Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 59 (2006) 397–434
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

The ramification problem in temporal databases: Changing beliefs about the past

Nikos Papadakis *, Grigoris Antoniou, Dimitris Plexousakis

Department of Computer Science, University of Crete, Institute of Computer Science, FORTH, GR-711 10 Heraklion, Crete, Greece

Received 21 February 2005; accepted 21 September 2005
Available online 18 October 2005

Abstract

In this paper we study the ramification problem in the setting of temporal databases. Standard solutions from the literature on reasoning about action are inadequate because they rely on the assumption that fluents persist, and because actions have effects on the next situation only. In this paper we provide a solution to the ramification problem based on an extension of the situation calculus and the work of McCain and Turner. More specifically, we study the case where the effects of an action refer to the past, a particularly complex problem.

Keywords: Ramification problem; Temporal databases; Common sense reasoning; Knowledge representation and reasoning; Belief revision

1. Introduction

The ramification and qualification problems [9] are hard problems that arise in robotics, software engineering, in databases, and generally speaking all systems exhibiting a dynamic behavior. In this paper we consider the case of temporal databases.

Let us illustrate the problems. Suppose we are interested in maintaining a database that describes a simple circuit, which has two switches and one lamp (Fig. 1A).

The circuit's behavior is described by 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 must not be lit. The integrity constraints are expressed as the following formulas, employing predicates *up* and *light*:

$$up(s_1) \wedge up(s_2) \equiv light$$
$$\neg up(s_1) \supset \neg light$$
$$\neg up(s_2) \supset \neg light$$

* Corresponding author.
E-mail addresses: npapadak@csd.uch.gr (N. Papadakis), ga@csd.uch.gr (G. Antoniou), dp@csd.uch.gr (D. Plexousakis).

0169-023X/$ - see front matter
doi:10.1016/j.datak.2005.09.003

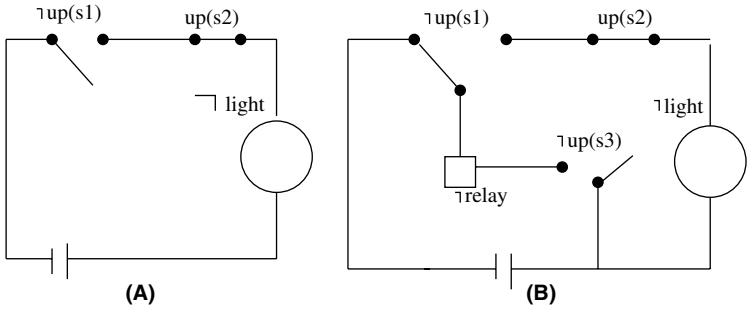

Fig. 1. The circuit.

Action *toggle_switch* changes the situation of a switch as follows:

$$toggle\_switch(s) \supset up(s) \quad \text{if } \neg up(s)$$
$$toggle\_switch(s) \supset \neg up(s) \quad \text{if } up(s)$$

The above propositions describe the direct effects of the action *toggle_switch*. A situation is called consistent when it satisfies all integrity constraints. Assume that the circuit is in situation $S = \{up(s_1), \neg up(s_2), \neg light\}$. The situation S is consistent, because it satisfies all integrity constraints. Now assume that we execute the action $toggle\_switch(s_2)$. This action has a sdirect effect to change the state of switch s_2 from $\neg up(s_2)$ to $up(s_2)$. Now the situation of the circuit is $S_1 = \{up(s_1), up(s_2), \neg light\}$. 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 $toggle\_switch(s_2)$. Notice that indirect effects exist 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*.

Several ways for addressing the ramification problem have been suggested in the literature. The majority of them are based on the situation calculus [9]. 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). There is a binary function $do(a, S)$ which yields the new situation resulting from the execution of action a in the situation S. Predicates, called fluents, may change truth value from one situation to another. Similarly, one can represent functions whose values are dependent on the situations on which they are evaluated (functional fluents).

The simplest of the technique suggested in the literature is the *minimal-change* approach [21]. Thiss suggests that, when an action occurs in a situation S, we need to find a consistent situation S' which has fewer changes from the situation S (S' is closer to S than to any other situation).

Another solution is the *categorization* of fluents [5–7]. Fluents are categorized as primary and secondary. A primary fluent may change only as a direct effect of an action, while a secondary fluent may change only as an indirect effect of an action. After an action takes place, we choose the situation with the fewest changes in primary fluents. The categorization of fluents solves the ramification problem only if all fluents can be categorized. If some fluents are primary for some actions and secondary for some others this solution is not satisfactory.

As we can observe from the above examples, the change of fluent f's truth value potentially affects the truth value of some fluents, while it does not affect that of the 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 $(up(s_1), up(s_2)) \notin I$. A fluent could change or remain unchanged after an action. This depends on the context in which an action takes place.

Causal relationships [8,19,20] capture this dependence between an action and an indirect effect. A causal relationship has the form

$$\epsilon \ causes \ \rho \quad \text{if } \Phi$$

where ϵ is an action, ρ is the indirect effect and Φ is the context. The context is a fluent formula. Each causal relation must be evaluated, after the execution of the action ϵ, if and only if the context is true. The binary relation I defines the dependence that exist between context Φ and fluent ρ.

In the above example, there are four causal relationships

$$up(s1) \ causes \ light \quad \text{if } up(s2)$$
$$up(s2) \ causes \ light \quad \text{if } up(s1)$$
$$\neg up(s1) \ causes \ \neg light \quad \text{if } \top$$
$$\neg up(s2) \ causes \ \neg light \quad \text{if } \top$$

In our work we adopt the idea of causal relationship and use it in the context of temporal databases. Let us illustrate the problem by means of an example. Assume that if a public employee commits a misdemeanor, then for the next 5 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 and cannot be promoted for the entire time interval over which s/he is considered illegal. Also, when a public employee is suspended, s/he cannot receive a salary until the end of the suspension period. Each public employee is graded for his/her work. If s/he receives a bad grade, then s/he is assumed to be a bad employee. If s/he receives a good grade, then s/he is assumed as a good employee and s/he may take a bonus if not suspended. Each public employee receives an increase and a promotion every 2 and 5 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*(p,l), *suspended* and *salary*(p,s). 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:[1]

$$occur(misdemeanor(p),t) \supset illegal(p,t_1) \wedge t_1 \in [t,t5] \tag{1}$$
$$occur(take\_pardon(p),t) \supset \neg illegal(p,t_1) \wedge t_1 \in [t,\infty) \tag{2}$$
$$occur(bad\_grade(p),t) \supset \neg good\_employee(p,t_1) \wedge t_1 \in [t,\infty) \tag{3}$$
$$occur(good\_grade(p),t) \supset good\_employee(p,t_1) \wedge t_1 \in [t,\infty) \tag{4}$$
$$occur(take\_increase(p),t) \wedge salary(p,s,24) \supset salary(p,s+1,0) \tag{5}$$
$$occur(take\_promotion(p),t) \wedge position(p,l,60) \supset position(p,l+1,0) \tag{6}$$

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 preconditions of the actions *take_increase*(p) and *take_promotion*(p) are

$$Poss(take\_increase(p),t) \equiv salary(p,s,24) \wedge \neg illegal(p,t) \ \wedge \ good\_employee(p,t)$$
$$Poss(take\_promotion(p),t) \equiv position(p,l,60) \wedge \neg illegal(p,t) \ \wedge \ good\_employee(p,t)$$

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

$$illegal(p,t_1) \supset suspended(p,t_1) \tag{7}$$
$$suspended(p,t_1) \supset \neg take\_salary(p,t_1) \tag{8}$$
$$\neg suspended(p,t) \wedge good\_employee(p,t) \supset take\_bonus(p,t) \tag{9}$$
$$\neg good\_employee(p,t_1) \supset \neg take\_bonus(p,t_1) \tag{10}$$
$$\neg suspended(p,t_1) \supset take\_salary(p,t_1) \tag{11}$$

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

The solutions which have been proposed for the ramification problem in conventional databases [13,1,5,7,15,19–21] cannot solve the ramification problem in temporal databases because they determine the direct and indirect effects only for the next situation. Also they assume that fluents persist, that is, no change in their truth value occurs without an action taking place.

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 also 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 5 months. In these 5 months, the action *good_grade*, could occur, but even if this happens, the employee cannot take up a promotion. This means that the world changes situations while the direct and indirect effects of some action still hold. Also, in this time span, other actions may occur leading to many different situations. Furthermore, 5 months after the execution of the action *misdemeanor* the situation changes without an action taking place (because the public worker is no longer considered illegal). This means that the transition from one situation to the next could happen without an action taking place. Hence, fluents cannot be assumed to persist until an action changes their truth value.

In our previous work [13,14] we have addressed the ramification problem in a temporal database when change refers to the future (an outline of the approach is present in Section 3.1). Here we study the problem where change may refer to the past. This case is more complex and poses various problems. We extend the remainder of our previous technique to solve the ramification problem in this case. This paper is structured as follows: in Section 2 we give some definitions, in Section 3 we review the previous work. In Section 4 we address the ramification problem in the case that an action could change our belief about the past. In Section 5.1 we present a solution for the case that an action can change the truth value of all fluents in the past. In Section 5.2 we present a solution for the case that an action can change the truth value only of some fluents in the past. In Section 5.3 we present a solution for the case that an action can change the truth value of all fluents in the past but its effects start to hold from the current time point. Finally in Section 6 we present a brief summary and future work.

2. Definitions

In this section we extend the situation calculus [9] in order to solve the ramification problem in case an action could change some beliefs about the past.

- Each fluent f is represented as $f(L)$, which means that fluent f is true in the time intervals that are contained in list L. Each element of list L is a time interval $[a,b]$, $a<b$. $\neg f(L')$ means that fluent f is false in the time intervals that are contained in list L'. It must hold that $L\cap L'=\emptyset$. List L (or L') is the last element of the fluent. This mean that each n-place fluent becomes an $n+1$-place fluent, the $n+1$st argument being the list L. The other elements do not change.
- We define functions *start*(a) and *end*(a), where a is an action. The former function returns the time moment at which action a starts while the latter returns the time moment at which it finishes.
- Actions are ordered as follows: For *instantaneous actions*[2] $a_1<a_2<\cdots<a_n$, when $start(a_1)<start(a_2)<\cdots<start(a_n)$. Also, for instantaneous actions, $start(a)=end(a)$ holds. In this case, two actions a_1,a_2 will be executed concurrently when $start(a_1)=start(a_2)$ holds.
- We define the predicate $occur(a,t,t_1)$ which means that action a is executed at time moment t but the effects of it is referred to time moment t_1. In the case that $t_1<t$ the action changes the *belief about the past*.
- We define function *start*(S) and *end*(S), where S is a situation. The former function returns the time moment at which situation S starts while the latter returns the time moment at which it finishes.
- We define the function $FluentHold(S,t)$ which returns the set of all fluents which are true in the time moment t.[3]

[2] In this paper we consider only instantaneous actions.

[3] Notice that the above representation allows one situation to contain information that some fluents will be true in the future, e.g., $FluentHold(\{f_1([5,9]),f([10,20])\},6)=\{f_1\}$.

- We define the functional fluent $f_{\alpha}(a)$ as $current\_moment - start(a)$, that is, the duration of execution of action a until the present moment.
- We extend predicate $poss(a, S)$ as follows:
  $poss(\{a_1, a_2, \ldots, a_n\}, S) = \bigwedge_{i=1,\ldots,n} poss(a_i, S)$. This means that the actions $\{a_1, a_2, \ldots, a_n\}$ can execute concurrently if and only if the preconditions of each action are true.
- We define as a legal (consistent) situation, a situation in which all integrity constraints are satisfied.

3. Previous work

3.1. Our previous work

In our previous work [13,14] we have dealt with the case that actions may only change the future. As we have already said, the previous approach to solve the ramification problem is inadequate in the case of temporal databases. They fall short of adequately addressing the problem in a temporal context because they only determine the direct and indirect effects of actions for the subsequent situation. Also they are based on the persistence of fluent assumption (i.e., no fluent may change the truth value without an action taking place).

The above weaknesses can be alleviated by constructing a correspondence between situations and actions with time. Some proposal for that has been done in [8,17,18,12,11,14]. We suggest the correspondence that appears in Fig. 2. There are three parallel axes: the situations axis, the time axis and the actions axis. When an action takes place, the database changes into a new situation.

We extended a previous proposal by McChain and Turner [8]. McChain and Turner suggest that for each action A there is a dynamic rule

$$occur(A) \supset \bigwedge F$$

which denotes the direct effects of action. Also for each fluent f there are two static rules, one for it and one for its negation

$$G \supset f$$
$$B \supset \neg f,$$

where G and B are fluent formulas. The static rules show the indirect effects.

We extend the above proposal as follows: An action A is represented as $A(t)$ which means that action A is executed at time t. For each action A we define one axiom of the form

$$A \supset \bigwedge F_i(L_i'),$$

where $F_i(L_i')$ is $f_i(L_i')$ or $\neg f_i(L_i')$. The above rules describe the direct effects of an action. For each fluent f we define two rules

$$G(t, t') \supset f([t, t'])$$
$$B(t, t') \supset \neg f([t, t'])$$

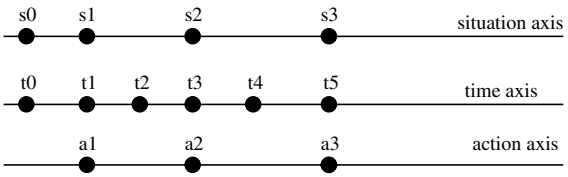

Fig. 2. The correspondence among time–actions–situations.

where $G(t,t')$ is a formula which, when true (at time point t), causes fluent f to become true at the time interval $[t,t']$ (respectively for $B(t,t')$). The above symbolism is equivalent with $G(t,L') \supset f(L')\ B(t,L') \supset \neg f(L')$, where $L' = [t,t']$. These rules encapsulate the indirect effects of an action. The former rules are dynamic because they are evaluated after the execution of an action, while the latter are static because they are evaluated at every time point at which the corresponding fluent is false.

Consider the public employee example from the introduction. In that example the dynamic rules are rules (1)–(6), while the static rules are

$$\begin{aligned} R = \{&illegal(p,L) \supset suspended(p,L),\ suspended(p,L) \supset \neg take\_salary(p,L),\\ &\neg suspended(p,L_1) \wedge good\_employee(p,L_2) \supset take\_bonus(p,L_1 \cap L_2),\\ &\neg good\_employee(p,L) \supset \neg take\_bonus(p,L), \neg suspend(p,L) \supset take\_salary(p,L)\} \end{aligned}$$

The static rules encapsulate the indirect effects of an action, which are caused by the existence of integrity constraints. The indirect effects of an action ensure that after the execution of an action the integrity constraints are satisfiable in the new situation. Thus the static rules must be derived in such a way that when an integrity constraint is not satisfied in a situation produced by application of dynamic rules at least one static rule is executable. After execution of static rules, the corresponding integrity constraints will be satisfied. The basic idea we propose is to translate each integrity constraint into CNF form

$$C_1 \wedge \cdots \wedge C_n, \quad \text{where } C_i \equiv f_{1i} \vee \cdots \vee f_{mi},$$

and to ensure that whenever a C_i is false, static rules can be executed and make C_i true. To ensure this there must be at least one static rule of the form

$$\neg\left[\bigvee f_p \quad \text{s.t. } p \in \{1i,\ldots,mi\} \quad \text{and} \quad p \neq ki\right] \vee \mathrm{FL} \supset f_{ki}$$

where FL is a fluent formula and $ki \in \{1i,\ldots,mi\}$. After the execution fluent f_{ki} will be true, thus the C_i will be true. If the above happens for each C_i of each integrity constraint then the integrity constraint will be satisfiable after the execution of the static rules.

One cornerstone of our work is the production of the static rules from integrity constraints, according to the above ideas. We make use of a binary relation I which is produced from the integrity constraints and encodes the dependences between fluents (see Section 4 for details). The algorithm for producing static rules is the following:

1. Transform each integrity constraint into its CNF form. Now each integrity constraint has the form $C_1 \wedge C_2 \wedge C_3 \cdots \wedge C_n$, where each C_i is a disjunct on fluents.
2. Set $R = \{False \supset f,\ False \supset \neg f$:for each fluent $f\}$
3. For each i from 1 to n do: assume $C_i = f_1 \vee \cdots \vee f_m$
   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
   $R = R \cup (\neg f_j\ causes\ f_k$ if $\bigwedge \neg f_l), l \neq j,k.$
4. For each fluent f_k the rules[4] in R have the following form:
   $\bigwedge f_i\ causes\ f_k$ if Φ, $\quad \bigwedge f'_i\ causes\ \neg f_k \quad$ if Φ'
   We change the static rules from $G \supset f_k, K \supset \neg f_k$

   $$to\ \left(G \vee \left(\bigwedge f_i \wedge \Phi\right)\right) \supset f_k, \quad \left(K \vee \left(\bigwedge f'_i \wedge \Phi'\right)\right) \supset \neg f_k$$

5. At time moment t, for the static rule $G(t,t_1) \supset f$ do

[4] Notice that for each fluent there could be more than one causal relationship. This happens in the case that for fluent f_k, there is more than one integrity constraint. At this step we integrate all this causal relationship into one. For example if $f_1 \wedge f_2$ *causes* f_3 if f_4 and $f_5 \wedge f_6$ *causes* f_3 if f_7 then the static rule $False \supset f_3$ is transformed first into $f_1 \wedge f_2 \wedge f_4 \supset f_3$ and finally into $(f_5 \wedge f_6 \wedge f_7) \vee (f_1 \wedge f_2 \wedge f_4) \supset f_3$.

(a) let $G = G_1 \vee \cdots \vee G_n$
(b) For each j from 1 to n do
- let $G_i = f_1([..]) \wedge \cdots \wedge f_n([..])$
  (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 terminated.
  (iii) else $t_i = t'' - t$.
- let $t_{\min} = \min(t_1, \ldots, t_n)$
- replace G_i with $G_i(t, t + t_{\min})$

Notice that the first four steps are static and executed once at the start. The fifth step is executed each time point at which the static rule must be evaluated. This happens because the formula $G_{f_p}(t, L')$ can be true for different values of t and L'.

Consider the example of the public worker. The above alogrithm works as follows:

The transformation of integrity constrtaints into the CNF form yields:

$$\neg illegal(p, t_1) \vee suspended(p, t_1)$$
$$\neg suspended(p, t_1) \vee take\_salary(p, t_1)$$
$$suspended(p, t_1) \vee \neg good\_employee(p, t_2) \vee take\_bonus(p, t_3)$$
$$good\_employee(p, t_1) \vee \neg take\_bonus(p, t_1)$$
$$suspended(p, t_1) \vee take\_salary(p, t_1)$$

This is step 1 of the algorithm. In step 3 we have

$$\begin{aligned} R = \{&illegal(p, t_1)\ causes\ suspended(p, t_1) \quad \text{if } T, \\ &suspended(p, t_1)\ causes\ \neg take\_salary(p, t_1) \quad \text{if } T, \\ &good\_employee(p, t_1)\ causes\ take\_bonus(p, t_1) \quad \text{if } \neg suspended(p, t_1), \\ &\neg suspended(p, t_1)\ causes\ take\_bonus(p, t_1) \quad \text{if } good\_employee(p, t_1), \\ &\neg good\_employee(p, t_1)\ causes\ \neg take\_bonus(p, t_1) \quad \text{if } T, \\ &\neg suspended(p, t_1)\ causes\ take\_salary(p, t_1) \quad \text{if } T\} \end{aligned}$$

In step 3 we estimate the causal relationships based on the binary relationship I. In step 4 we have

$$\begin{aligned} R = \{&illegal(p, t_1) \supset suspended(p, t_1), \\ &suspended(p, t_1) \supset \neg take\_salary(p, t_1), \\ &\neg suspended(p, t_1) \wedge good\_employee(p, t_1) \supset take\_bonus(p, t_1), \\ &\neg good\_employee(p, t_1) \supset \neg take\_bonus(p, t_1), \\ &\neg suspended(p, t_1) \supset take\_salary(p, t_1)\} \end{aligned}$$

In step 4 we construct for each fluent the fluent formula which makes the fluent true. Notice that perhaps there are many causal relationships which affect the same fluents. We integrate these causal relationships in this step. In step 5 we get

$$\begin{aligned} R = \{&illegal(p, L) \supset suspended(p, L), \\ &suspended(p, L) \supset \neg take\_salary(p, L), \\ &\neg suspended(p, L_1) \wedge good\_employee(p, L_2) \supset take\_bonus(p, L_1 \cup L), \\ &\neg good\_employee(p, L) \supset \neg take\_bonus(p, L), \\ &\neg suspended(p, L) \supset take\_salary(p, L)\} \end{aligned}$$

As we observe the production of static rules is based on the binary relation I. Our idea is that when C_i is false there must be at least one static rule which is executable. For this to happen there must be at least one pair $(f_j, f_w) \in I$ such that $C_i = f_j \vee f_w \vee C_i'$.

In [13,14] we have presented an algorithm for the evaluation of static and dynamic rules for the sequential execution of actions when the effects of the actions refer only to the future.

When a static rule $G(t,L) \supset f(L)$ evaluates the element, $[t,t']$ is added to list L and is removed from the L', where $\neg f(L')$.[5]

Consider the above example with of public worker. We have the following dynamic rules:

$$occur(misdemeanor(p),t) \supset illegal(p,[t,t+5])$$
$$occur(take\_pardon(p),t) \supset \neg illegal(p,[t,\infty])$$
$$occur(bad\_grade(p),t) \supset \neg good\_employee(p,[t,\infty])$$
$$occur(good\_grade(p),t) \supset good\_employee(p,[t,\infty])$$

Assume that the initial situation is

$$S_0 = \{\neg take\_bonus(p,[[0,\infty]]) take\_salary(p,[[0,\infty]]),$$
$$\neg suspended(p,[[0,\infty]]), \neg good\_employee(p,[[0,\infty]]), \neg illegal(p,[[0,\infty]])\}$$

Assume that the following actions occur at the following time points, assuming that the time starts at 0 and time granularity is that of months. Assume the execution of the action

$$occur(misdemeanor(p),2)$$

The new situation is

$$S_1' = \{\neg take\_bonus(p,[[0,\infty]]), take\_salary(p,[[0,\infty]]),$$
$$\neg suspended(p,[[0,\infty]]), \neg good\_employee(p,[[0,\infty]]),$$
$$illegal(p,[[2,7]]), \neg illegal(p,[[7,\infty]])\}$$

The following integrity constraint is not satisfied in S_1':

$$illegal(p,t_1) \supset suspended(p,t1)$$

But the static rule

$$illegal(p,[[2,7]]) \supset suspended(p,[[2,7]])$$

is executable. After the execution of the above static rule the new situation is

$$S_1'' = \{\neg take\_bonus(p,[[0,\infty]]), take\_salary(p,[[0,\infty]]),$$
$$suspended(p,[[2,7]]), \neg suspended(p,[[7,\infty]]), \neg good\_employee(p,[[0,\infty]]),$$
$$illegal(p,[[2,7]]), \neg illegal(p,[[7,\infty]])\}$$

As we observe in that situation the following integrity constraint is not satisfied.

$$suspended(p,t_1) \supset \neg take\_salary(p,t_1)$$

But the static rule

$$suspended(p,[[2,7]]) \supset \neg take\_salary(p,[[2,7]])$$

is executable in S_1''. After the execution the final situation is

$$S_1 = \{\neg take\_bonus(p,[[0,\infty]]),\ \neg take\_salary(p,[[2,7]]), take\_salary(p,[[7,\infty]]),$$
$$suspended(p,[[2,7]]), \neg suspended(p,[[7,\infty]]), \neg good\_employee(p,[[0,\infty]]),$$
$$illegal(p,[[2,7]]), \neg illegal(p,[[7,\infty]])\}$$

We adopt the algorithm for the production of static rules and we address the ramification problem in the case that the effect of the actions could change beliefs about the past.

[5] List L' is the list, which contains the time intervals, in which fluent $\neg f$ is true.

By the production of the static rules we have that

Theorem 3.1. *When a static rule is executable at least one integrity constraint is unsatisfiable.*

Proof. Assume that a static rule

$$G_f(\cdots) \supset f$$

is executable in a situation S. Then the fluent formula G_f must be true. We have that

$$G_f \equiv G_f^1 \vee \cdots G_f^w$$

where

$$G_f^i \equiv \left(\neg \bigwedge f_j(t_j, j = 1, \ldots, m)\right)$$

Each G_f^i is derived from the integrity constraint $C_1 \wedge \cdots \wedge C_n$, such that there is a $C_i \equiv f \vee f_1 \vee \cdots \vee f_m$. Thus when the G_f^i is true it must have all fluents $f_j, j = 1, \ldots, m$ to be false. In order to be executable the above static rule must be f to be false. In that case $C_i \equiv f \vee f_1 \vee \cdots \vee f_m$ is false, thus the integrity constraint $C_1 \wedge \cdots \wedge C_n$ is unsatisfiable. □

In order to have a consistent situation, the set of integrity constraints must be satisfied for some conditions. We study the case that the set of integrity constraints is satisfiable. For example, if there are two integrity constraints of the form

$$f_1 \supset f_2$$
$$f_2 \supset f_3$$
$$f_3 \supset \neg f_1$$
$$\neg f_1 \supset \neg f_2$$
$$\neg f_2 \supset \neg f_3$$
$$\neg f_3 \supset f_1$$

then there is no situation in which the above two constraints are satisfiable because the set

$$\{(\neg f_1, f_2), (\neg f_2, f_3), (\neg f_3, \neg f_1),\ (f_1, \neg f_2), (f_2, \neg f_3), (f_3, f_1)\}$$

is not satisfiable.[6] By the above six constraints we formulate six static rules which will be evaluated one after the other for infinitely. This happens because there is no consistent situation and thus always at least one static rule will be executable.

Theorem 3.2. *Consider a set of static rules G. Then if for a fluent f_1 there is a sequence of*

$$G_{f_2}(\cdots) \supset f_2(\cdots) \quad \text{where } G_{f_2} \equiv f_1 \vee G'_{f_2}$$
$$G_{f_3}(\cdots) \supset f_3(\cdots) \quad \text{where } G_{f_3} \equiv f_2 \vee G'_{f_3}$$
$$\cdots$$
$$G_{f_n}(\cdots) \supset f_n(\cdots) \quad \text{where } G_{f_n} \equiv f_{n-1} \vee G'_{f_n}$$
$$G_{\neg f_1}(\cdots) \supset \neg f_1(\cdots) \quad \text{where } G_{\neg f_1} \equiv f_n \vee G'_{\neg f_1}$$
$$G_{f_{n+1}}(\cdots) \supset f_{n+1}(\cdots) \quad \text{where } G_{f_{n+1}} \equiv \neg f_1 \vee G'_{f_{n+1}}$$
$$\cdots$$
$$G_{f_{n+m}}(\cdots) \supset f_{n+m}(\cdots) \quad \text{where } G_{f_{n+m}} \equiv f_{n+m-1} \vee G'_{f_{n+m}}$$
$$G_{f_1}(\cdots) \supset f_1(\cdots) \quad \text{where } G_{f_1} \equiv f_{n+m} \vee G'_{f_1}$$

then the set of rules is unsatisfiable.

[6] We have that $\neg f_1 \vee f_2 \equiv f_1 \supset f_2$, $\neg f_2 \vee f_3 \equiv f_2 \supset f_3, \ldots$

Proof. Assume that f_1 is true. If we iteratively apply the modus ponen we have

$$
\begin{array}{l}
f_1 \vee G'_{f_2} \supset f_2 \\
f_1 \\
\\
f_2 \\
f_2 \vee G'_{f_3} \supset f_3 \\
\\
f_3 \\
f_3 \vee G'_{f_4} \supset f_4 \\
\\
f_4 \\
\cdots \\
f_n \\
f_n \vee G'_{\neg f_1} \supset \neg f_1 \\
\\
\neg f_1
\end{array}
$$

We conclude that $\neg f_1$ holds. Assume that $\neg f_1$ is true. If we apply iteratively the modus ponen we have

$$
\begin{array}{l}
\neg f_1 \vee G'_{f_{n+1}} \supset f_{n+1} \\
\neg f_1 \\
\\
f_{n+1} \\
f_{n+1} \vee G'_{f_{n+2}} \supset f_{n+2} \\
\\
f_{n+2} \\
f_{n+2} \vee G'_{f_{n+3}} \supset f_{n+3} \\
\\
f_{n+3} \\
\cdots \\
f_{n+\mathrm{m}} \\
f_{n+m} \vee G'_{f_1} \supset f_1 \\
\\
f_1
\end{array}
$$

We conclude that f_1 holds. Thus we always have that $f_1 \wedge \neg f_1$. This is not satisfied. □

For the rest of the paper we assume that the set of the integrity constraints is satisfiable (this means that there is no sequence as described in Theorem 3.2). This will be ensuring no static rule is executable in the initial situation at time point 0 (by that Theorem 3.1).

We can discover if a set of the integrity constraints is satisfiable if we transform each of them in to a CNF form. Then if we have n integrity constraints we have

$$
\begin{array}{l}
C_{11} \wedge \cdots \wedge C_{1n} \\
\cdots \\
C_{1n} \wedge \cdots C_{nn}
\end{array}
$$

Each C_{ik} is a disjoint. Then if the set $C = \{C_{11}, \ldots, C_{1n}, \ldots, C_{1n}, \ldots, C_{nn}\}$ is satisfiable, then the set of integrity constraints is satisfiable. We can use the algorithm of the tree reconstruction in order to see if the set C is satisfiable.

Also in order to be consistent in the set of the static rule R, for each pair $(f, \neg f)$ it holds that $G_f \wedge B_f \equiv FALSE$, when $G_f \supset f, B_f \supset \neg f$. In the other case we have the infinite execution of the rule $G_f \supset f, B_f \supset \neg f$ (one after the other) if $G_f \wedge B_f = TRUE$. This is the second precondition which we assume for the rest of this paper.

Theorem 3.3. *If in a set of static rules R there is a pair* $(f, \neg f)$ *such that* $G_f \wedge B_f \not\equiv FALSE$, *when* $G_f \supset f, B_f \supset \neg f$ *then there is a case that the static rules* $G_f \supset f, B_f \supset \neg f$ *may be executed infinitely.*

3.2. Other previous works

The other most prevalent previous works are those by Reiter [18], Reiter and Pinto [16,17], Kakas [2,3] and the work on the event calculus. 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[7] 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 further extension of the situation calculus based on Reiter's proposal. Kakas [2,3] 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

A happens at T

A initiates F when C

A terminates F when C

L whenever C

A needs C

As we may observe, the third and fourth axioms are dynamic because they evaluate when an action executes, while the last two are static because they evaluate at each time moment. In E, one cannot declare effects that persist over a time span as in the aforementioned example where, if someone did a misdemeanor then s/he is illegal for the subsequent 5 months. In order to achieve this, it is necessary for an action to occur after 5 months. This means that the users must explicitly determine all the indirect and direct effects. Also this assumption creates many new problems. For example, assume the following execution:

$$occur(misdemeanor(p), 3)$$
$$occur(take\_pardon(p), 5)$$
$$occur(misdemeanor(p), 7).$$

The first action has no persistence effects that the fluent *illegal* holds for 5 months. Thus we must determine that an action *end-illegal* will be executed at time point 8. At time point 5 the action *take_pardon*(p) takes place and cancels the effect of the first action. Thus we must cancel the execution of action *end-illegal* because if it is executed at will cancel the effect of the third action and this is wrong. The effect of third action must be cancelled at time point 12. An obvious solution could be to determine preconditions for these "cancelling" actions. In the above example the preconditions could be the fluent *illegal* to hold. This is wrong because in the above example the fluent *illegal* is true at time point 8 but the action *end-illegal* must not be executed. Thus the determination of the preconditions is very complex. Also the number of actions are increased very much because we must define one action of each no persistent effect of each action.

Also, E cannot represent delayed effects, as e.g., if someone does a misdemaenour then s/he becomes illegal 2 months later and remains illegal for the next 5 months. We consider these assumptions rather strong and examine the problem in a strictly more general setting. The language E works satisfactorily only when the world which is described is based on the persistence of fluents (like the circuit).

[7] The problem of determining which predicates and functions are not affected when an action is executed is the *frame problem* [9].

The *event calculus* has been proposed in [10,4]. In the event calculus the time is discrete. We define the following predicates and relations:

$Initiates(a, f, t)$
$Terminates(a, f, t)$
$Initially_P(f)$
$Initially_N(f)$
$t_1 < t_2$
$Happens(a, t)$
$Happens(t_1, a, t_2)$
$HoldsAt(f, t)$
$Clipped(t_1, f, t_2)$
$declipped(t_1, f, t_2)$
$Releases(a, f, t)$.

The first predicate means that the fluent f starts to hold after action a at time t, the second means that the fluent f ceases to hold after action a at time t, the third means that the fluent f holds from time 0, the fourth means that the fluent f does not hold from time 0, the fifth relation means that time point t_1 is before time point t_2, the sixth predicate means that the action a occurs at time point t, the seventh means that the action a starts at time point t_1 and ends at time point t_2, the eighth means that fluent f holds at time point t, the nineth means that fluent f ceases to hold between times t_1 and t_2, the tenth means that fluent f starts to hold between times t_1 and t_2. The last predicate means that fluent f is not subject to inertia after action a at time point t.

The event calculus is very similar to the language E. The most important difference is that the event calculus could encapsulate effects which do not start or terminate in the discrete time point (the predicates $Clipped(t_1,f,t_2)$, $declipped(t_1,f,t_2)$, $Happens(t_1,a,t_2)$).

4. Fluent dependencies

This section describes algorithms for discovering dependencies between fluents. As we have already explained the aim of the binary relation I is to encapsulate the dependencies between fluents and to ensure that when an integrity constraint is not satisfied, then there is at least one static rule which is executable and after the execution the integrity constraint will be satisfied.

Assume that we have two kinds of integrity constraints

(a) $G_f \supset K_f$
(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 second kind, when $\neg G_f$ holds then $\neg K_f$ also holds, whereas this is not necessarily the case for the first. 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.

Algorithm 1 for constructing I

1. 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.
2. For the second kind of constraints, 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 Fig. 3. The integrity constraints specifying the behavior of this system are expressed as the following formulae:

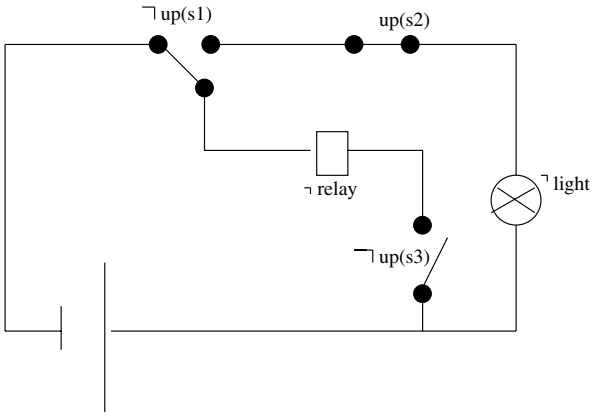

Fig. 3. Theilsher's circuit.

(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 $(up(s1), relay), (up(s3), relay), (relay, (up(s1))), (relay, (up(s3)))$ to be in I and from rule (c) we obtain $(relay, up(s2)) \in I$.

By the second step of Algorithm 1 we have that $(up(s1), light) \in I$, while $(up(s_1), up(s_2)) \notin I$. Assume that the circuit is in the situation that is depicted in Fig. 3. The action $toggle\_switch(s_1)$ has an 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.

Algorithm 2 for constructing I

1. For each $f \in G_f, f' \in K_f$, where $G_f \supset K_f$ is a specified constraint, add the pair $(f, f') \in I$.
2. 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 a 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 as a single rule of the form $G_f \equiv K_f$. For example, consider the circuit in Fig. 4. 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 Fig. 4. 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

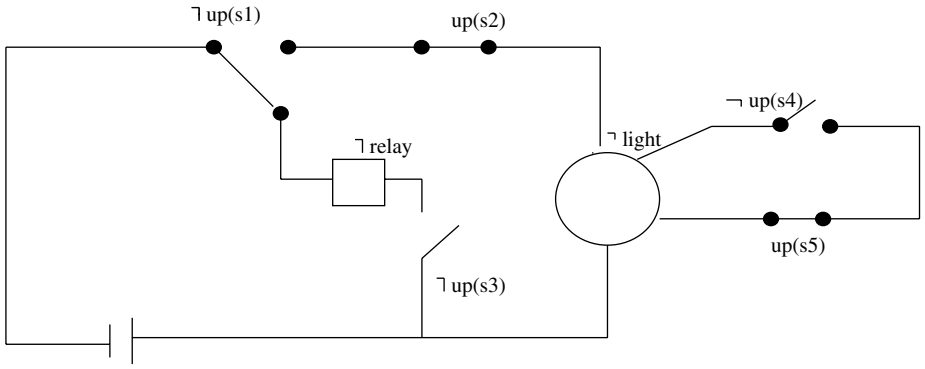

Fig. 4. A more complex circuit.

$\neg up(s1), up(s2), up(s4), up(s5), \neg up(s3), \neg relay, light$, 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. As we observe the circuit of the Fig. 4, consists two smaller circuits. The first consists of the switches s_1, s_2 and the lamp, while the second switches s_4, s_5 and the lamp. The reason is that the lamp light is on when one of the two circuits is closed. This is ensured by the second set of integrity constraints. The first set of integrity constraints ensure that when one circuit is closed then the second must also be closed. This is not reasonable.

Theorem 4.1. *Let $A \supset B$ be a constraint and $C_1 \wedge \cdots \wedge C_n$ its CNF form. Both algorithms produce I in such a way that for each C_i there is at least one pair $(f_1, f_2) \in I$ and $C_i = f_1 \vee f_2 \vee C_i'$.*

Proof. The $A \supset B$ is equivalent with $\neg A \vee B$. Assume that $\Gamma \equiv \neg A$.

Assume that the DNF form of Γ and B are

$$\gamma_1 \vee \cdots \vee \gamma_n$$
$$\beta_1 \vee \cdots \vee \beta_m$$

respectively.

Assume that

$$\gamma_1 = f_{1_{\gamma_1}} \wedge \cdots \wedge f_{n_{\gamma_1}}$$
$$\cdots$$
$$\gamma_n = f_{1_{\gamma_n}} \wedge \cdots \wedge f_{n_{\gamma_n}}$$
$$\beta_1 = f_{1_{\beta_1}} \wedge \cdots \wedge f_{n_{\beta_1}}$$
$$\cdots$$
$$\beta_m = f_{1_{\beta_m}} \wedge \cdots \wedge f_{n_{\beta_m}}.$$

The integrity constraint $A \supset B$ is equivalent to

$$\gamma_1 \vee \cdots \vee \gamma_n \vee \beta_1 \vee \cdots \vee \beta_m.$$

This is equivalent to the following:

$C_1 \wedge \cdots \wedge C_w$, where
(1) $w = n_{\gamma_1} * \cdots * n_{\gamma_n} * n_{\beta_1} * \cdots * n_{\beta_m}$
(2) $C_i = f_{i_1} \vee \cdots \vee f_{i_n} \vee f_{i_{n+1}} \vee \cdots \vee f_{i_{n+m}}$, where
$f_{i_1} \in \gamma_1$,
...
$f_{i_n} \in \gamma_n$
$f_{i_{n+1}} \in \beta_1$
...
$f_{i_{n+m}} \in \beta_m$.

For the above two algorithms (from the step 1) we have the pairs

$\{(f_{i_1}, f_{i_{n+1}}), \ldots, (f_{i_1}, f_{i_{n+m}})\} \subseteq I$
...
$\{(f_{i_n}, f_{i_{n+1}}), \ldots, (f_{i_n}, f_{i_{n+m}})\} \subseteq I$.

Thus for each C_i there is at least one pair $(f,f') \in I$, where f,f' are disjunct of C_i. □

Theorem 4.2. *Let $A \equiv B$ be a constraint and $C_1 \wedge \cdots \wedge C_n$ its CNF form. Algorithm* 1 *generates I in such a way that for each C_i there is at least one pair $(f_1,f_2) \in I$ and $C_i = f_1 \vee f_2 \vee C_i'$. For the first algorithm for each constraint $A \equiv B$ if $C_1 \wedge \cdots \wedge C_n$ is the CNF form then for each C_i there is at least one pair $(f_1,f_2) \in I$.*

Proof. The integrity constraint $A \equiv B$ is equivalent with $(A \supset B) \wedge (B \supset A)$. Assume that

$C_1 \wedge \cdots \wedge C_w$
$C_1' \wedge \cdots \wedge C_w'$

are the CNF forms of $A \supset B$ and $B \supset A$, respectively. Then from the previous theorem we have that for each C_i there is at least one pair $(f,f') \in I$ and $f,f' \in C_i$ and for each C_i' there is at least $(f_1,f_2) \in I$ and $f_1, f_2 \in C_i'$.

The integrity constraint $A \equiv B$ is equivalent to

$C_1 \wedge \cdots \wedge C_w \wedge C_1' \wedge \cdots \wedge C_w'$.

In order to transform the CNF form from the above proposition we must examine if there are some pair (C_i, C_j) at which C_i "subsume" C_j, in which case we must delete the C_j. Thus the CNF form $A \equiv B$ contains a subset of $C_1, \ldots, C_w$. Thus we have proven the theorem. □

Theorem 4.2 does not hold for Algorithm 1 because it eliminates some pairs which Algorithm 1 produces. The problem in these C_i which all the fluents contain and belong to one side of the rule cannot change as direct effects of an action and some of the fluents of the C_i belong to another integrity constraint too. For example assume the following two integrity constraints:

$f \equiv B$
$f \equiv B'$

Fluent f cannot change as a direct effect of an action. Then for all fluents f_i which are in B or in B' and could change their truth value as a direct effect of an action, there is a pair $(f_i,f) \in I$, while $(f, f_i) \notin I$. If an action makes B true, then f must become true, too. After that B' must be true. But fluent f cannot affect the truth value of no fluent in B', since $(f, f_i) \notin I$, so the IC $f \equiv B'$ is violated.

An example for such a situation is the following:

(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) $relay \supset \neg up(s2)$

We identified above the following situation which is problematic for Algorithm 2.

$A \equiv B$ and $C \supset B$ are *ICs* and $A \neq C$

where A, B, C are fluent formulas and B contains fluents which may change their truth value only as an indirect effect of an action. The following results show a weaker condition under which Algorithm 1 is guaranteed to lead to an I which presents consistency.

Theorem 4.3. *For each integrity constraint $A \equiv B$ and if for each fluent A_1 and B_1 which belong to A and B, respectively and could change their truth value only as indirect effects then Algorithm 2 does not produce an inconsistency when there are no C and D, such that $C \supset B_1$ and $D \supset A_1$, $C \neq A$ and $D \neq B$.*

Proof. Assume that integrity constraint $A \equiv B$ and A_1 and B_1 are the fluent formulas which contain the fluents which belong to A and B, respectively and could change its truth value only as indirect effects. Assume that there is no integrity constraint $C \supset B_1$ or $D \supset A_1$ such that $C \neq A$ and $D \neq B$.

Assume that the initial situation satisfied the integrity constraint. Then assume that an update in the database occurs. There are two cases. First this update does not influence any of the fluents which belong to A and B. Then the integrity constraint is satisfiable in the new situation. The second case is this update changes the truth value of some fluents which belong to A or B. This change refers to fluents which belong in $A \backslash A_1$ or $B \backslash B_1$ because the fluents which belong to A_1 and B_1 could change their true value only as indirect effects of an action and no other integrity constraints could affect them (because there is no $C \supset B_1$ or $D \supset A_1$ such that $C \neq A$ and $D \neq B$).

Thus the fluents which change their truth value could affect the other part of the integrity constraint. □

In cases where the condition of Theorem 4.3 is not satisfied, we use Algorithm 1 for generating I. Otherwise we use Algorithm 2.

5. Changing the belief about the past

5.1. Motivation

Recall the example from Section 1. We now extend the example in order to present the problem in case that an action could change the belief about the past. We change the function fluents *position*, *salary* as follows: $position(p, l, t, t_1)$, $salary(p, l, t, t_1)$ where the first means that the public worker p at time point t is in position l for t_1 time points, while the last means that the public worker p at time point t takes salary l for $t_1$time points. Assume that the action *take_pardon* could change the belief about the past.

$$take\_pardon(p, t, t_1) \supset \neg illegal(p, [t_1, \infty])$$

means that the action *take_pardon* occurs at time point t and its effects start to hold for time point t_1 (perhaps $t_1 < t$). We present the problem with some examples.

Example 1. Consider the following execution:

$$occur(misdemeanor(p), 20, 20), \quad occur(take\_pardon(p), 24, 21)$$

The time starts at 0 and has the granularity of months. Consider the initial situation

$$\begin{aligned} S_0 = \{&\neg take\_bonus(p, [0, \infty]), take\_salary(p, [0, \infty]), \\ &\neg suspended(p, [0, \infty]), good\_employee(p, [0, \infty]), \neg illegal(p, [0, \infty]), \\ &position(p, 1, 0, 38), salary(p, 3, 0, 2)\} \end{aligned}$$

At time point 20 the action *misdemeanor* executes and the new situation is

$$S_1 = \{\neg take\_bonus(p,[0,\infty]), \neg take\_salary(p,[20,25]),\\ take\_salary(p,[[0,20],[25,\infty]]),\ suspended(p,[20,25]),\\ \neg suspended(p,[[0,20],[25,\infty]]),\ good\_employee(p,[0,\infty]),\ illegal(p,[20,25]),\\ \neg illegal(p,[[0,20],[25,\infty]]),\ position(p,1,20,58), salary(p,3,20,22)\}$$

Time point 22 is the time when a public worker could take an increase and promotion. This cannot happen because s/he is illegal. At time point 24 the situation is

$$S_1' = \{\neg take\_bonus(p,[0,\infty]), \neg take\_salary(p,[20,25]),\\ take\_salary(p,[[0,20],[25,\infty]]),\ suspended(p,[20,25]),\\ \neg suspended(p,[[0,20],[25,\infty]]), good\_employee(p,[0,\infty]),\ illegal(p,[20,25]),\\ \neg illegal(p,[[0,20],[25,\infty]]),\ position(p,1,24,62), salary(p,3,24,26)\}$$

At this time point the public worker takes pardon which means that for time 21 the public worker ceases to be assumed illegal. This has as an indirect effect that promotion and increase should have been granted at time point 22. We must change the value of fluents for the past time points. The fluent *illegal* changes its value at time point 21. Now the following propositions hold at time point 22:

$$position(p,1,22,60) \wedge \neg illegal(p,[21,\infty])$$
$$salary(p,3,22,24) \wedge \neg illegal(p,[21,\infty]).$$

So the public worker must take promotion and increase at time point 22. Thus the next increase in salary must happen 24 time points after time point 22 (resp. 60 months for promotion).

Example 2. Execution of an action may have as indirect effects to disqualify[8] an action which has already been executed.

Assume that the action *misdemeanor* could refer to the past.

$$occur(misdemeanor(p),26,20)$$

The time starts at 0 and has the granularity of months. Consider the initial situation

$$S_0 = \{\neg take\_bonus(p,[0,\infty]),\ take\_salary(p,[0,\infty]),\\ \neg suspended(p,[0,\infty]),\ good\_employee(p,[0,\infty]),\ \neg illegal(p,[0,\infty]),\\ position(p,1,0,36), salary(p,3,0,0)\}$$

At time point 24 the actions *grant_promotion* and *grant_increase* execute. Now the new situation is

$$S_1 = \{\neg take\_bonus(p,[0,\infty]), take\_salary(p,[0,\infty]),\\ \neg suspended(p,[0,\infty]), good\_employee(p,[0,\infty]), \neg illegal(p,[0,\infty]),\\ position(p,2,0,0), salary(p,4,0,0)\}$$

At time point 26 the action *misdemeanor* executes and has as an effect that the public worker be illegal at time point 20. This means that the situation must change at time point 20 in the following:

$$S_1' = \{\neg take\_bonus(p,[0,\infty]),\ \neg take\_salary(p,[20,25]),\\ take\_salary(p,[[0,20],[25,\infty]]), suspended(p,[20,25]),\\ \neg suspended(p,[[0,20],[25,\infty]]), good\_employee(p,[0,\infty]),\ illegal(p,[20,25]),\\ \neg illegal(p,[[0,20],[25,\infty]]),\ position(p,1,20,56), salary(p,3,20,20)\}$$

[8] This means that if we change the past, perhaps the preconditions of some action which has been executed in the past, become false and thus this action must not have been executed.

Now the action *grant_promotion* and *grant_increase* cannot execute at time point 24. They will be executed at time 25 when the suspension ceases. Now the next promotion and increase must happen at 60 and 24 time points after time point 25.

Example 3. Another problem is the case that an action which could change the belief about the past leads to an infinite loop. Assume the following execution:

$$occur(misdemeanor(p), 26, 20), \quad occur(take\_pardon(p), 27, 22)$$

The time starts at 0 and has the granularity of months. Consider the initial situation

$$\begin{aligned} S_0 = \{&\neg take\_bonus(p, [0, \infty]),\ take\_salary(p, [0, \infty]),\\ &\neg suspended(p, [0, \infty]),\ good\_employee(p, [0, \infty]),\ \neg illegal(p, [0, \infty]),\\ &position(p, 1, 0, 36),\ salary(p, 3, 0, 0)\} \end{aligned}$$

At time point 24 the actions *grant_promotion* and *grant_increase* execute. Now the new situation is

$$\begin{aligned} S_1' = \{&\neg take\_bonus(p, [0, \infty]), take\_salary(p, [0, \infty]),\\ &\neg suspend(p, [0, \infty]), \neg good\_employee(p, [0, \infty]), \neg illegal(p, [0, \infty]),\\ &position(p, 2, 0, 0), salary(p, 4, 0, 0)\} \end{aligned}$$

At time point 26 occurs the action *misdemeanor* which tells us that the public worker did something illegal at time point 20. This means that the situation must change at time point 20 to the following:

$$\begin{aligned} S_1'' = \{&\neg take\_bonus(p, [0, \infty]), \neg take\_salary(p, [20, 25]), take\_salary(p, [[0, 20], [25, \infty]]),\\ &suspended(p, [20, 25]), \neg suspended(p, [[0, 20], [25, \infty]]),\\ &\neg good\_employee(p, [0, \infty]), illegal(p, [20, 25]), \neg illegal(p, [[0, 20], [25, \infty]]),\\ &position(p, 1, 20, 56), salary(p, 3, 20, 20)\} \end{aligned}$$

Now the actions *grant_promotion* and *grant_increase* cannot execute at time point 24. They will be executed at time 25. Now the new promotion and increase must happen 60 and 24 time points after time point 25.

But at time point 27 the action *take_pardon* executes which tells us that the public worker ceases to be illegal at time point 22. Now the new situation is

$$\begin{aligned} S_2 = \{&\neg take\_bonus(p, [0, \infty]), \neg take\_salary(p, [20, 25]),\\ &take\_salary(p, [[0, 20], [25, \infty]]), suspended(p, [20, 25]),\\ &\neg suspended(p, [[0, 20], [25, \infty]]), good\_employee(p, [0, \infty]),\ illegal(p, [20, 22]),\\ &\neg illegal(p, [[0, 20], [22, \infty]]),\ position(p, 1, 20, 56), salary(p, 3, 20, 20)\} \end{aligned}$$

Thus at time point 24 we must execute the actions *grant_promotion* and *take_salary*. This means that we must repeat the execution for time point 22. But at time 26 we will again execute again the action *misdemeanor*. As we observe the second execution of the action *misdemeanor* does not change the truth value of any fluent at time point 20 (because the fluent *illegal* is true at time point 20 as we observe in situation S_1''. Situation S_1'' ceases to hold at time point 22). If we evaluate the dynamic rule $occur(misdemeanor(p), 26, 20) \supset illegal(p, [20, 25])$, the actions *grant_promotion* and *grant_increase* cannot be executed at time point 24. At time point 27 after the execution of the action *take_pardon* ($occur(take\_pardon(p), 27, 22) \supset \neg illegal(p, [22, 25])$) the actions *grant_promotion* and *grant_increase* will be executed at time point 24, and so on. Thus we have an infinite loop.

We must repeat the execution from one time point in the past if and only if an action changes the truth value of some fluents (e.g., from $illegal(p, [20, 22])$ to $\neg illegal(p, [20, \infty])$ as happened in the execution of the action *take_pardon*) and not in the case that only the time intervals at which some fluents are true (e.g., from $illegal(p, [20, 22])$ to $illegal(p, [20, 25])$ as happened in the second execution of the action *misdemeanor*). The above problem arises because the execution of the actions $occur(misdemeanor(p), t_3, t_3')$ and $occur(take\_pardon(p), t_4, t_4')$ satisfied the condition $t_3' < t_4' < t_3 < t_4$. This means that the execution of one action

"intersect" the execution the other action, as shown in Fig. 5. The solution to this problem is to reject the former action whose execution of time is smaller than that of the second action which is executed more recently. *In other words, we break the infinite loop by preferring the most recent information.*

Example 4. Consider the following execution:

$$occur(misdemeanor(p), 26, 23), \quad occur(take\_pardon(p), 27, 22)$$

Assume the same initial situation

$$\begin{aligned} S_0 = \{&\neg take\_bonus(p, [0, \infty]), take\_salary(p, [0, \infty]), \\ &\neg suspended(p, [0, \infty]), good\_employee(p, [0, \infty]), \neg illegal(p, [0, \infty]), \\ &position(p, 1, 0, 36), salary(p, 3, 0, 0)\} \end{aligned}$$

We now have that the actions *grant_increase* and *grant_promotion* execute at time point 24. Now the new situation is S_1' (the same as in the previous example). After the execution of the action *misdemeanor* we repeat the execution for time point 23. Notice that at time point 23 situation S_0 holds (not the S_1 which starts to hold from 24); thus the effects of the *misdemeanor* change situation S_0. The new situation at time point 23 is

$$\begin{aligned} S_2'' = \{&\neg take\_bonus(p, [0, \infty]), \neg take\_salary(p, [23, 28]), \\ &take\_salary(p, [[0, 23], [28, \infty]]), \ suspended(p, [23, 28]), \\ &\neg suspended(p, [[0, 23], [28, \infty]]), \ \neg good\_employee(p, [0, \infty]), \ illegal(p, [23, 28]), \\ &\neg illegal(p, [[0, 23], [28, \infty]]), \ position(p, 1, 23, 59), salary(p, 3, 23, 23)\}. \end{aligned}$$

At time point 24 we do not execute the two actions. After the execution of the action *take_pardon* at time point 27 we repeat the execution for time point 22 (notice that the effect of the action *take_pardon* changes situation S_0 because situation S_2'' starts to hold from time point 23). Now the situation at time point 22 is

$$\begin{aligned} S_2 = \{&\neg take\_bonus(p, [0, \infty]), \ take\_salary(p, [0, \infty]), \\ &\neg suspended(p, [0, \infty]), \ good\_employee(p, [0, \infty]), \\ &\neg illegal(p, [0, \infty]), \ position(p, 1, 22, 58), \ salary(p, 3, 22, 22)\} \end{aligned}$$

and we execute the two actions at time point 24. At time 26 the action *misdemeanor* is executed again and we again have situation S_2'' at time 23. Now the two actions do not execute. At time point 27 the action *take_pardon* executes again and thus we have an infinite loop.

The reasonable way of breaking this loop is to not execute the action misdemeanor for the second time because the action *take_pardon* which has the opposite effect is more recent. As we observe the problem occurs because the actions $occur(misdemeanor(p), t_3, t_3')$ and $occur(take\_pardon(p), t_4, t_4')$ have opposite effects and $t_4' < t_3' < t_3 < t_4$. The last condition means that the action *take_pardon* "contains" the action *misdemeanor*, as shown in Fig. 5B. Notice that again we break the infinite loop by preferring the action which executes more recently.

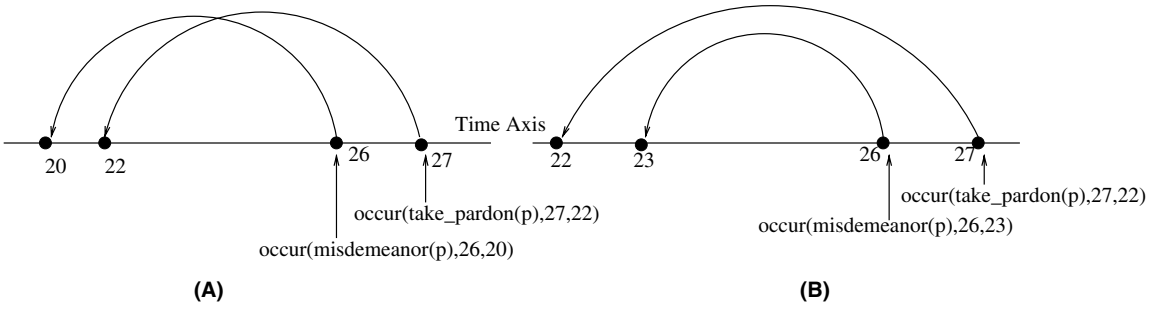

Fig. 5. The scenarios of execution.

In this paper we propose an algorithm for avoiding the infinite loop by rejecting the execution of some actions. As we observed from the above examples the correspondences of Fig. 2 cannot represent the correspondence between situations, actions and time in the case that the effects of the actions refer to the past. This happens because after the execution of an action which changes the past (at time t) we repeat the execution from time point (t). For example consider in the execution of Example 3

$$occur(misdemeanor(p), 26, 20), \quad occur(take\_pardon(p), 27, 22)$$

As we observe the following holds before the execution of the action *misdemeanor*:

$$start(S_0) = 0 \quad end(S_0) = 24$$
$$start(S_1') = 24 \quad end(S_1') = 26$$

After the execution of the action *misdemeanor*

$$start(S_0) = 0 \quad end(S_0) = 20$$
$$start(S_1'') = 20 \quad end(S_1'') = 25$$
$$start(S_1'') = 25 \quad end(S_2) = 27$$

At time point 21 we have two different situations S_0 and S_1''. This also happens at time points 22–26. The correspondences of Fig. 2 ensure that at a time point there is only one situation because the situation axis is linear.

In Example 3 before the execution of the actions *misdemeanor* the action *grant_increase* and *grant_promotion* execute at time point 24, while at time 25 no action takes place. After the execution of the action *misdemeanor* the action *grant_increase* and *grant_promotion*are executed at time point 25, while at time 24 no action takes place. The correspondence of Fig. 2 cannot represent that because it can represent only one history of execution while in Example 3 we have two. The problem is the linear action axis.

We propose to use branching axes for situations and actions, while the time axis remains linear (see Fig. 6). When an action changes the past we start two new linear axes, one for the situations and one for the actions.

At each time point we believe that one linear line of the situation axis is the real evolution of the world. We call this linear line the *actual line*.

When an action changes the past there are three main assumptions that we could adopt:

1. An action may change all the fluents in the past.
2. An action may change only some fluents in the past.
3. The past may change but the effects of these changes start to hold from the current moment.

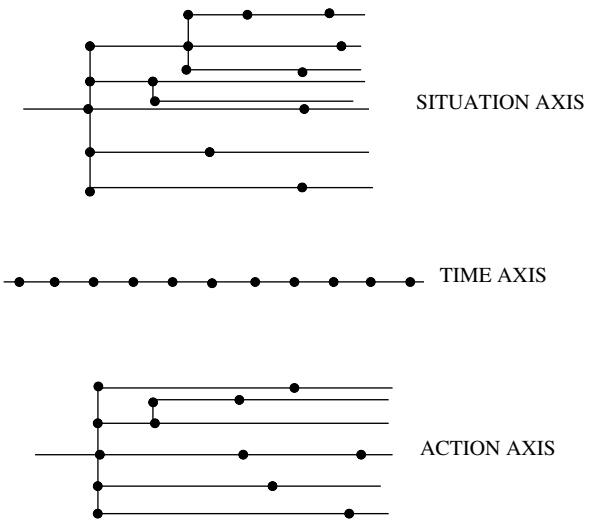

Fig. 6. The correspondence between time–actions–situations.

In the rest of the paper we study the implications of these three assumptions. First we must extend the situation calculus to seem our proposes.

5.2. Further extensions to the situation calculus

In this section we extend the situation calculus in order to solve the ramification problem in case that an action could change the belief about the past.

- We define the as actual line, a sequence of situations which is believed to be the evolution of the world up to the current time point.
- We define the fluent $actual(S,t)$ which shows that situation S is on the *actual line*. The fluent *actual* is defined as follows:
  - $actual(S_0,t_0)$ holds always. S_0 is the initial situation and t_0 is the initial time moment.
  - If $actual(S,t)$ holds and at time point t the situation changes without an action taking place[9] then, if the new situation is S', then $actual(S',t)$ is true.
  - When $occur(a, t, t_1)$ is true and $t_1 >= t$ then, if $actual(S,t_1)$ holds and $S' = do(S,a)$, then $actual(S',t_1)$ is true.
  - When $occur(a, t, t_1)$ is true and $t_1 < t$ then
    - for each situation S s.t. $end(S) = t' < t_1$, if before the execution of action a the predicate $actual(S,t'')$ is true for $t'' \in [start(S), end(S)]$ then after the execution $actual(S,t'')$ still holds.
    - for each situation S s.t. $start(S) = t'_1 < t_1$, if $end(S) = t'_2 >= t_1$ and before the execution of action a the predicate $actual(S,t'')$ is true for $t'' \in [start(S), end(S)]$ then after the execution the following holds: $start(S) = t'_1$ and $end(S) = t_1 - 1$ and $actual(S,t''')$ for all $t''' \in [start(S), t_1 - 1]$.
    - for each situation S s.t. $start(S) = t'_1 > t_1$ and before the execution of action a the predicate $actual(S,t'')$ is true for $t'' \in [start(S), end(S)]$ then after the execution the predicate $actual(S,t''')$ is false for each time point t''. In that case for each time point $t'' > t_1$ the predicate *actual* must be estimated again.[10]
- We categorize the fluents into two sets the F_P and F_S. The first set contains the fluent which cannot change their true value in the past, and the second contains the fluents which could change their true value in the past.
- We define the predicate $ACCEPTANCE(S,t)$ which shows if situation S in time point $t < now$ is acceptance. If S' is the situation for which $actual(S',t)$ is true before the execution of an action which changes the past then S is acceptable if its different from S' does not contain change in the fluents which belong in the F_P. The predicate $ACCEPTANCE$ defined as follows:
  - If an action $occur(a, t_2, t_1)$ has been executed at time point t_2 and change the past at time point t_1 then for each time point t s.t. $t_1 < t <= now$ $ACCEPTANCE(S,t)$ is true if and only if $actual(S',t)$ is true before the execution of action a and S is consistent and $T = FluentHold(S,t) \backslash FluentHold(S',t)$ and $T \cap F_P = \emptyset$.[11]

Now we informally explain the predicates $ACCEPTANCE$ and *actual*. Consider Fig. 7 and assume that the actual line is the top line of the situation axis. Assume that the current time point is 10 and an action a_1 which changes the belief about the past at time point 5 takes place. As we observe we start to construct a new actual line which is the bottom line of the situation axis. In order for the execution of action a_1 to be acceptable the fluents in F_P must not change their truth values. This mean that at time points 5 and 6 situation S_2 (in the new actual line) and situation S_1 (in the previous actual line) must contain the same truth values for all fluents which belong in the F_P. Also at time points 7 and 8 situation S_2 (in the new actual line) and situation S_3 (in the previous actual line) must contain the same truth values for all fluents which belong in the F_P. At time point 9 situation S_4 (in the new actual line) and the situation S_3 (in the previous actual line) must contain the

[9] Because some fluent ceases to hold.
[10] With the execution of the algorithm which we propose in the following sections.
[11] This means that there is no change in the fluents of set F_P.

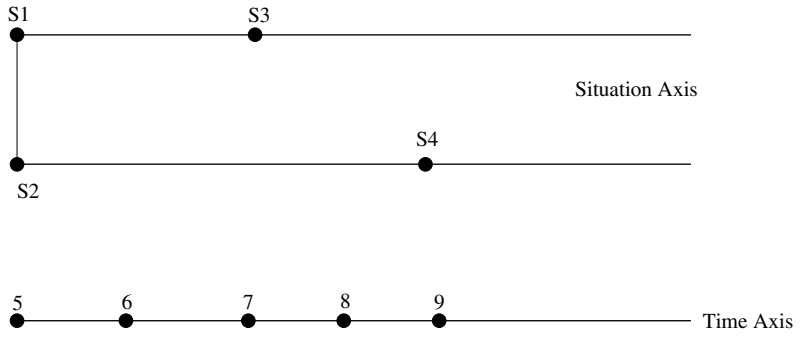

Fig. 7. The effects of the past.

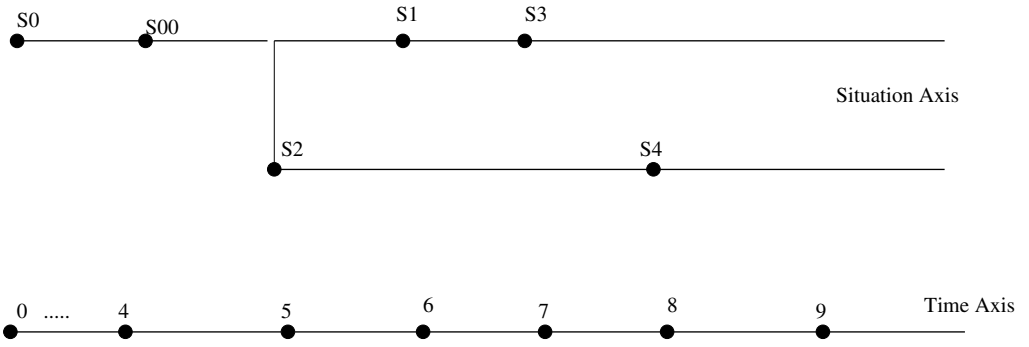

Fig. 8. The branching axes.

same truth values for all fluents which belong in the F_P. In any opposite case the action must be rejected and the actual line remains the top line. The predicate $Acceptance(S_2, 6)$ is true if and only if the following holds:

$$[FluentHold(S_1) \setminus FluentHold(S_2)] \cap F_p \equiv \emptyset$$

Consider Fig. 8. Assume that the current time point is 10 and the actual line is the top line (contains situations S_0, S_{00}, S_1, S_3). As we observe the following hold:

$$start(S_0) = 0 \quad end(S_0) = 4$$
$$start(S_{00}) = 4 \quad end(S_{00}) = 6$$
$$start(S_1) = 6 \quad end(S_3) = 7$$
$$start(S_3) = 7 \quad end(S_3) = 10$$

Assume the execution $occur(a_1, 10, 5)$. As we observe action a_1 changes the belief about the past (*at time point 5*). Thus we must construct the new actual line. The construction of the new actual line concludes the three steps (as seen in the formal definition).

Step 1. For situation S_0 the following holds:

$$start(S_0) = 0 < end(S_0) = 4 < \mathbf{5}$$

This means that situation S_0 is in the new actual line.[12]

Step 2. For situation S_{00} the following holds:

$$start(S_{00}) = 4 < \mathbf{5} < end(S_{00}) = 6$$

[12] The changes happen after the end of situation S_0. Thus the past does not change in the time interval in which situation S_0 holds.

This means that in the time interval that situation S_{00} holds happens the change of the past. The end of situation S_{00} change $end(S_{00}) = 5$. Situation S_{00} is in the new actual line in the time interval [4, 5]. At time point 5 it must be producing a new situation such that it contains the effects of action a_1.

Step 3. For situations S_1 and S_3 the following hold:

$$start(S_1) = 6 > \mathbf{5}$$
$$start(S_3) = 7 > \mathbf{5}$$

The new actual line does not contain situations S_1, S_3 because their start is after the change of the past and thus we must formulate a new situation which contains the effects of action a_1. These new situations are S_2, S_4 such that $start(S_2) = 5$.

5.3. Fluent dependencies

This section describes an algorithm that discovers dependencies between fluents in the case that an action changes the belief about the past. We must distinguish between the future and the past because some fluents cannot change the truth value in the past. More specifically it is possible a fluent can affect a fluent in the future but it cannot affect in the past. In the case of that change in the future the algorithm is the same as we present in Section 4.

In the past only the fluents belonging to F_s can change their truth value. In order to achieve that we change the algorithm as follows:

1. For each $f \in G_f, f' \in K_f$, where $G_f \supset K_f$ is a specified constraint then
   (a) if fluent $f' \notin F_p$ then add the pair $(f,f') \in I$.
2. For each $f \in G_f, f' \in K_f$, where $G_f \equiv K_f$ is a specified constraint do
   (a) If f can change its truth value as the direct effect of an action and $f' \notin F_p$, then add (f,f') in I.
   (b) If f' can change its truth value as a direct effect of an action and $f \notin F_p$ then add (f',f) in I.

5.4. Production of static rules

As we have already seen the binary relation I is defined twice, first is the case that the effects refer to the future, and second is the case that the effects refer to the past. The set of the static rules will be derived from the set of integrity constraints and from the binary relation I. This means that the set of static rules is not the same in the above two cases. Thus we construct two binary relations I_{Future} and I_{Past} and two set of static rules R_{Future} and R_{Past}.

An action may change the future and the past too, but this happens in different executions (e.g., $occur(take\_pardon(p, 24, 21))$ and $occur(take\_pardon(p, 24, 25))$. If an action changes the past at time point $t' < now$ then in the time interval $[t', now)$ we evaluate the set of static rules R_{Past}. At each time $t \geqslant now$ we evaluate the set of static rules R_{Future}.

Notice that in this paper we do not consider the combination of the concurrent execution of actions, some of the change refers to the past and some to the future. Such a combination poses difficult problem which will be addressed in a future work.

5.5. Case 1: Change in the past may affect all the fluents

In this case

$$F_P = \emptyset$$
$$F_S = F$$
$$Acceptance(S, t) \equiv TRUE$$

where F is the set of fluents. All fluents may change their true value in the past, the predicate *Acceptance* is always true.[13] Also there is one set of static rules which is evaluated regardless of whether the effects of an action change the future or the past. Thus the set of static rules is the same as we have presented in the previous section. We propose the following algorithm which returns a consistent situation at each time point (the ramification problem).

Algorithm 1 for constructing a consistent situation

1. At each time point at which some action which changes the truth value of some fluents at time point t' in the past is executed, do: evaluate the dynamic rule which refers to this action, evaluate the static rules (until no change occurs) and set $E = \{S_1, t'\}$, where S_1 is the new situation at the smallest time point t' to which the effects of the action are referred.
2. Repeat the execution for the smaller time point t' above. Every time an action is executed, add it to set E_1. At every time t'' that change situation into a new situation S_2 do:
   - If the tuple (S_2, t'') is already in E then call the rejection algorithm and go on without the action which it rejected.
   - Else go on until no change occurs
3. At each time point at which no action is executed which change the past do: evaluate the dynamic rule (if some action executed). Evaluate the static rules until no change occurs.

Notice that set E contains the situations which are produced as effects of the change of the past. This helps us to understand when there is an infinite loop, which happens when we formulate the same situation at the same time point. In that case three are two same pairs (S, t) in set E. Set E_1 contains the action which takes place after the execution of the action which changes the past. Thus when there is an infinite loop we must reject one action which is in set E_1. This is achieved by the following algorithm:

The algorithm for the rejecting actions

1. If in the set of actions E_1 there are two actions a_1, a_2 such that $a_1 \supset f_i([t_1, t_1'])$ and $a_2 \supset \neg f_i([t_2, t_2'])$ and $occur(a_1, t_3, t_1)$ and $occur(a_2, t_4, t_2)$ and $t_2 < t_1 < t_3 < t_4$ *then* reject the a_1.
2. Else reject the action which was executed more recently in the current time moment.

Example 1 (*continued from Section 5.1*). We have the execution

$$occur(misdemeanor(p), 20, 20), \quad occur(take\_pardon(p), 24, 21)$$

The time starts at 0 and has the granularity of months. Consider the initial situation:

$$\begin{aligned} S_0 = \{&\neg take\_bonus(p, [0, \infty]), take\_salary(p, [0, \infty]), \\ &\neg suspended(p, [0, \infty]), good\_employee(p, [0, \infty]),\ \neg illegal(p, [0, \infty]), \\ &position(p, 1, 0, 38), salary(p, 3, 0, 2)\} \end{aligned}$$

At time point 20 the action *misdemeanor* executes and the new situation becomes

$$\begin{aligned} S_1 = \{&\neg take\_bonus(p, [0, \infty]), \neg take\_salary(p, [20, 25]), \\ &take\_salary(p, [[0, 20], [25, \infty]]),\ suspended(p, [20, 25]), \\ &\neg suspended(p, [[0, 20], [25, \infty]]),\ \neg good\_employee(p, [0, \infty]),\ illegal(p, [20, 25]), \\ &\neg illegal(p, [[0, 20], [25, \infty]]),\ position(p, 1, 20, 58), salary(p, 3, 20, 22)\} \end{aligned}$$

Time point 22 is the time that a public worker could take increase and promotion. This cannot happen because s/he is illegal. At time point 24 the situation is

[13] Notice that the predicate *Acceptance* ensures that the fluents which belong to the set of F_P do not change their truth value in the past.

$$S_1' = \{\neg take\_bonus(p,[0,\infty]),\ \neg take\_salary(p,[20,25]),\ take\_salary(p,[[0,20],[25,\infty]]),$$
$$suspended(p,[20,25]),\neg suspended(p,[[0,20],[25,\infty]]),$$
$$\neg good\_employee(p,[0,\infty]),\ illegal(p,[20,25]),\neg illegal(p,[[0,20],[25,\infty]]),$$
$$position(p,1,24,62),salary(p,3,24,26)\}$$

Suppose that at time point 24 the public worker takes pardon which means that for time 21 the public worker stops to be assumed $illegal(occur(take\_pardon(p),24,21))$. Now we repeat the execution for time point 21. The new situation at time point 21 is

$$S = \{\neg take\_bonus(p,[0,\infty]),\ \neg take\_salary(p,[20,25]),take\_salary(p,[[0,20],[25,\infty]]),$$
$$suspended(p,[20,25]),\neg suspended(p,[[0,20],[25,\infty]]),$$
$$\neg good\_employee(p,[0,\infty]),\ illegal(p,[20,21]),\neg illegal(p,[[0,20],[21,\infty]]),$$
$$position(p,1,21,59),salary(p,3,21,23)\}$$

At time point 22 the following static rule will be evaluated

$$position(p,1,22,60)\wedge\neg suspended(p,[21,\infty])\supset position(p,2,22,0)$$
$$salary(p,3,22,24)\wedge\neg suspended(p,[21,\infty])\supset salary(p,4,22,0).$$

Now the new situation is

$$S_1' = \{\neg take\_bonus(p,[0,\infty]),\ \neg take\_salary(p,[20,25]),$$
$$take\_salary(p,[[0,20],[25,\infty]]),\ suspended(p,[20,25]),$$
$$\neg suspended(p,[[0,20],[25,\infty]]),\ \neg good\_employee(p,[0,\infty]),\ illegal(p,[20,21]),$$
$$\neg illegal(p,[[0,20],[21,\infty]]),\ position(p,2,22,0),salary(p,4,22,0)\}$$

At time point 24 the action $take\_pardon(occur(take\_pardon(p),24,21))$ executes again but it does not change the past because the public worker is not illegal at time point 21. This means that the algorithm does not repeat the execution from time point 21 but it goes on at time point 25.

Example 3 (*continued from Section 5.1*).

$$occur(misdemeanor(p),26,20),\quad occur(take\_pardon(p),27,22)$$

The time starts at 0 and time has the granularity of months. Consider the initial situation

$$S_0 = \{\neg take\_bonus(p,[0,\infty]),\ take\_salary(p,[0,\infty]),$$
$$\neg suspended(p,[0,\infty]),good\_employee(p,[0,\infty]),\ \neg illegal(p,[0,\infty]),$$
$$position(p,1,0,36),salary(p,3,0,0)\}$$

At time point 24 the actions *grant_promotion* and *grant_increase* will be evaluated. Now the new situation is S_1'. At time point 26 occurs the action *misdemeanor* which tells us that the public worker has done something illegal at time point 20. This means that the situation must change at time point 20 in situation S_1''. Now we add (S_1'',20) in to set E and the action *misdemeanor* in to set E_1.

$$E = \{(S_1'',20)\}E_1 = \{misdemeanor\}$$

But at time point 27 is executed the action *take_pardon* which tells us that the public worker stopped being illegal at time point 22. Now the new situation is S_2.

$$S_2 = \{\neg take\_bonus(p,[0,\infty]),\ \neg take\_salary(p,[20,25]),$$
$$take\_salary(p,[[0,20],[25,\infty]]),\ suspended(p,[20,25]),$$
$$\neg suspended(p,[[0,20],[25,\infty]]),\ good\_employee(p,[0,\infty]),\ illegal(p,[20,22]),$$
$$\neg illegal(p,[[0,20],[22,\infty]]),\ position(p,1,20,56),salary(p,3,20,20)\}$$

We add the pair $(S_2,22)$ in to set E and the action *take_pardon* in to set E_1.

$$E = \{(S_1'', 20), (S_2, 22)\} \quad E_1 = \{misdemeanor, take\_pardon\}$$

We repeat the execution from time point 22 and at time point 24 the actions *grant_promotion* and *take_salary* must be executed. But at time point 26 we must again execute the action *misdemeanor*. We know that $occur(misdemeanor(p), 26, 20) \supset illegal(p, [20, 25])$. But at time point 20 situation S_1'' holds and in that situation the fluent *illegal* is true at time point 20, because $illegal(p, [20, 25]) \in S_1''$. This means that the action *misdemeanor* does not change the belief about the past and thus the algorithm does not evaluate the above dynamic rule.

Example 4 (*continued from Section 5.1*).

$$occur(misdemeanor(p), 26, 23), \quad occur(take\_pardon(p), 27, 22)$$

Consider the same initial situation as in the previous example.

$$\begin{aligned} S_0 = \{&\neg take\_bonus(p, [0, \infty]),\ take\_salary(p, [0, \infty]),\\ &\neg suspended(p, [0, \infty]), good\_employee(p, [0, \infty]),\ \neg illegal(p, [0, \infty]),\\ &position(p, 1, 0, 36), salary(p, 3, 0, 0)\} \end{aligned}$$

We now have that the actions *grant_increase* and *grant_promotion* are executed at time point 24. The new situation is S_1'.

$$\begin{aligned} S_1' = \{&\neg take\_bonus(p, [0, \infty]),\ take\_salary(p, [0, \infty]),\\ &\neg suspended(p, [0, \infty]),\ \neg good\_employee(p, [0, \infty]),\\ &\neg illegal(p, [0, \infty]),\ position(p, 2, 22, 0), salary(p, 4, 22, 0)\} \end{aligned}$$

After the execution of the action *misdemeanor* we repeat the execution for time point 23. The new situation at time point 23 is S_2''

$$\begin{aligned} S_2'' = \{&\neg take\_bonus(p, [0, \infty]),\ \neg take\_salary(p, [23, 28]),\\ &take\_salary(p, [[0, 23], [28, \infty]]),\ suspended(p, [23, 28]),\\ &\neg suspended(p, [[0, 23], [28, \infty]]),\ \neg good\_employee(p, [0, \infty]),\ illegal(p, [23, 28]),\\ &\neg illegal(p, [[0, 23], [28, \infty]]),\ position(p, 1, 23, 59), salary(p, 3, 23, 23)\} \end{aligned}$$

We add $(S_2'', 23)$ into E and the *misdemeanor* into E_1.

$$E = \{(S_2'', 23)\}\ E_1 = \{misdemeanor\}$$

At time point 24 we do not execute the two natural actions. After the execution of the action *take_pardon* at time point 27 we repeat the execution for time point 22. Now the situation at time point 22 is S_2.

$$\begin{aligned} S_2 = \{&\neg take\_bonus(p, [0, \infty]),\ \neg take\_salary(p, [20, 25]),\\ &take\_salary(p, [[0, 20], [25, \infty]]),\ suspended(p, [20, 25]),\\ &\neg suspended(p, [[0, 20], [25, \infty]]),\ good\_employee(p, [0, \infty]),\ illegal(p, [20, 22]),\\ &\neg illegal(p, [[0, 20], [22, \infty]]),\ position(p, 1, 20, 56),\ salary(p, 3, 20, 20)\} \end{aligned}$$

We now add the $(take\_pardon, 22)$ into E and the *take_pardon* into E_1.

$$E = \{(S_2'', 23), (S_2, 22)\} \quad E_1 = \{misdemeanor, take\_pardon\}$$

We repeat the execution from time point 22 and at time point 24 must execute the actions *grant_promotion* and *take_salary*. But at time point 26 we again execute the action *misdemeanor*. Now the new situation at time point 23 is the S_2''. But $(S_2'', 23) \in E$. Thus we must call the rejection algorithm. This algorithm found that $E_1 = \{misdemeanor, take\_pardon\}$ and

$$\begin{aligned} &occur(misdemeanor(p), 26, 23)) \supset illegal(p, [23, 28])\\ &occur(take\_pardon(p), 27, 22) \supset \neg illegal(p, [22, \infty]) \end{aligned}$$

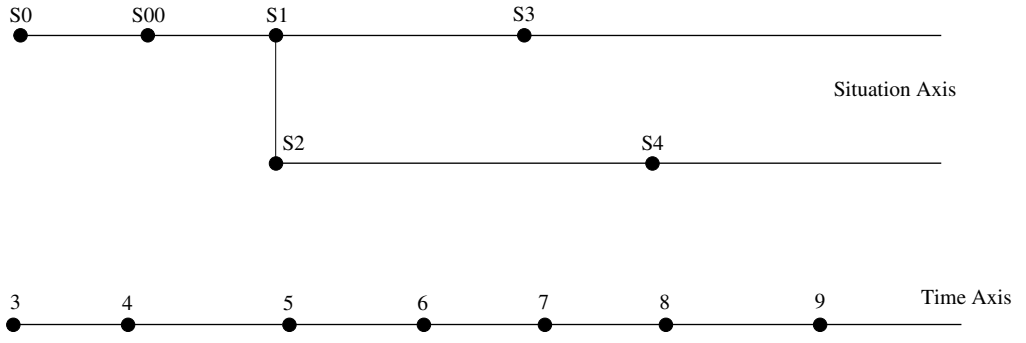

Fig. 9. The effects in the past.

We observe that $22 < 23 < 26 < 27$, which means that the algorithm rejects the action *misdemeanor*(p). This means that we do not execute the action *misdemeanor*(p) for a second time, so the infinite loop is broken.

We now present two formal results for the above algorithm.

Theorem 5.1. *The above algorithm always returns a consistent situation in the case that some action could change all the beliefs about the past.*

Proof. Let t be the earliest time at which a change occurs. We must ensure that there is a sequence of consistent situations from t until the current time point.

Assume that an action a takes place in time point t_1 and changes the belief about the past in time point $t < t_1$. In that case we can assume without loss of generality that we take the sequence of situations from t_0 (the start of time) until t, and action a will be executed at time point t and its effects start to hold from this time point.[14] Now it is suffices to prove that all situations from time point t until the current point are consistent (Fig. 9).

As we observe the algorithm at each step (steps 1–3) evaluates the static rules until no change occurs. The three steps (together) cover all time points. So it is enough to prove that if an integrity constraint is not satisfied at a time point then there is a static rule which is executable and after its execution the integrity constraint will be satisfied.

Suppose that at time point t' $(t < t' \leqslant t_1)$ the algorithm returns a situation S. Assume that integrity constraint Law_j is not satisfied in situation S, and let its CNF be $C_1 \wedge \cdots \wedge C_n$. Then one of the $C_1, \ldots, C_n$ is false. Assume that $C_i = f_1 \vee \cdots \vee f_m$ is false. Then all fluents $f_j, j = 1, \ldots, m$ are false. Assume that f_k and f_p are two of these for which $(f_k, f_p) \in I$ (by Theorems 4.1 and 4.2 there is at least one such pair). Then for the algorithm of the production of static rules (steps 3 and 4) we have that for f_p it must be the case that: $G_{f_p}(t', L) = G'(..) \vee (\neg \bigwedge f_j(t_j, j = 1, \ldots, m, j \neq p)$. If all fluents f_j, $j = 1, \ldots, m$ are false then $(\neg \bigwedge f_j$, $j = 1, \ldots, m, j \neq p)$ is true. Thus $G_{f_p}(t')$ is true. This means that the static rule $G_{f_p}(t', L) \supset f_p(L)$ must be evaluated and thus, f_p is true. A contradiction. □

We must prove that the algorithm avoids the infinite loops.

Theorem 5.2. The above algorithm terminates always.

Proof. In order to prove that the algorithm does not go into infinite loops we must prove that:

1. First the execution of static rules returns a consistent situation in a finite number of steps.
2. Second the repeat of the execution of actions (from a time point in the past until now) terminates. □

[14] This means that the predicate *occur*(a, t_1, t) is "equivalent" to the predicate *occur*(a, t, t). For example consider Fig. 9 and assume that the current time point is 10 and the actual line is the top. We execute the action *occur*(a_1, 10, 5). Then the new actual line contains situations S_0, S_{00} until time point 5. This means that in the time before 5 the old and new actual line are the same. At time point 5 the effects of actions a_1 start to hold. Between time points 5 and 10 we must estimate the new actual line (S_2, S_4). This is equivalent to the execution *occur*(a_1, 5, 5) in situation S_{00}, because the actual line will be the same.

Proof 1. Assume that at time unit t the algorithm does not terminate. Then, there must be an infinite loop. Assume that S_t^0 is the initial situation at time t. Then, there is a non-terminating sequence $S_t^0, S_t^1, \ldots, S_t^k, \ldots$[15]

In this proof, the term "situation" means the truth value of the fluents. Thus the transition from one situation to the next happens only when a fluent changes its truth value.[16] Notice that because a static rule is evaluated only when the corresponding fluent is false, it is not possible that a static rule $G(t,t') \supset f(t')$ is evaluated when fluent f is true in point t. The static rule will be evaluated when f becomes false. Thus the transition from one situation to the next occurs only when fluent f changes from f to $\neg f$.

If F is the number of fluents then there are 2^F different situations Thus in the above sequence, there are two identical situations because of the infinite loop. Without a loss of generality we assume $S_t^l = S_t^k$, $l < k$.

Thus in the sequence $S_t^l, \ldots, S_t^k$ there is at least one fluent f which changes from f to $\neg f$ and eventually becomes f again.

Assume that f' is one such fluent, and consider the static rules associated with it.

$$G(t,t') \supset f'(L')$$
$$B(t,t'') \supset \neg f'(L'')$$
$$L' \cap L'' \neq \emptyset$$

First suppose that f' holds. Then we must evaluate the rule $B(t,t'') \supset \neg f'$ and afterwards the $G(t,t') \supset f'$. Then one of the following holds:

- At time t the proposition $G \wedge B$ must be true. But the conditions G and B are mutually exclusive. A contradiction.
- There is a sequence of static rules as Theorem 3.2 describes. In that case the integrity constraints are unsatisfiable. A contraction (the initial situation satisfies all integrity constraints). □

Proof 2. Assume that the algorithm executes an infinite sequence of actions. Let this happen after the execution of action a at time point t_1 which changes the past at time point t. This mean that there is a sequence of consistent situations $(S_t^0, t), (S_t^1, t^1), \ldots, (S_t^k, t^k), \ldots$. Step 1 of the algorithm adds the pair (S_t^0, t) to the E and the action a to E_1. Each time t' that the situation changes to the new situation S', step 2a add the pair (S', t') to E. Also, when an action take places, step 2 adds it to E_1. Thus all the above sequence of pairs is in E.

All the actions executed in the time interval $[t, t_1]$ suppose that the earliest reference in the past is t^0. Then $t^0 < t^1, \ldots, t^n < t_1$. An infinite loop can only happen if some action is executed in the same situation infinite times, since the number of action is finite. This means that there is $(S_l, t^l) = (S_m, t^m)$. In that case the second time that the tuple (S_l, t^l) is produced the algorithm will reject the execution because the same tuple exist twice in set E. In that case the algorithm rejects the execution of an action (from the set E_1) and repeats the execution without the specific action. This process is repeated until no infinite loop occurs. □

5.6. Case 2: Only some fluent could change in the past

This case is more complex because we must distinguish between fluents that may change in the past, and fluents that change in the future only. For example, we might specify

$$F_P = \{position, take\_bonus\}$$
$$F_S = \{illegal, suspended, good\_employee, take\_salary, salary\}$$

We must product two different sets of fluent dependencies, one for each category of fluents. In our example, the first is

[15] The transition from one situation to the next happens after the evolution of one or more static rules.

[16] This mean that each $S_t^i = FluentHold(S_m, t)$, for a temporal situation S_m.

$$I_{\text{Future}} = \{(illegal, suspended), (suspended, \neg take\_salary), (\neg suspended, \neg take\_bonus), (good\_employee, \neg take\_bonus), (\neg good\_employee, take\_salary), (\neg suspended, take\_salary)\}$$

The second is referred in the past

$$I_{\text{Past}} = \{(illegal, suspended),\ (suspended, \neg take\_salary), (\neg good\_employee, take\_salary),\ (\neg suspended, take\_salary)\}$$

Now the algorithm of product static rules return two sets, one for the future effects and one for the past effects.[17]

$$R_{\text{Future}} = \{illegal(p,L) \supset suspended(p,L), suspended(p,L) \supset \neg take\_salary(p,L), \neg suspended(p,L_1) \wedge good\_employee(p,L_2) \supset take\_bonus(p, L_1 \cap L_2)), \neg good\_employee(p,L) \supset \neg take\_bonus(p,L), \neg suspended(p,L) \supset take\_salary(p,L)\}$$

and

$$R_{\text{Past}} = \{illegal(p,L) \supset suspended(p,L), suspended(p,L) \supset \neg take\_salary(p,L), r\neg suspended(p,L) \supset take\_salary(p,L)\}$$

$$I_{\text{Past}} \subseteq I_{\text{Future}}$$

$$R_{\text{Past}} \subseteq R_{\text{Future}}$$

always hold because all fluents can change their truth value in the future. Thus all fluents "participate" in the producing sets I_{Future} and R_{Future} (set of static rules). Only some fluents can change their truth value to the past. Only these fluents can "participate" in the producing sets I_{Past} and R_{Past}. Thus for each pair $(f,f') \in I_{\text{Past}}$ we have that $(f,f') \in I_{\text{Future}}$ while for each pair $(f_1,f_2) \in I_{\text{Future}}$ such that f_1 or f_2 cannot change its truth value to the past we have that $(f_1,f_2) \notin I_{\text{Past}}$. Steps 3 and 4 of the algorithm for producing the static rules and the number of static rules depend on the set of fluent dependencies I. For each pair in I there is a corresponding static rule. Thus the set R_{Future} is superset of R_{Past}.

We must extend the algorithm which is present in the previous section in order to solve the ramification problem in this case. The algorithm returns a sequence of consistent new situations.

Algorithm 2 for constructing a consistent situation

1. At each time point at which some action is executed and changes the truth value of some fluents in the past do:
   (a) Execute the dynamic rule which refers to this action, execute the static rules which belongs to set R_{Past}, set $E = \{(S_1, t')\}$, where S_1 is the new situation at the smallest time point t' to which the effects of the action refer. Then:
      (i) If $Acceptance(S_1, t')$ holds then go on,
      (ii) Else reject the execution of the action.
   (b) Repeat the execution from the earliest time point to which the effects of the action refer. Every time that an action a must be executed.
      • if it has direct effect to change a fluent which belong in the set F_P then reject the action
      • else add the action to set E_1.

   Every time t'' that change situation into a new situation S_2 do
      (i) If $Acceptance(S_2, t'')$ holds then add (S_2, t'') to E else call the algorithm of rejection of an action.

[17] The algorithm for deriving static rules is the algorithm which we present in Section 4. This algorithm depends on set I. Given different inputs (e.g., $I_{\text{Past}}, I_{\text{Future}}$), it will return different sets of static rules.

(ii) Else if the pair (S_2, t'') there is already in E then call the algorithm for rejecting actions and go on without the action which is rejected.
(iii) Else go on until no change occurs

2. At each time point at which some action which does not change the past executes do: execute the dynamic rule which refers to this action, execute the static rules which belongs to set R_{Future} until no change occurs.
3. At each time point $t \geqslant now$ execute the static rules of R_{Future} until no change occur.

The algorithm for the rejecting actions was presented in the previous section.

Example 3 (*continued from Section 5.1*).

$$occur(misdemeanor(p), 26, 20), \quad occur(take\_pardon(p), 27, 22)$$

The time starts at 0 and has the granularity of months. Assume the initial situation

$$\begin{aligned} S_0 = \{&\neg take\_bonus(p, [0, \infty]),\ take\_salary(p, [0, \infty]),\\ &\neg suspended(p, [0, \infty]), good\_employee(p, [0, \infty]),\ \neg illegal(p, [0, \infty]),\\ &position(p, 1, 0, 36), salary(p, 3, 0, 0)\} \end{aligned}$$

At time point 24 the natural actions *grant_promotion* and *grant_increase* are executed. Now the new situation is

$$\begin{aligned} S_1' = \{&\neg take\_bonus(p, [0, \infty]),\ take\_salary(p, [0, \infty]),\\ &\neg suspended(p, [0, \infty]), \neg good\_employee(p, [0, \infty]),\ \neg illegal(p, [0, \infty]),\\ &position(p, 2, 0, 0), salary(p, 4, 0, 0)\} \end{aligned}$$

At time point 26 occurs the action *misdemeanor* which tells us that the public worker did something illegal at time point 20. So the situation must change at time point 20 as follows:

$$\begin{aligned} S_1'' = \{&\neg take\_bonus(p, [0, \infty]),\ \neg take\_salary(p, [20, 25]),\ take\_salary(p, [[0, 20], [25, \infty]]),\\ &suspended(p, [20, 25]), \neg suspended(p, [[0, 20], [25, \infty]]),\\ &\neg good\_employee(p, [0, \infty]),\ illegal(p, [20, 25]), \neg illegal(p, [[0, 20], [25, \infty]]),\\ &position(p, 1, 20, 56), salary(p, 3, 20, 20)\} \end{aligned}$$

Now the actions *grant_promotion* and *grant_increase* cannot be executed at time point 24. They will be executed at time point 25 which referred to in the past. But if the action *grant_promotion* executed at time point 25 despite at 24 the fluent $position \in F_P$ changes value in the past (the change happens at time point 24 at which before the execution of the action *misdemeanor* the public worker has taken promotion and thus was one position greater while after the execution does not take promotion and remains in same position). In order to avoid this we must reject the execution of action *misdemeanor*. This must be because the action *misdemeanor* disqualifies the action *grant_promotion* which change the truth value of a fluent which belong in the set F_P.

The above algorithm does that because after the execution of action *misdemeanor* at time 24 situation is the S_1''. Before the execution of action *misdemeanor* at time 24 the situation is S_1' and $S_1'' \setminus S_1' = \{position\} \subseteq F_P$. This mean that the predicate $ACCEPTANCE(S_1'', 24) = FALSE$. Thus the algorithm reject the execution of action *occur*(*misdemeanor*(*p*),26,20).

Example 2 (*continued from Section 5.1*). Consider now the following execution:

$$occur(take\_pardon(p), 26, 20)$$

The time start at 0 has the granularity of months. Assume the initial situation

$$\begin{aligned} S_0 = \{&\neg take\_bonus(p, [0, \infty]),\ \neg take\_salary(p, [0, 21]), take\_salary(p, [21, \infty])\\ &suspended(p, [0, 21]), \neg suspended(p, [21, \infty]),\ \neg good\_employee(p, [0, \infty]), illegal(p, [0, 26]),\\ &\neg illegal(p, [26, \infty]), position(p, 1, 0, 36), salary(p, 3, 0, 0)\} \end{aligned}$$

At time point 21 the static rules which correspond to the fluents *suspended*, $\neg take\_salary$ will be evaluated and the new situation is

$$S'_0 = \{\neg take\_bonus(p,[0,\infty]),\ \neg take\_salary(p,[21,26]), take\_salary(p,[26,\infty]),\\ suspended(p,[21,26]), \neg suspended(p,[26,\infty]),\ \neg good\_employee(p,[0,\infty]),\ illegal(p,[0,26]),\\ \neg illegal(p,[26,\infty]), position(p,1,0,36), salary(p,3,0,0)\}$$

At time point 24 the actions *grant_promotion* and *grant_increase* cannot be executed. At time point 26 the action *take_pardon* is executed and changes the past at time point 20 (at situation S_0). The new situation at time point 22 is

$$S'_1 = \{\neg take\_bonus(p,[0,\infty]),\ take\_salary(p,[21,\infty]),\\ \neg suspended(p,[21,\infty]), \neg good\_employee(p,[0,\infty]),\ \neg illegal(p,[20,\infty]),\\ position(p,1,22,58), salary(p,3,22,22)\}$$

Now at time point 24 the preconditions of actions *grant_promotion* and *grant_increase* hold, but the action *grant_promotion* cannot be executed because it changes the truth value of the fluent *position* which cannot change is value in the past. Thus only the action *grant_increase* will be executed. Now the new situation at time point 24 is

$$S_1 = \{\neg take\_bonus(p,[0,\infty]), take\_salary(p,[21,\infty]),\\ \neg suspended(p,[21,\infty]), \neg good\_employee(p,[0,\infty]),\ \neg illegal(p,[20,\infty]),\\ position(p,1,24,60), salary(p,4,24,0)\}$$

Assume the same execution, but the initial situation

$$S_0 = \{\neg take\_bonus(p,[0,\infty]),\ \neg take\_salary(p,[0,21]), take\_salary(p,[21,\infty]),\\ suspended(p,[0,21]), \neg suspended(p,[21,\infty]),\ good\_employee(p,[0,\infty]), illegal(p,[0,26]),\\ \neg illegal(p,[26,\infty]), position(p,1,0,36), salary(p,3,0,0)\}$$

The difference is that now the fluent *good_employee* holds. At time point 21 the static rules which corresponded to the fluents *suspended*, $\neg take\_salary$ will be evaluated and the new situation is

$$S'_0 = \{\neg take\_bonus(p,[0,\infty]),\ \neg take\_salary(p,[21,26]), take\_salary(p,[26,\infty]),\\ suspended(p,[21,26]), \neg suspended(p,[26,\infty]),\ good\_employee(p,[0,\infty]),\ illegal(p,[0,26]),\\ \neg illegal(p,[26,\infty]), position(p,1,0,36), salary(p,3,0,0)\}$$

At time point 24 the actions *grant_promotion* and *grant_increase* cannot execute. At time point 26 the action *take_pardon* executes and changes the past at time point 20. The new situation at time point 22 is

$$S'_1 = \{\neg take\_bonus(p,[0,\infty]),\ take\_salary(p,[21,\infty]),\\ \neg suspended(p,[21,\infty]), good\_employee(p,[0,\infty]),\ \neg illegal(p,[20,\infty]),\\ position(p,1,22,58), salary(p,3,22,22)\}$$

After the execution of the static rules R_{Past} the situation becomes:

$$S'''_1 = \{\neg take\_bonus(p,[0,\infty]),\ take\_salary(p,[22,\infty]),\\ \neg suspended(p,[22,\infty]),\ good\_employee(p,[0,\infty]),\ \neg illegal(p,[22,\infty]),\\ position(p,1,22,58),\ salary(p,3,22,22)\}$$

This situation is not consistent because the following integrity constraint

$$\neg suspended(..) \wedge good\_employee \supset take\_bonus(\cdots)$$

is violated. Thus we must reject the execution of the action *take_pardon*. Notice that in the set R_{Future} there is the static rule $\neg suspended(L_1) \wedge good\_employee(L_2) \supset take\_bonus(L_1 \cap L_2)$ which is executable in the situation

S_1'''. By Theorem 3.1 we have that when a static rule is executable in a situation then this situation is inconsistent. We use that in order to describe a way to find these inconsistent situations. The following algorithm discovers inconsistent situations.

Algorithm for checking consistency of a situation

1. After the completion of the execution of static rules in set R_{Past} do
   (a) In the produced situation try to evaluate the rules which belong to the set $R_{\text{Future}} \backslash R_{\text{Past}}$.
   (b) If at least one rule evaluates then the situation is inconsistent.
   (c) Else it is consistent.

This algorithm is executed every time that the previous algorithm returns a situation. If it returns an inconsistent situation the predicate *Acceptance* become false.

Theorem 5.3. *The above algorithm discovers all inconsistent situations.*

Proof. Assume that a situation S is inconsistent but the algorithm returns "consistent". Then there must be an integrity constraint Law_i which does not hold in S. Assume that $Law_i = C_1 \wedge \cdots \wedge C_n$. Then at least one of the $C_1, \ldots, C_n$ must be false. Assume that C_j is false and $C_j = f_1 \vee \cdots \vee f_m$. Thus f_l is false for each $l = 1, \ldots, m$. By Theorems 4.1 and 4.2 we have at least one pair $(f_k, f_p) \in I$ and $C_j \equiv f_k \vee f_p \vee C_j'$. There are two cases. First $f_p \in F_S$. Then by steps 3 and 4 of the algorithm of production of static rules we have that: $G_{f_p}(t,..) = G' \vee (\neg \bigwedge f_j(t_j, j = 1, \ldots, m, j \neq p))$ and $G_{f_p}(t,..) \supset f_p \in R_{\text{Past}}$. If all fluents f_j, $j = 1, \ldots, m$ are false then $(\neg \bigwedge f_j, j = 1, \ldots, m, j \neq p)$ is true. Thus $G_{f_p}(t)$ is true. This means that the static rule $G_{f_p}(t,..) \supset f_p(..)$ must be evaluated (before the calling of the consistency checking algorithm) and thus, f_p is true. Thus C_j is true and Law_i is satisfied. So the situation return Algorithm 2 which is not inconsistent. A contradiction.

Second case is $f_p \in F_p$. Then the rule $(G_{f_p}(t) \supset f_p) \notin R_{\text{Past}}$. Then by steps 3 and 4 of the algorithm of production of static rules we have that: $G_{f_p}(t) = G' \vee (\neg \bigwedge f_j(t_j, j = 1, \ldots, m, j \neq p))$. If all fluents f_j, $j = 1, \ldots, m$ are false then $(\neg \bigwedge f_j, j = 1, \ldots, m, j \neq p)$ is true. Thus $G_{f_p}(t,..)$ is true (at time point t). So that the rule is executable in situation S but Algorithm 1 did not evaluate as it does not belong to set R_{Past}. This rule belongs to R_{Future}. Thus $(G_{f_p}(t,..) \supset f_p(t,..)) \in (R_{\text{Future}} \setminus R_{\text{Past}})$. The algorithm discovered the above static rule is executable in S and returns inconsistency. A contradiction.[18]

Now we must prove that the algorithm which was present in the previous section together with the above algorithm always terminate in a finite number of steps and return a consistent situation.

Theorem 5.4. *The algorithms always return a consistent situation*

Proof. The above algorithm discovered all the inconsistent situations. The algorithm in the previous section before returning a situation calls the algorithm for discovery inconsistent situations. This means that it cannot return a inconsistent situation. If a situation is inconsistent then it call the algorithm of rejection actions. □

Theorem 5.5. *Algorithm* 2 *always returns a consistent situation*

Proof. The above algorithm discovered all the inconsistent situations which Algorithm 1 returns. Algorithm 2 before accepting a situation calls the consistency checking algorithm which discovered all inconsistent situations. This means that it cannot accept an inconsistent situation. □

Theorem 5.6. *The algorithms terminate in a finite number of steps.*

The proof is similar with Theorem 5.2.

[18] We have assume that the algorithm return consistent.

5.7. Case 3: The effects of changes of the past start to hold from the current time point

First we explain this case using an abstract example.

Consider in Fig. 10. Assume that the top line is the evolution of the world and at time point 11 the action a_2 takes place due to the changes the past at time 3. The evolution of the world before the execution of the action a_2 is

$$start(S_0) = 0 \quad end(S_0) = 3$$
$$start(S_1) = 3 \quad end(S_1) = 5$$
$$start(S_2) = 5 \quad end(S_2) = 7$$
$$start(S_3) = 7$$

We now evaluate the dynamic rule which corresponds to the action a_2 (at point 3). Then we evaluate the static rules until the current time point 11. Notice that we do not re-execute the action a_1 at time point 5. The new situation at time point 11 is S_4. Now we have that $end(S_3) = 11$ and $start(S_4) = 11$. The situations S_5 and S_6 after the end of "virtual" execution do not exist. The evolution of the world after the evolution is

$$start(S_0) = 0 \quad end(S_0) = 3$$
$$start(S_1) = 3 \quad end(S_1) = 5$$
$$start(S_2) = 5 \quad end(S_2) = 7$$
$$start(S_3) = 7 \quad end(S_3) = 11$$
$$start(S_4) = 11$$

As we observe the effects of the action a_2 start to hold from the current time point (time point 11), while it does not change the evolution of the world in the past (the situations S_5 and S_6 after the end of "virtual" execution does not exist).

In this case no fluent changes its truth value in the past. As we already explained we create a "virtual" sequence of situations from a time point in the past until the current time point but we adopt only the last as the current situation. We assume the past situations as they were before the execution of the action which gives us information about the past. Notice that in order to create the "virtual" sequence of situations we execute all the static rules. This happen because we adopt only the last situation and thus no fluent changes its truth value in the past. The important in this case is to produce a consistent situation which starts to hold from the current time point and encapsulates the effects which are created if we change the past.

In this case it is not necessary to assume that the situation and action axis are branching. The linear correspondence of Fig. 2 is sufficient. This happens because first we do not change the situation in the past but in the current time point, and second we do not re-execute actions in the past.

Consider the last execution of the previous chapter. In that execution the action $occur(take\_pardon(p), 26, 22)$ has been rejected because it has as consequence an inconsistent situation. Now we do not reject this action because we may execute all static rules and produce a consistent situation for time point 26.

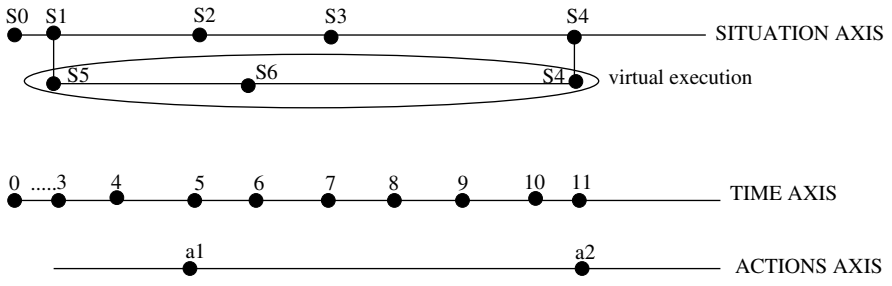

Fig. 10. The scenarios of execution.

Example 2 (*continued from Section 5.1*).

$$occur(take\_pardon(p), 26, 20)$$

The initial situation.

$$S_0 = \{\neg take\_bonus(p,[0,\infty]),\ \neg take\_salary(p,[0,21]), take\_salary(p,[21,\infty]),\\ suspended(p,[0,21]), \neg suspended(p,[21,\infty]),\ good\_employee(p,[0,\infty]),\ illegal(p,[0,26]),\\ \neg illegal(p,[26,\infty]), position(p,1,0,36), salary(p,3,0,0)\}$$

At time point 21 the static rules which corresponded to the fluents *suspended*, $\neg take\_salary$ will be evaluated and the new situation is

$$S_0' = \{\neg take\_bonus(p,[0,\infty]),\ \neg take\_salary(p,[21,26]),\ take\_salary(p,[26,\infty]),\\ suspended(p,[21,26]), \neg suspended(p,[26,\infty]),\ good\_employee(p,[0,\infty]),\ illegal(p,[0,26]),\\ \neg illegal(p,[26,\infty]), position(p,1,0,36), salary(p,3,0,0)\}$$

At time point 24 the actions *grant_promotion* and *grant_increase* cannot execute. At time point 26 the action *take_pardon* executes and changes the past at time point 20. The new situation at time point 22 is

$$S_1' = \{\neg take\_bonus(p,[0,\infty]), take\_salary(p,[21,\infty]),\\ \neg suspended(p,[21,\infty]),\ good\_employee(p,[0,\infty]),\ \neg illegal(p,[22,\infty]),\\ position(p,1,22,58), salary(p,3,22,22)\}$$

We now can execute all the static rules and the final situation will be

$$S_1 = \{take\_bonus(p,[22,\infty]), take\_salary(p,[21,\infty]),\\ \neg suspended(p,[21,\infty]), good\_employee(p,[0,\infty]), \neg illegal(p,[20,\infty]),\\ position(p,1,22,58), salary(p,3,22,22)\}$$

The difference from the above two cases is that there we had $start(S_1) = 22$, while now we want to find a situation S_1 s.t. if no action is executed from time point 22 until now (26), then S_1 is the situation in 26. Thus at time point 26 the new situation is

$$S_2 = \{take\_bonus(p,[22,\infty]),\ take\_salary(p,[21,\infty]),\\ \neg suspended(p,[21,\infty]), good\_employee(p,[0,\infty]),\ \neg illegal(p,[20,\infty]),\\ position(p,1,26,62), salary(p,3,26,26)\}$$

As we observe at time point 26 the actions *grant_promotion* and *grant_increase* must be executed. Thus the new situation is

$$S_2 = \{take\_bonus(p,[22,\infty]), take\_salary(p,[22,\infty]),\\ \neg suspended(p,[22,\infty]), good\_employee(p,[0,\infty]), \neg illegal(p,[22,\infty]),\\ position(p,2,26,0), salary(p,4,26,0)\}$$

We want an algorithm which produces the current consistent situation. The following algorithm addresses the ramification problem in the last case.

Algorithm 3 for producing a consistent situation

1. If an action ($occur(a, t, t_1)$) which changes the past ($t_1 < t$) is executed then
   (a) Execute the dynamic rule at situation S s.t. $Actual(S, t_1)$.
   (b) Execute the static rules in the new situations until no change occurs.
   (c) At each time point until the current time point execute the static rules.
   (d) Return the last situation S'.
   (e) Set $Actual(S', t)$.
2. Else execute the dynamic rule and afterwards the static rules until no change occurs.

In this case there is no case of infinite loops because we do not re-execute the actions.

Theorem 5.7. *The above algorithm returns a consistent situation in the case that the action change the belief about the past but the effect start to hold from the current time point.*

Proof. Assume that the algorithm returns an inconsistent situation.

So there is an integrity constraint which is not satisfied. Assume that integrity constraint Law_j is not satisfied in one situation. Assume that the CNF of this law is $C_1 \wedge \cdots \wedge C_n$. Then it must be the case that one of the $C_1, \ldots, C_n$ is false. Assume that $C_i = f_1 \vee \cdots \vee f_m$ is false. Then all fluents $f_j, j = 1, \ldots, m$ are false. Then by Theorems 4.1 and 4.2 there are fluents f_k and f_p such that $(f_k, f_p) \in I$ and $C_i \equiv f_k \vee f_p \vee C_i'$. Then by steps 3 and 4 of the algorithm of production of the static rule we have that: $G_{f_p}(t,..) = G' \vee (\neg \bigwedge f_j(t_j, j = 1, \ldots, m, j \neq p))$. If all fluents $f_j, j = 1, \ldots, m$ are false then $(\neg \bigwedge f_j, j = 1, \ldots, m, j \neq p)$ is true. Thus $G_{f_p}(t,..)$ is true. This means that the static rule $G_{f_p}(t,..) \supset f_p(\cdots)$ must be evaluated and thus, f_p is true. This will happen because at each step of the above algorithm we evaluate the static rules until no change occurs. Thus at each time point we evaluate the static rules until no change occur. A contradiction. □

6. Conclusion and future work

6.1. Summary

In this paper we examine two infamous problems, *ramification* in the setting of temporal databases.

The ramification problem in temporal databases has many different views depending on the assumptions one makes. Almost all solutions which have been proposed for the ramification problem are based on the persistence of fluents. This mean that nothing changes except if an action takes places. This assumption simplifies the solution of the ramification problem, but it is restrictive, because in a temporal reasoning setting the persistence of fluents is unreasonable. This happens because when time consideration are imported, one action could have as effect that the fluent f hold for t time points after the execution of an action. The solutions based on the persistence of fluents cannot encapsulate effects like the above. To achieve this we must specify the time point of an action execution as well as the duration of its effects.

In order to address the ramification problem in temporal databases we propose an extension of the situation calculus in order to encapsulate the time. We propose a new representation of fluents in order to be able to encapsulate the non-persistent effects of actions. More specifically, each fluent f is represented as $f(L)$, which means that the fluent f is true in the time intervals in the list L. Each element of the list L is a time interval $[a, b]$, $a < b$.

In a temporal context, we need to describe the direct and indirect effects of an action not only in the immediately resulting next situation but also possibly for many future or past situations as well. This means that the world being modeled may change from one situation to another while the direct and/or indirect effects of an action still hold. Also, in this time span other actions may occur leading to many different situations.

We address the ramification problem in a temporal database when the action changes the beliefs about the past. The linear correspondences of Fig. 2 cannot represent the correspondence between situations, actions and time in the case that the effects of the actions refer to past. This happens because after the execution of an action which changes the past we repeat the execution from the time point (in the past), to which referred the effects of the action. Thus we need a branching axis for the situation and an action axis. The time axis remains linear. We propose the correspondence shown in Fig. 6. When an action changes the past we start two new linear axes, one for the situations and one for the actions. As we observe each time there is a linear line in the branching axis of situations which is the "actual" line. This line contains the situations which are the history of time point t. Because we repeat the execution of actions after an action changes the belief about the past there is the problem of infinite loops. We distinguish three cases and provided a solution for each one of them:

- When an action could change the value of all the fluents in the past. In that case the only problem is which there are infinite loops.
- When an action could change the value in only some fluents in the past. In that case there are different sets of fluents. The first set F_P contains the fluents which cannot change their truth value in the past, and the

second set F_S contains the fluents which can change their truth value in the past.
In addition to the above problem we must ensure that the fluents which belong to F_P do not change their value in the past. We extended the algorithm which was appropriate for the first case.

- When an action could change the value of all the fluents in the past but the effects affect only the current situation. In this case it is not necessary to assume that the situation and action axes are branching. The linear correspondence of Fig. 2 is enough. This happens because we do not change the situation in the past but in the current time point and because we do not re-execute actions in the past.

6.2. Future work

As a future work we intend to address the ramification problem in the case that two or more actions are executed concurrently and their effects may refer both in the past and in the future. Assume the following execution:

$$occur(grant\_bonus(p), 10, 7)$$
$$occur(bad\_grade(p), 10, 11)$$

This means that the following must hold:

$$take\_bonus(p, [[7, \infty]]) \wedge bad\_employee(p, [[11, \infty]])$$

This is consistent but

$$bad\_employee(p, [[11, \infty]]) \supset \neg take\_bonus(p, [[11, \infty]])$$

Finally the following:

$$take\_bonus(p, [[7, \infty]]) \wedge \neg take\_bonus(p, [[11, \infty]])$$

must hold. Now there are two choices. First to reject the execution of the two actions and second to accept

$$take\_bonus(p, [[7, 11]]) \wedge \neg take\_bonus(p, [[11, \infty]])$$

In that case there is the problem in which situation to execute the dynamic rules, and how to construct the actual line between the smallest time point in the past (which some actions referred) and the time point in the future. In our example we first executed the action which referred in the past and after the action which referred in the future, etc.

Also as a future work we want to examine the ramification problem in the case when the effects of one action is non-deterministic. For example suppose there is a new action *grant_accolate* which has as direct effects to take bonus or take a pardon but not both of them. Consider the following execution:

$$occur(misdemeanor(p), 8)$$
$$occur(grant\_accolate(p), 10)$$
$$occur(grant\_promotion(p), 11)$$

As we observe if the action *grant_accolate* which takes place at time point 10 has as direct effect to take pardon the public worker p then the action *grant_promotion* could be executed at time point 11. Else if it has as direct effect to take bonus the public worker p, then the action *grant_promotion* cannot be executed at time point 11.

References

[1] M. Ginsberg, D. Smith, Reasoning about action I: a possible worlds approach, Artificial Intelligence 35 (1988) 165–195.
[2] A.C. Kakas, R.S. Miller, F. Toni, E-RES: reasoning about actions, events and observations, in: Proceedings of LPNMR2001, Springer-Verlag, Berlin, 2001, pp. 254–266.
[3] A. Kakas, R. Miller, A Simple Declarative Language for Describing Narratives with Actions, The Journal of Logic Programming 31 (1–3) (1997) 157–200 (special issue on reasoning about action and change).

[4] R.A. Kowalski, Database updates in the event calculus, Journal of Logic Programming (1992).
[5] V. Lifshitz, Towards a metatheory of action, in: J.F. Allen, R. Fikes, E. Sandewall, (Eds.), Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, MA, 1991, pp. 376–386.
[6] V. Lifshitz, Frames in the space of situations, Artificial Intelligence 46 (1990) 365–376.
[7] V. Lifschitz, Towards a metatheory of action, in: Proceedings of KR'91, Cambridge, MA, 1991, pp. 376–386.
[8] N. McCain, H. Turner, A causal theory of ramifications and qualifications, in: Proceedings of IJCAI-95, Montreal, Canada, August 1995, pp. 1978–1984.
[9] J. McCarthy, P.J. Hayes, Some philosophical problem from the standpoint of artificial intelligence, in: B. Meltzer, D. Mitchie (Eds.), Machine Intelligence, vol. 4, American Elsevier, 1969, pp. 463–502.
[10] R. Miller, M. Shanahan, The event calculus in classical logic—alternative axiomatisations, Linkping Electronic Articles in Computer and Information Science 4 (16) (1999).
[11] N. Papadakis, D. Plexousakis, Action theories in temporal databases, in: Proceedings of the 8th Panhellenic Conference on Informatics, Cyprus, November 2001, pp. 254–264.
[12] N. Papadakis, D. Plexousakis, The ramification and qualification problems in temporal databases, in: Proceedings of the 2nd Hellenic Conference on AI, Lecture Notes on Artificial Intelligent vol. 2308, 10–11 April 2002, Thessaloniki, Greece, pp. 18–30.
[13] N. Papadakis, D. Plexousakis, Action with duration and constraints: the ramification problem in temporal databases, in: 14th IEEE ICTAI, 2002 , Washington, DC.
[14] N. Papadakis, D. Plexousakis, Action with duration and constraints: the ramification problem in temporal databases, International Journal of Artificial Intelligent Tools (IJTAI) (special issue on selected papers of ICTAI 2002), in press.
[15] D. Plexousakis, J. Mylopoulos, Accomodating integrity constraints during database design, in: Proceedings of EDBT 1996, Avignon, France, 1996, pp. 497–513.
[16] J. Pinto, Temporal reasoning in the situation calculus, Ph.D. thesis, Department of Computer Science, University of Toronto, January 1994.
[17] J. Pinto, R. Reiter, Temporal reasoning in logic programming: a case for the situation calculus, in: Proceedings of 10th International Conference on Logic Programming, Budapest, Hungary, June 21–24, 1993.
[18] R. Reiter, Natural actions, concurrency and continuous time in the situation calculus, KR 96, 1996, pp. 2–13.
[19] M. Thielscher, Ramification and causality, Artificial Intelligence 89 (1–2) (1997) 317–364.
[20] M. Thielscher, Reasoning about actions: steady versus stabilizing state constraints, Artificial Intelligence 104 (1988) 339–355.
[21] M. Winslett, Reasoning about action using a possible models approach, in: Proceedings of AAAI-88, Saint Paul, MN, August 1988, pp. 89–93.

Further reading

[1] C. Elkan, Reasoning about action in first order logic, in: Proceedings of the Conference of the Canadian Society for Computational Studies in Intelligence (CSCSI), Vancouver, May 1992, pp. 221–227.
[2] A. Fusaoka, Situation calculus on a dense flow of time, in: Proceedings of AAAI-96, 1996, pp. 633–638.

Nikos Papadakis is Visiting Professor of Computer Science at the University of Crete, Greece and researcher at the Institute of Computer Science, Foundation for Research and Technology—Hellas (ICS-FORTH). He teaching Databases and Programming and supervise more than 70 graduate thesis. He receives bachelor from Computer Scieznce Department at the University of Cyprus at 1997, and his M.Sc. and Ph.D. from Computer Science Department of the University of Crete at 1999 and 2004, respectively. His research interests are: Databases and Artificial intelligence: Integrity constraints, Knowledge Representation and Reasoning, belief revision, Distributed algorithms: Algorithms models, communication protocols, fault tolerance, Networks: Protocol and Algorithms for Routing, Resource Management.

Grigoris Antoniou is Professor of Computer Science at the University of Crete, Greece and head of the Information Systems Laboratory (ISL) at the Institute of Computer Science, Foundation for Research and Technology—Hellas (ICS-FORTH). He studied Computer Science at the University of Karlsruhe, Germany, and earned a Ph.D. in 1987 at the University of Osnabrueck, Germany. Prior to coming to Crete he held professorial appointments at Griffith University, Australia, and the University of Bremen, Germany. His research interests lie in the field of theory and application of logic-based knowledge representation. Particular interests include nonmonotonic reasoning, rule-based systems, the semantic web, and web services. He has published over 100 technical papers in international journals and conference proceedings. Finally he is the author or co-author of three books published by international publishers, including the forthcoming "A Semantic Web Primer" by the MIT Press.

Dimitris Plexousakis is an Associate Professor, Associate Chair and Director of Graduate Studies at the Department of Computer Science, University of Crete and a Researcher at the Information Systems Laboratory of the Institute of Computer Science, FORTH in Greece. His research interests span the following areas: Knowledge Representation, Knowledge Base Design; Distributed Database Systems and Databases on the Web; the Semantic Web; Formal reasoning systems, applications of artificial intelligence in database systems; Business process and e-service modeling. He has published over 60 articles in international conferences and journals and has served on the program committees of numerous international conferences and journal editorial boards. He was the executive chair of the 9th International Conference on Extending Database Technology (March 14–18, 2004, Heraklion, Greece) and the Program Committee co-Chair of the 3rd International Semantic Web Conference (November 7–11, 2004, Hiroshima, Japan). He is leading the ERCIM Working Group on the Semantic Web.