a_2 execute concurrently when $start(a_1) \leq start(a_2) \leq end(a_1) \leq end(a_2)$ holds. We assume that all actions which execute at the same time moment execute concurrently.
- Function do is defined as $do : action^n \times situation \rightarrow situation$. $do(\{a_1, a_2, .., a_n\}, S) = S_1$ means that the actions $a_1, a_2, .., a_n$ execute concurrently in the situation S and the result is the situation S_1.
- For two situations S_1, S_2, $S_1 < S_2$, when $end(S_1) \leq start(S_2)$. It is not necessarity the case that $S_2 = do(\{a_{i1}, a_{i2}, ...\}, ..., do(\{a_{j1}, ...\}, S_1)...)$.
- We extend predicate $poss(a, S)$ as follows: $poss(\{a_1, a_2, ...a_n\}, S) = \bigwedge_{i=1,...,n} poss(a_i, S)$, meaning that actions $\{a_1, a_2, ...a_n\}$ can execute concurrently iff all their preconditions are true.

4.1 Fundamental Axioms

We use the axioms defined by Reiter [17]

$$S_0 \neq do(\{a_1, ..., a_n\}, S) \qquad (1)$$
$$do(\{a_1, ..., a_n\}, S) = do(\{a_1^{'}, ..., a_n^{'}\}, S^{'}) \supset$$
$$\{a_1, ..., a_n\} = \{a_1^{'}, ..., a_n^{'}\} \wedge S = S^{'} \qquad (2)$$
$$start(a(t)) = t \qquad (3)$$
$$(\forall P).P(S_0) \wedge (\forall \{a_1, ..., a_n\}, S)\ [P(S) \supset$$
$$P(do(\{a_1, ..., a_n\}, S)] \supset (\forall S)P(S) \qquad (4)$$

Axiom (4) is an inductive axiom which means that each situation is the result of the execution of a sequence of actions. Thus, if the intial situation is khown we can determine which fluent is true in each situation. This axiom does not hold in our case because the transition from one situation to the next does not necessarily happen after the execution of an action. In order for the above axiom to hold, we define a natural action a_f for each fluent f. The only direct effect

[1] The problem of determining which predicates and functions are not affected when an action is executed is the **frame problem** [12].

Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'02)
1082-3409/02 $17.00 © 2002 **IEEE**

of the action a_f is that the fluent f becomes false ($f(0)$). Hence, when an action a has as effect $f(10)$ the action a_f will execute 10 time moments later. Natural actions do not affect the world being modeled. They are employed to ensure that the transition from one situation to the next is the result of the execution some action (natural or not). The transition from one situation to the next happens when the truth value of at least one fluent changes. By the inclusion of natural actions no fluent can change its truth value without some action taking place.

4.2 Axioms for the Description of the World

For each action A we define one axiom of the form

$$A \supset \bigwedge L_i(t') \,,$$

where $L_i(t')$ is $f_i(t')$ or $\neg f_i(t')$. Such an axiom describes the direct effects of an action. For each fluent f we define two axioms

$$G(t) \supset f(t'), \quad K(t) \supset \neg f(t') \,,$$

where $G(t)$ is a formula which when true causes fluent f to become true at the next t' time moments (respectively for $K(t)$). These axioms encapsulate the indirect effects of an action. Axioms of the former type are dynamic (they are evaluated after the execution of an action), while those of the latter are static (they are evaluated at each time point).

5 Sequential Action Execution

In this section we address the ramification problem in temporal databases when actions execute sequentially. This solution extends the solution which has been proposed in [11]. Each action A is represented as $A(t)$ which means that the action A executes at time t. For each action A, we define one axiom of the form

$$A \supset \bigwedge L_i(t')$$

For each fluent f we define two axioms

$$G(t) \supset f(1), \quad K(t) \supset \neg f(1) \,,$$

where $G(t)$ is a formula which when true causes fluent f to become true at the next time moment (respectively for $K(t)$). The systematic generation of these axioms solves the ramification problem in temporal databases, beause the last two axioms encapsulate the indirect effects not only for the next time moment, but for each time moment. We need $O(A+2*F)$ such axioms, where A is the number of actions and F the number of fluents.

Consider the following example: if a public employee commits a misdemeanor, then for the next five months s/he is considered illegal, except if s/he receives a pardon. When a public employee is illegal, then s/he must be suspended for the entire time interval over which s/he is considered illegal. These are expressed by the following constraints assumming that the time granularity is that of months[2]:

$occur(misdemeanor(p), t) \supset illegal(p, t_1) \wedge \ t_1 < t + 5$

$take\_pardon(p, t) \supset \neg illegal(p, \infty)$

$illegal(p, t_1) \supset suspended(p, t_1)$,

where t and t_1 are temporal variables and the predicate $occur(misdemeanor(p), t)$ denotes that the action $misdemeanor(p)$ occurs at time t. 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. In the above example, the action $misdemeanor(p)$ has the indirect effect that the public worker is in suspension for the next five months. The dynamic and static axioms are

$occur(misdemeanor(p), t) \supset illegal(p, 5)$

$take\_pardon(p, t) \supset \neg illegal(p, \infty)$,

$illegal(p, t_1) \supset suspended(p, 1)$.

The first axiom is dynamic and "encapsulates" the direct effects, while the second and third are static and "encapsulate" the indirect effects of the action at time t.

5.1 Production of Static Rules

The static rules encapsulate the indirect effects of the execution of each action. Indirect effects exist because of the presence of integrity constraints. Hence, it is reasonable to produce the static rules from the integrity constraint. Static rules are produced as follows:

1. Transform each integrity constraint in its CNF form. Now each integrity constraint has the form $C_1 \wedge C_2 \wedge C_3 ..... \wedge C_n$, where each C_i is a disjunct.

2. For each i from 1 to n do
   (a) assume $C_i = f_1 \vee .... \vee f_m$
   For each j from 1 to m do
   For each k from 1 to m, and $k \neq j$, do
   if $(f_j, f_k) \in I$ then
   $R = R \vee (\neg f_j \ \ causes \ \ f_k \ \ if \ \ \bigwedge \neg f_l), \ l = 1, ..m, l \neq j, k.$

3. For each fluent f_k the rules have the following form
   $\bigwedge f_i \ \ causes \ f_k \ \ if \ \ \Phi, \quad \bigwedge f'_i \ \ causes \ \neg f_k \ \ if \ \ \Phi'$
   We change the static rules from $G_{f_k} \supset f_k$ and $K_{f_k} \supset \neg f_k$ into $G'_{f_k} \supset f_k$ and $K'_{f_k} \supset \neg f_k$ (respectively), where $G'_{f_k} = G_{f_k} \vee (\bigwedge f_i \wedge \Phi)$, and $K'_{f_k} = K_{f_k} \vee (\bigwedge f'_i \wedge \Phi')$.

4. For each rule $G_{f_p} \supset f_p$, we replace each fluent f with $f(t)$, as has been defined above. Static rules have the form $G_p(t) \supset f_p(1)$.

The proposition $G_{f_p}(t)$ could contain information which permit us to understand that the fluent f_p is true for a time interval greater than one time unit. In that case, it is not necessary to execute all static rules at every time unit. We change the static rules in order to encapsulate the above observation. The rules change from $G_{f_p}(t) \supset f_p(1)$ to $G_{f_p}(t, t') \supset f_p(t')$, where $G_{f_p}(t, t')$ means that, if G_{f_p} is true at time moment t, then the fluent f_p is true for the next t' time units:

[2]Quantifiers are committed in the expression of these propositions. They are considered to be implicitly universally quantified over their temporal and non-temporal arguments.

Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'02)
1082-3409/02 $17.00 © 2002 IEEE

1. For each static rule $G(t) \supset f$ do
   (a) let $G = G_1 \vee .... \vee G_n$
   (b) For each j from 1 to n do
      - let $G_i = f_1(t_1) \wedge .... \wedge f_n(t_n)$
      - let $t = min(t_1, ..., t_n)$
      - replace G_i with $G_i(t)$
   (c) $t' = max(t_i : G_i \quad is \quad true)$
   (d) replace $G(t)$ with $G(t, t')$

5.2 Dynamic ans Static Rule Execution

As we noted in the previous section, if the proposition $G_{f_p}(t)$ contains information which permit us to understand that the fluent f_p is true for a time interval greater than one time unit, then it not necessary to execute the static rule $G_{f_p}(t) \supset f_p$ at every time moment. The generated rules are evaluated as follows:

1. After the execution of one action execute the dynamic rule which references this action.
2. At each time moment do
   (a) If no action takes place at this moment do
      i. Execute all static rules except these which have as conclusion fluents which are true for time greater than one time unit.
      ii. if a fluent $f(t)$ becomes true after an execution of a static rule, then set $\neg f(0)$ to true.
      iii. Repeat until no change occurs.
   (b) If at this time moment an action takes place do
      i. Execute all static rules.
      ii. if a fluent $f(1)$ becomes true after an execution of a static rule, then set $\neg f(0)$ to true.
      iii. Repeat until no changes occur.

In the aforementioned example with the public employee, the static rule now changes to:

$$illegal(p, t) \wedge (t > 0) \supset \ suspend(p, t)\,.$$

This means that the rule need not be evaluated at each time moment. We have proved the following result:[3]

Theorem 1 *At each time unit, the algorithms terminate in a finite number of steps and return a consistent situation.*

5.3 Actions with Effects in Future Situations

The ramification problem becomes more complex when the direct and indirect effects of an action arise after some time moments and not necessarily at the immediate next. For instance, assume that if someone drinks alcohol, s/he becomes drunk after 30 minutes and remains drunk for a time interval of 5 hours. In this case, the above representation of fluents cannot encapsulate the direct and indirect effects of action $drink\_alcohol$. We change the representation of fluents as follows: each fluent f is represented as $f(L)$, where $L = [[t, t'], ...]$ is a list whose members are time intervals $[t, t']$, meaning that the fluent is true at each such time interval $[t, t']$. At each time moment $t^{''}$, we ignore (and delete from the list) the time intervals ($[t, t']$), which refer in the past ($t^{''} > t'$). Now the rules (dynamic and static) need to change in order to encapsulate the above change.

The static rules are produced from the same algorithm as before. At each time moment at which it is necessary to execute a static rule, prior to the evaluation of the rule, execute the following algorithm

1. At time moment t, for the static rule $G(t, t_1) \supset f$ do
   (a) let $G = G_1 \vee .... \vee G_n$
   (b) For each j from 1 to n do
      - let $G_i = f_1([..]) \wedge .... \wedge f_n([..])$
         i. for each fluent $f_i(L)$ take the first element $[t^{'}, t^{''}]$ of the list L.
         ii. if $t^{'} > t$ then G is false and the algorithm terminates.
         iii. else $t_i = t^{''} - t$.
      - let $t_{min} = min(t_1, ..., t_n)$
      - replace G_i with $G_i(t, t + t_{min})$

When a static rule $G(t, t^{'}) \supset f(L)$ is evaluated, the element $[t, t^{'}]$ is added in the list L and it is removed from L', where L' is the list of time intervals that the fluent $\neg f$ is true. More specific we add some subsets of $[t, t^{'}]$ in the list L, so that after that the list L contains the interval $[t, t^{'}$, and we remove some subsets of $[t, t^{'}]$ ($L \cap [[t, t^{'}]] = [t, t^{'}]$) from the list $L^{'}$ so that after this preceess the list $L^{'} \cap [[t, t^{'}]] = \emptyset$.

The same algorithm applies to dynamic rules as well. The algorithm for evaluating the dynamic and static rules does not change. We have proved the following result:

Theorem 2 *The algorithm for evaluating the dynamic and static rules always retun a legal situation when the direct and indirect effects refer to future states.*

Consider the example with the drunk driver. Assume that the time granularity is that of an hour. The dynamic and static rules are:

$$driver(p) \wedge drink\_alcohol(p, t) \ \supset$$
$$drunk([t + 0.5, t + 0.5 + 5])$$
$$drunk(L) \supset \neg drive(L)\,.$$

As we can observe, with the above rules we can capture all direct and indirect effect of the action $drink\_alcohol$. The rules are not evaluated at each time moment, but only one time after the execution of action $drink\_alcohol$. This is enough for ensuring that the indirect effects referring to the future time intervals are captured correctly.

5.4 Actions with Duration

So far, we have assumed that actions are instantaneous. In the case of actions with duration, all effects must be determined with reference to the start, the end and the duration

[3] Proofs are ommitted due to lack of space

Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'02)
1082-3409/02 $17.00 © 2002 **IEEE**

of action. If all direct and indirect effects can be described by reference to the start and the end of the action, then we can assume that one action with duration is equivalent to two instanteous actions, one for the start and one for the end. In that case, the dynamic rules must be defined as for instantaneous actions. The previously presented algorithms can be used to address the ramification problem without change.

In the case that the effects of an action depend on its duration, the above solution cannot solve the ramification problem. Consider the example that someone who drinks alcohol for more than 10 consentive minutes is drunk, otherwise s/he is not. Usually, the duration of an action is unknown before the end. So we cannot describe the direct and indirect effects of an action with reference to its start and to its end. Hence, we must change the dynamic and static rules. The fluent representation does not change. For each action a, we define a new fluent $f_a(t,x),\ \ x \in \{0,1\}$, such that when $x = 1$ then t is the duration of action until now. If $x = 0$ then the execution of the action has completed. At the start of the execution of action a we set $f_a(0,1)$ (as direct effect) and at the end we set $f_a(t,0)$.

The fluent f_a helps us determine the indirect effects of an action which depends on the duration of action a. All direct effects of an action do not depend on the duration of the action's execution. We must change the static rules in order to encapsulate the fluents f_a. The following algorithm incorporates this change.

1. For each static rule $G(t,t') \supset f$ do
   (a) let $G = G_1 \vee .... \vee G_n$
   (b) For each action a which can affect the fluent f if executed for more than one time instant do
      i. set $G' = G \vee (f_{a_i}(t,x) \wedge (t \geq b))$
      ii. set $G = G'$
   (c) let $G = G_1 \vee .... \vee G_m$
   (d) For each j from 1 to m do
      - let $G_i = f_1(t_1) \wedge .... \wedge f_n(t_n)$
      - let $t = min(t_1, ..., t_n)$
   (e) set $t'' = max(t_i : G_i \quad is \quad true)$
   (f) replace $G(t,t')$ with $G(t,t'')$

The above algorithm "adds" at each static rule the effects which depend on the duration of an action. The algorithm for evaluating the dynamic and static rules does not need to change. The following result has been shown to hold:

Theorem 3 *The algorithm always return a legal situation in case that the effect of an action depends on its duration.*

Consider again the example of the drunk driver. Assume that a driver is drunk if s/he drinks alcohol for more than 10 minutes. We define the fluent $f_{drink\_alcohol}$ as we explained above. Now the granularity of time is the minute. The dynamic and static rules are:

$driver(p) \wedge f_{drink\_alcohol}(t,1) \wedge (t > 10) \supset drunk([now, now + 5])$

$driver(p) \wedge end_{drink\_alcohol}(t) \wedge f_{drink\_alcohol}(t',1) \wedge (t' > 10) \supset drunk([t, t + 5])$

$drunk(L) \supset \neg drive(L)$,

where $end_{f_{drink\_alcohol}}$ is the end of action $drink\_alcohol$. The algorithm checks a static rule $G_f \supset f$ iff the fluent f is false. Thus, the first rule will be executed for each time moment that the fluent $drunk$ is false. This means that if a driver drinks for 5 hours and 30 minutes, the rule executed each time minute before the 10 minutes. In the last execution the driver was drunk for 5 hours (thus the fluent $drunk$ becomes true for the next five hours). Hence, the rule will be executed again after 5 hours (5 hours and 10 minutes from the start of action drink).

6 Concurrent execution of instantaneous actions

In this section, we examine the case that two or more instantaneous actions can execute concurrently. The direct and indirect effects of an action do not start necessarily from the next time moment. This means that two or more actions cannot necessarily be executed concurrently if the preconditions holds. It must be determined that the direct and indirect effects of these actions are consistent not only in the next time moment but in the future, as well.

For example a person cannot work in the public and in the private sector at the same time. Suppose the actions $hire\_in\_public$ and $hire\_in\_company$ are defined:

$hire\_in\_public(p,t) \supset public\_worker(p,[t + 10, \infty])$

$hire\_in\_company(p,t) \supset private\_employee(p,[t + 10, \infty])$

This means that the employment started ten time moments after the action took place. The integrity constraints

$public\_worker(p,L) \supset \neg private\_employee(p,L)$

$private\_employee(p,L) \supset \neg public\_worker(p,L)$

denote that the actions $hire\_in\_public$ and $hire\_in\_company$ cannot execute concurrentily. The following algorithm addresses the ramification problem.

1. Before an action's execution check the preconditions. If some precondition does not hold, reject the action.
2. After the execution of concurrent actions execute the dynamic rules which refer to those actions.
3. Execute the algorithm of the evaluation of static rules (see below)
4. If the algorithm of the evaluation of static rules returns inconsistency, then reject the last action, else continue.
5. Until some other action executes use the situations which have been estimated by the algorithm of the execution of static rules.

Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'02)
1082-3409/02 $17.00 © 2002 **IEEE**

The following is the algorithm for the execution of static rules. This algorithm cannot change the direct effects of the actions.

1. After the execution of the rule $G_f \supset f(L_1)$, if $\neg f(L_2)$ and $L_1 \cap L_2 \neq \{\}$ then
   (a) if $L_1 \cap L_2$ does not belong in the direct effects of one action set $L_2 = L_2 \setminus (L_1 \cap L_2)$ and execute the rule $K_f \supset \neg f(...)$.
   (b) else return inconsistency
2. Repeat step 1 until L_1 and L_2 do not change or until they take previous values. In the second case return inconsistency.
3. Repeat the step 1 and 2 for all rules.
4. Repeat the step 1,2 and 3 for all time moments at which there are references.

In the above example assume that we try to execute concurrentily the actions $hire\_in\_public$ and $hire\_in\_company$. Note that inconsistency does not exist in the next time moment but after 10 minutes. The algorithm of the execution of static rules detects an inconsistency.

7 Summary and Future Research

The ramification problem in temporal databases is a complex and many-faceted problem. We have described a solution to this problem for the cases that the effects of an instantaneous action (direct and indirect) refer to the current and future situations. Also we have described a solution for these cases when the actions have durations.

Further research includes the study of the problem for concurrent actions with duration and non-deterministic actions, as well as the problem of changing time granularities. Another direction is the study of the problem in the case of actions changing our beliefs about the past. In this case, effects may be periodically recursive and one needs to be able to determine what is allowed to change in the past and what isn't. The related qualification problem, which refers to determininig the preconditions which must hold prior to the execution of an action, is a topic of current research. We are studying the extension of the proposed framework for solving the qualification problem by defining static rules specifying when actions become disqualified.

References

[1] A. Borgida, J. Mylopoulos and R. Reiter. On the Frame Problem in Procedure Specifications. IEEE Transactions on Software Engineering, 21(10), Oct. 1995, pp.785-798.

[2] C. Elkan. Reasoning about action in first order logic. Proceedings of the Conference of the Canadian Society for Comptutational Studies in Intelligence (CSCSI), pp 221-227, Vancouver, May 1992.

[3] M. Ginsberg and D. Smith. Reasoning about action I: A possible worlds approach. Artificial Intelligence, 35:165-195, 1988.

[4] J. Gustafon. Extending Temporal Action Logic for Ramification and Concurency, Thesis No 719 of Linkoping Studies in Science and Technology, 1998.

[5] A. Fusaoka. Situation Calculus on a Dense Flow of Time, Proceedings of AAAI-96, pp. 633-638, 1996

[6] A.C. Kakas, R.S. Miller and F. Toni, E-RES: Reasoning about Actions, Events and Observations, in Proceedings of LPNMR2001, pp. 254-266, Springer Verlag, 2001.

[7] Antonis Kakas and Rob Miller, A Simple Declarative Language for Describing Narratives with Actions, The Journal of Logic Programming, Vol 31(1-3) (Special Issue on Reasoning about Action and Change), pages 157-200, Elsevier, 1997.

[8] V. Lifshitz. Towards a metatheory of action. In J.F. Allen, R. Fikes, and E. Sandewall, editors, Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, pages 376-386, Cambridge, MA, 1991.

[9] V. Lifshitz. Frames in the space of situations, Artificial Intelligence, 46:365-376, 1990.

[10] V. Lifschitz. Towards a metatheory of action. Proceedings of KR'91, pp. 376-386, Cambridge, MA, 1991.

[11] N. McCain and H. Turner. A causal theory of ramifications and qualifications. Proceedings of IJCAI-95, pp. 1978-1984, Montreal, Canada, August 1995.

[12] J. McCarthy and P.J. Hayes. Some philophical problem from the standpoint of artificial intelligence. In B. Meltzer and D. Mitchie, editors, Machine Intelligence 4, pp. 463-502. American Elsevier, 1969.

[13] Nikos Papadakis and Dimitris Plexousakis. Action Theories in Temporal Databases. Proceedings of the 8th Panhellenic Conference on Informatics, pp. 254-264, Cyprus, Nov. 2001.

[14] Dimitris Plexousakis, John Mylopoulos. Accomodating Integrity Constraints During Database Design. Proceedings of EDBT 1996, pp. 497-513, Avignon, France, 1996.

[15] J. Pinto. Temporal Reasoning in the Situation Calculus. Ph.D. Thesis, Dept. of Computer Science, Univ. of Toronto, Jan. 1994.

[16] J. Pinto and R. Reiter. Temporal Reasoning in Logic Programming: A Case for the Situation Calculus, Proceedings of 10th Int. Conf. on Logic Programming, Budapest, Hungary, June 21-24, 1993.

[17] R. Reiter. Natural Actions, Concurrency and Continous Time in the Situation Calculus, KR 96, pages 2-13, 1996.

[18] R. Reiter. Khowledge in Action: Logical Foundation for specifying and implemending Dynamical Systems, MIT Press, 2001.

[19] M. Thielscher. Ramification and causality. Artifical Intelligence, 89(1-2):317-364, 1997.

[20] M. Thielscher. Reasoning about actions: Steady versus stabilizing state constraints. Artifical Intelligence, 104:339-355, 1988.

[21] M. Winslett. Reasoning about action using a possible models approach. Proceedings of AAAI-88, pp. 89-93, Saint Paul, MN, August 1988.

Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'02)
1082-3409/02 $17.00 © 2002 **IEEE**