UNIVERSITY OF CRETE
DEPARTMENT OF COMPUTER SCIENCE

Actions Theories in Temporal Databases

NIKOS PAPADAKIS

DOCTORAL THESIS

HERAKLION CRETE GREECE
MARCH 2004

Actions Theories in Temporal Databases

Nikos Papadakis

DOCTORAL THESIS

Department of Computer Science

University of Crete, Greece

Abstract

Reasoning about action and change has been one of the main research themes of the knowledge representation and planning communities of the last 15 years. Action theories providing an axiomatic basis for managing change are applicable to a wide area of disciplines including software engineering, (cognitive) robotics and data/knowledge base systems. In this thesis, we first review action theories proposed for reasoning about the dynamics of database systems. We examine how these theories deal with the three infamous problems associated with this area, namely: (a) the **frame problem**, which refers to the identification of predicates or functions that remain unchanged as a result of action execution, (b) the **ramification problem**, which refers to determining the indirect effects of actions, and (c) the **qualification problem**, which refers to determining the preconditions which must hold prior to the execution of an action. We briefly describe the solutions which have been proposed for these problems and position these problems in a temporal database context. We also introduce an abstract solution for the frame and qualification problems in temporal databases based on the language of situation calculus.

In this thesis we concentrate on the ramification problem. The ramification problem is of great importance to database systems. The users and designers may not know exactly all the indirect effects of their transactions and such indirect effects mays violate the integrity constraints. A solution to the ramification problem permits the designers to discover errors in their design and to correct them. All the solutions to the ramification problem in conventional databases are based on the idea of the persistence of fluents. This means that nothing changes until an action takes place. The above solutions cannot produce automatically the non persistent effects. If such effects exists, they have to be described manually. This is very difficult and impractical in the large databases with complex transactions. In temporal databases the predicate and function values can change as time progresses without any action taking place. Thus the above solutions cannot solve the ramification problem in temporal databases.

We examine the ramification problem in the following cases: (a) sequential action execution both for instanteous actions and actions with duration when an action may have effects only in the future, (b) concurrent action execution both for instanteous actions and actions with duration when an action may have effects only in the future and (c) sequential action (instanteous) execution when an action may change beliefs about the past. The problem in the last case is very complex with many philosophic aspects. For each case, we propose a solution which permits to the designers to determine the effects of their transaction, to discover the errors and to correct them.

Supervisor:
Dimitris Plexousakis,
Professor of Computer Science,
University of Crete

To my parents

Acknowledgments

First, I would like thank my supervisor Prof. Dimitris Plexousakis, for providing me with the quidance and support over course of my research. Prof. Plexousakis has spent many hours with me discussing all of the topics of my thesis and writting papers. I would like thank the other members of my advisory committee (Prof. Grigoris Antoniou and Prof. Panos Constantopoulos). More importantly, i want to thank them for teaching me what constitutes quality research.

I would like to thank of my disseration committee from the University of Crete, Professors Vassilis Christophides, and George Georakopoulos, as well as the external members Professors Stavros Christodoulakis and Manolis Koubarakis (Technical University of Crete), for their helpful comments and questions.

Especially, I want to express my gratitude to Professor Grigoris Antoniou. Prof. Antoniou has spent many hours with me discussing my thesis and writting papers.

Finally, I would like to acknowledge my parents and my brothers and my sister for their support.

Contents

List of Figures

1 Introduction

A database is a collection of data and relations among them. With the term data we mean facts which have evident importance and can be stored. A database has three basic attributes, namely

1. It represents a few views of real world which are called mini-worlds. Any change in any of the mini-worlds will be reflected in the database.
2. The data which are stored in a database are connected among them.
3. The database is created in order to serve a concrete aim.

A world represented in a database is not static. This means that it changes continuously. These changes are reflected in the database via the change of data which are stored in the database (with insertion of new data, deletion or changes in the values of the data).

For example, assume a database in which the grades of the students of a department are stored. The students receive new grades in courses in which they were not examined in the past (appending new data) or improve their grade in some course they had been examined previously (change of value of the existing data).

Data are subject to restrictions (constraints) which refer to the values the particular relationships that may exist among them may assume. In the previous example a constraint is that the grade that a student takes cannot be grater than 10 or smaller than 0. Another constraint can be that the grade should always be an integer or decimal but the decimal part must be 0.5. There may exist more complicated constraints as e.g., when a student is re-examined in a course then we must store the grater of the grades received. All these constraints are named integrity constraints.

In a database the data that are stored should satisfy the integrity constraints in order for the database to be considered consistent.

1.1 Consistency of the Database

The guarantee of consistency of data that is stored in a database is a very important and difficult problem. The consistency of data, is determined by the satisfaction of integrity constraints in the different database states. A database state is considered valid only when all integrity constraints are satisfied.

Static constraints express properties that should hold in any database state. Syntactically, all references to the database state in question are made through predicates or functions that are interpreted in the database state whose consistency needs to be determined. Dynamic constraints on the other hand, may involve the comparison of predicate truth values in several states. These cannot be evaluated in a single state. Transition constraints are a special case of dynamic constraints. They express properties related to the transition from one state to the next.

In databases without time, static and transition constraints are enough in order to ensure the consistency of the database. The presence of time in databases (temporal databases) however renders essential the utilization of dynamic integrity constraints. This happens because the execution of an action (delete, update, insert) in such databases will have consequences not only in the current but also in past/future situations. Such an example is a constraint referring to the grade of students which attend for a second time some course and finally take a smaller grade in the course than the grade they had originally received. Then, the older grade must be kept and not the new grade. This constraint can be expressed as the following formula of many sorted 1st-order predicate calculus:

$$(d/Date \;\; l/Lessons \;\; st/Student) \; grades(l, st, d, gr) \wedge grades(l, st, today, gr') \wedge d < today \supset gr' > gr \,.$$

In order to enforce this constraint the following condition must be verified when a new grade is assigned on a date d' to the student in the same lesson l that he previously attended:

$$\neg grades(l, st, d', gr') \wedge gr' < gr$$

This condition is sufficient in order to ensure that the constraint will not be violated. The general problem of ensuring database consistency is very hard. The difficulty comes from the potential number of database states in which conditions must be evaluated and from the difficulty in determining the satisfaction of logical formulae. This becomes even worse if we take into consideration the indirect effects that actions may have and which arise because of the presence of rules that implicitly generate consequences (deductive rules, integrity constraints).

Consider the above example and assume that apart from of the grades, the database contains information about which course the student attended and which is the current average grade. The action $take-new-grade(p, l, gr, t)$ means that the student p has taken the grade gr in the course l at time point t. In order for the action $take-new-grade$ to execute, the student must attend the corresponding course. Thus, the precondition which must hold in order to enable the execution of the action $take-new-grade$ is that the student attends the course l at time t. The problem which refers to the determination of the preconditions of an action is called the **qualification problem**.

As we have already mentioned the action $take-new-grade(p, l, gr, t)$ may or may not change the contents of the database. This depends on whether the student p has already received a grade in the course l. If s/he has then, if the original grade is greater than the last, the action $take - new - grade(p, l, gr, t)$ does not change the content of the database; otherwise the action change the content. The problem which refer to determining what remains unchanged by an action is the **frame problem**.

Assume that the database stores information about students that have graduated. A student graduates if s/he succeeds in more than 30 courses. Thus, at each time point the database must satisfy the integrity constraints

$$grades(l_1, st, d_1, gr_1) \wedge \ldots \wedge grades(l_n, st, d_n, gr_n) \wedge n \geq 30 \supset graduate(st) \qquad (1)$$
$$grades(l_1, st, d_1, gr_1) \wedge \ldots \wedge grades(l_n, st, d_n, gr_n) \wedge n < 30 \supset \neg graduate(st) \qquad (2)$$

where gr_i is the grade that the students takes at the course l_i. Assume that a student st has succeeded in 29 courses. When the action $take-new-grade$ (refers to a new course) takes place and changes the content of the database, the first integrity constraint is not satisfiable in the new database state. In order for it to be satisfiable the predicate $\neg graduate(st)$ must be changed to $graduate(st)$. This change is the indirect effect of the action $take - new - grade$. The indirects effects of an action arise because of the integrity constraints. The problem which refers to the determination of the indirect effects of an action in the presence of integrity constraints is the **ramification problem**.

1.1.1 The Importance of the Ramification Problem in Databases

The ramification problem is of great importance to database systems. Database users and designers may not known exactly all the indirect effects of their transactions. This means that the users/designer can execute a transaction which has as result to violate the integrity constraints. The most obvious solution is the designer and the users determine manually all the indirect effects. The problem with this solution is that in a large database with a large number of constraints and transactions, the indirect effects may be too many to be taken into account manually. Thus, we need an automatic way which determines the indirect effects of transactions and enables the verification of constraints.

A solution to the ramification problem permits to the designers to realize the effects of their design. For example a transaction can produce an inconsistent situation or can produce a situation which contains undesirable indirect effects. This means that the designers can discover erroneous specifications. This discovery is very difficult or impossible to be done manually. The solution of to ramification problem permits to the designer to discover errors in the design of database (database schema, integrity constraints, transactions) and redesign the database if this is necessary.

The solution to ramification problem is necessary in databases because it enables the design of correct, reliable and consistent databases. Many solutions have been proposed for the ramification problem in conventional databases. As we explain below none of them address in satisfactory manner the ramification problem in temporal databases.

When time is present an action could have an effect which holds for a time interval. For example consider the action $registration(p, l, t)$ which means that the student p registers for the course l at time t. This means that the student attends the course l for the next semester. Assume that the duration of the semester is 6 months. Then the predicate $attend(p, l, t_1)$ is true if $t \leq t_1 \leq t + 6$. After the time point $t + 6$ the predicate $attend(p, l, t_1)$ is false without any action taking place. This means that the effects of an action in a temporal database may not persist. All the solutions to the ramification problem in conventional databases are based on the idea of the persistence. This means that the non persistent effects must be described manually. In most cases, this is impossible because the users/designers do not known all the non-persistent effects. If the designers determine all the indirect effects manually then they solve the ramification problem manually. As we have already explained this is practically impossible in large databases. For the above reasons, the solution to the ramification problem in conventional databases cannot be used in the temporal databases.

The purpose of this thesis is to address the ramification problem in temporal databases. We describe the ramification problem in temporal databases and provide a solution which encapsulate non persistent effects.

1.2 Action Theories

We can assume that a database system is an agent which interacts with its environment. This agent needs precise knowledge as to the effects of its actions in order to act purpose-oriented and so to achieve pre-determined goals. The latter requires to draw the right conclusions from this knowledge in view of particular situations. As we can conclude all effects of the agent can assume as effects which produces by the execution of some actions. Thus, is very important to be able to describe these effects. Action theories providing an axiomatic basis for managing changes which happens as result of the execution some actions.

An atomic database transaction can be considered as an action. Thus, we can assume that the changes in a database occur as results of actions. Actions have direct and indirect effects which may affect the integrity constraints. This means that the database may not be consistent after the execution of an action. From the execution of the actions arise the three infamous problems **frame, ramification** and **qualification** which we presented above and we describe in detail in the following sections. First, we are produce some definitions about action theories as Thielscher [122] gives these.

"An action theory consists of a formal language that allows adequate specifications of action domains and scenarios, and it tells us precisely what conclusions can be drawn from these specifications."

In order to define these more formally, we must give a definition of all the crucial terms used.

First, we must define the action domain. By the action domain we mean any aspect of the world worth formalizing in which the execution of actions plays a central role.

Second, we must define the action scenario. By action scenario we mean exactly these particular situations which give us some information for past, current and future state of the world (which we describe in the database).

Finally, we must determine a way that allows us the specification of actions and their effects as naturally as possible.

Action theories have much in common with logic. They are based on a formal language and they include an entailment relation among the expressions in this language.

1.3 The purpose of this Thesis

In the database literature the frame, ramification and qualification problems have been addressed extensively. All the solutions which have been proposed describe how the transition from the current state to the next is realized. In this case, the truth values of predicates and functions do not change unless an action takes place.

In temporal databases, this assumption is very restricting because the effects of an action could refer to a temporal interval and not to the next situation only. This happens because when time is present an action could have as effect that the fluent f holds for t time points after the execution of an action. The solutions which are based on the idea of the persistence of fluents cannot encapsulate effects like the above. Because these solutions assume that any change occurs only after an action take places, if we want to represent no persistent effects we must define a "new" action for each non-persistent effect which shows the changes. Also we must determine the time of execution for each "new" action. This has two major disadvantages. First, if some other action cancels the non-persistent effect then the corresponding "new" action must be cancelled. Second, the number of actions may greatly increase because we must define one action for each non persistent effect of each action(e.g. if each action has two non persistent effects then if A is the number of actions we need $2 \times A$ extra actions). Thus the complexity increases significantly.

Previous proposals about addressing the ramification problem have employed so called *causal relationships* [21, 82, 43, 124, 125]. They come short in 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 fluents assumption (i.e., no fluent may change truth value without an action taking place). The same weakness characterizes all other solutions of the ramification problem in conventional databases (e.g., [131, 62, 63, 64, 14, 50, 51]).

The most prevalent previous works, in temporal databases, are those by Reiter [105], Reiter and Pinto [101, 102] and by Kakas [52, 53]. 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. Kakas [52, 53] 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 axiom schemas for the description of the world. All these are works based on the idea of the persistence of fluents.

The assumption of the persistence of fluents makes easier addressing the problem but also restricts the problem very much. In the case that there are no persistent effects the above solutions cannot encapsulate them except if we describe explicitly all the effects.

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 possibly for many future situations as well. These effects refer to a time interval and not to the next state. This means that the database state could change without an action taking place.

In this thesis we focus on the **ramification** problem which has many different views in temporal databases.

1.4 Organization of Thesis

The rest of this thesis is organized as follows:

In chapter 2 we present the frame(2.1), the ramification(2.2) and the qualification (2.3) problems in Conventional Databases and we review the most important solutions that have been proposed.

In chapter 3 we address these problems in temporal databases and we propose one solution for the frame and qualification problems. Also, we present the most important previous work on the ramification problem in temporal databases and we extended the situation calculus in order to encapsulate time.

In chapter 4, in section 4.1.1 we present an algorithm in order to discover the dependencies which there are between the fluents, while in the rest section 4 we address the ramification problem in temporal databases with many different assumptions (when an action changes only the current and the future) and we present solution of all these cases.

In chapter 5 we address the ramification problem in case that an action could change the beliefs about the past.

Finally, in chapter 6 we present the conclusion of this thesis and the future works.

2 Action Theories in Conventional Databases

2.1 Frame Problem

As it was explained in the previous chapter the ability for comprehension of changes that happen in a database is very important.

For example assume a database in which we store the things that exist in a room with their place. Assume that there exists a bookcase with books and a lamp on a table. When we move the lamp from the middle of table to an end, the position of the bookcase or books does not change. When we move the bookcase in a new place then the position of books as well as all the things which are on the bookcase change..

As we can see from the above example, some things are influenced by some actions while others remain unaffected. The problem of the determination of predicates or functions that are not influenced when some action takes place, is called the frame problem and it was first defined by McCarthy [81] in 1969.

2.1.1 Monotonic Approach

Various solutions have been proposed for addressing the frame problem. The simplest among these is the monotonic approach, which was proposed by MacCarthy [81]. According to this approach, there exist explicitly declared axioms from which one can draw conclusions about which predicates and functions are affected by each likely action. These axioms are separated in two categories

1. the action axioms which indicate which propositions are affected when some action takes place.
2. the frame axioms which indicate the things which remain unaffected when some action takes place.

A simple example is when a robot moves things in a house. If it moves an object x from its current place to another place l, then in order for it to be possible, two conditions must hold: first, no object should be in l and no object should be above the object x. In this case, as soon as the movement takes place, the object x will be found in the place l. The action axiom which describes the above change is

$$clear(x)_s \wedge clear(l)_s \rightarrow on(x,l)_{do(move(x,l),s)}$$

The predicate $clear(x)_s$ indicates that x is free in the situation s. The $on(x,l)_{do(move(x,l),s)}$ means that the object x will be in the place l after the execution of action $move(x,l)$. The

$do(move(x,l),s)$ denotes situation which results after the execution of action $move(x,l)$ in the situation s. Apart from the above action axioms, the following frame axioms are necessary

$$\begin{aligned} &on(x,l')_s \wedge y \neq x \ \rightarrow on(x,l')_{do(move(y,l),s)} \\ &color(x,c)_s \rightarrow color(x,c)_{do(move(u,v),s)} \\ &shape(x,c)_s \rightarrow shape(x,c)_{do(move(u,v),s)} \end{aligned}$$

The first axiom states that some object is not moved if it is not on some other object which was moved. The second states that any object which is moved does not change the color of any object and the third that if an object moved, its shape does not change.

This method has two very important disadvantages. The first is that are must determine explicitly all the frame axioms for each action separately. This means that, for each predicate, it must be determined whether it changes or not after the execution of an action. If the number of actions is a and the number of predicates is r then the number of frame axioms will be in the order of $a \times r$.

The second disadvantage is complexity. In order to determine what remains true after an action, all predicates must be examined. Thus, when an action takes place, then if there exist n facts in the database, all must be examined.

2.1.2 Default approach

The bigger problem with the monotonic approach is that are should determine whether some object remains unaffected when an action takes place. Usually, when an action takes place, it changes very few objects while most remain unaffected. The Default approach [104] proposes that it is necessary to provide axioms only for these predicates which change as the result of a concrete action, while for the rest, there exists a default axiom which states that, each object which has not been declared to change after an action, remains as is.

For example the default axiom could have the form

$$\frac{p_s : p_{do(a,s)}}{p_{do(a,s)}}$$

which means that if the predicate p is true in the situation s and after the execution of action a it continues to hold, we can conclude p after the execution of action a. This method does not suffer from the first disadvantage of the monotonic approach but has large evaluation complexity. This happens because the default axiom will be evaluated for all predicates after the execution of an action, in order to ascertain which of them are true.

2.1.3 STRIPS Approach

This approach was proposed by Fikes [23] in 1971 and it is based on the observation that the world does not change very much from one instance to the other. This method adopts a simple model which induce the preconditions, the append list and the delete list.

Preconditions are conditions which they should hold in order for the execution of an action to be possible. When an action executes then all predicates that become true, are added in the append list, while they are removed from the delete list.

2.1.4 Situation Calculus

The Situation Calculus, proposed by MacCarthy [81] in 1969 for the solution of the frame problem, provides a formalism for actions and their effects(direct and indirect).

The Situation Calculus is a second-order language which has been designed for the representation of changes that takes place in a world of interest. All the changes that happen in a world are the result of the execution of some actions. A likely evolution of the world is a sequence of actions and is represented by a first order term, which is called a situation. The initial situation is symbolized with S_0 and it is the situation in which no actions have happened. In the Situation Calculus the binary function *do* is defined, with $do(a, s)$ denoting the situation that will result from the execution of action a in the situation s. An action could be $put(x, y)$ which means that object x is placed on the object y. For example $do(put(x, y), do(put(y, z), do(z, p)))$, means that first z will be placed on p, then y will be placed on z and finally x will be placed on y.

Generally the values of predicates and functions in a dynamic world differ from one situation to the next. The predicates and the functions whose value changes from one situation to another are called *fluent predicates* and *functional fluents*, respectively. For the rest of the thesis, we refer to these as fluent. An example of a fluent is $colour(x, s)$, which denotes the color of object x in situation s.

For the execution of an action to be possible must hold *some conditions*. We call these preconditions, as we have already mentiened. The binary predicate *Poss* declares whether a precondition holds. When the predicate $Posss(a, s)$ is true then the action a can be executed in the situation s. For example, assume that a robot wants to pickup object x. In order for it to be possible, the robot must not hold something else, the object x must not be very heavy and the robot must be close to the object x. These are shown below:

$$\begin{array}{l} Poss(pickup(r, x), s) \supset \\ [(\forall z)\neg holding(r, x, z) \wedge \ \neg heavy(x) \wedge nextTo(r, x, s)\,. \end{array}$$

The predicate *Poss* expresses the necessary conditions which must be hold before the execution of action pickup.

The next step is the determination of causal laws that show how actions influence the values of fluents. These laws are named rules of result. For example, when a robot drops a fragile object on the floor, then this object will break. The rule expressing the causal relationship is expressed as:

$$fragile(x, s) \supset broken(x, do(drop(r, x), s))$$

If a robot repairs an object then, the action has as effect that the object is no longer broken.

$$\neg broken(x, do(repair(r, x), s))$$

The action of painting an object with color c has as effect

$$color(x, do(paint(x, c), s)) = c\,.$$

A basic axiom in situation calculus is the axiom of induction

$$(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \;\supset (\forall s)P(s)$$

This axiom limits the set of situations to be isomorphic with the set **S** so that the following conditions satisfied

1. $S_0 \in \mathbf{S}$, where **S** is the set of situations.
2. If $s \in \mathbf{S}$, and $a \in \mathbf{A}$, then $do(a, s) \in \mathbf{S}$, where **A** is the set of action in the model.

As we may observe from in second condition, the model includes only deterministic actions. This means that $do(A, s)$ refers to a unique situation.

Then a partial order $<$, is definied on situations. $s < s'$ means that the situation s' results from s with the execution of one or more actions. From this two more axioms result:

$$(\forall s.s \neq S_0)S_0 < s \;\; and \;\; \neg s < S_0$$
$$(\forall a, s, s').s < do(a, s') \equiv Poss(a, s') \;\wedge s \leq s'$$

where $s \leq s'$ means $s < s' \vee s = s'$.

Let P be a binary relation between situations. Then from the above three axioms, we can conclude that

$$(\forall s, s').s < s' \equiv (\forall P).\{[(\forall a,\ s_1).Poss(a, s_1) \supset P(s_1, do(a, s_1))] \wedge$$
$$[(\forall a, s_1, s_2).Poss(a, s_2) \wedge P(s_1, s_2)\ \supset P(s_1, do(a, s_2))]\}$$
$$\supset P(s, s').$$

This conclusion means that the smallest relation between two situations is $\leq$.

The Situation Calculus has big expressive power and has been used for the solution of the frame problem by several researchers. Two of the most important solutions for the frame problem based on situation calculus have been proposed by Pednault and by Hass (the other important solution proposed by Reiter).

The first solution was proposed by Pednault [91]. Consider a simple electric circuit that includes several lamps and each one of them has its own switch. When lamp is on(off) and the corresponding switch is flipped, then the lamp will turn off (turn on,resp.). Thus,

$$\neg on(x, s) \supset on(x, do(flip(x), s))$$
$$on(x, s) \supset \neg on(x, do(flip(x), s))\,,$$

The above axioms are equivalent with

$$\neg on(x, s) \wedge y = x \supset on(x, do(flip(y), s))$$
$$on(x, s) \wedge y = x \supset \neg on(x, do(flip(y), s))\,,$$

Finally we draw the conclusion

$$on(x, s) \wedge \neg on(x, do(flip(y), s)) \supset on(x, s) \wedge y = x\,.$$

which is equivalent with

$$on(x, s) \wedge y \neq x \supset on(x, do(flip(y), s))\,.$$

This is a positive frame axiom, which means that the action $flip(y)$ does not influence fluent $on(x, s)$ when y is different from x. Negative frame axioms can be written in similar way.

$$\neg on(x,s) \wedge y \neq x \supset \neg on(x, do(flip(y), s))\,.$$

Generalizing, assume a set of positive and negative frame axioms (one for each action). Then, for each fluent $F(x,s)$

$$(\epsilon^+)_F(x,y,s) \supset F(x, do(A(y), s))$$
$$(\epsilon^-)_F(x,y,s) \supset \neg F(x, do(A(y), s))\,.$$

$(\epsilon^+)_F(x,y,s)$ shows the preconditions that should hold, so that the execution of action $A(y)$, the fluent F becomes true ($(\epsilon^-)_F(x,y,s)$ is defined respectively). In the above example (where $\neg on(x,s) \wedge y \neq x \supset \neg on(x, do(flip(y), s))$ is the negative frame axiom, we have that $(\epsilon^-)_F(x,y,s) = (y \neq x)$. This means that when $\neg on(x,s)$ holds, in order for it to continue to hold after the execution of action $flip(y)$, it should be the case that $y \neq x$ holds.

Assume that $F(x,s)$ and $\neg F(x, do(A(y), s))$ hold. This means that F is true in the situation s and the execution of action A makes it false. This is possible if and only if $(\epsilon^-)_F(x,y,s)$ is true. Thus

$$F(x,s) \wedge \neg F(x, do(A(y), s)) \supset (\epsilon^-)_F(x,y,s)$$

This is equivalent with

$$F(x,s) \wedge \neg(\epsilon^-)_F(x,y,s) \supset F(x, do(A(y), s))$$

Similarly

$$\neg F(x,s) \wedge \neg(\epsilon^+)_F(x,y,s) \supset \neg F(x, do(A(y), s))$$

This solution to the frame problem has the disadvantage that it requires the definition of $2 \times A \times F$ frame axioms.

The other solution that has been proposed by Hass [45], is based on a similar idea. Consider the example with the robot and assume that $holding(r,x,s)$ and $\neg holding(r,x,do(a,s))$ hold. This means that the robot r has left on the floor the object x or it has dropped it. Thus,

$$\begin{aligned}&holding(r,x,s) \wedge \neg holding(r,x,do(a,s))\\&\supset a = putDown(r,x) \vee a = drop(r,x)\,.\end{aligned}$$

The important thing to note is that the above proposition is universally quantified on a. The frame axiom is

$$\begin{aligned}&holding(r,x,s) \wedge a \neq putDown(r,x) \wedge a \neq drop(r,x)\\&\supset holding(r,x,do(a,s))\,.\end{aligned}$$

This means that all the actions except from $putDown(r,x)$ and $drop(r,x)$ do not influence fluent *holding*. The frame axioms have one of the following two forms.

$$\begin{aligned}&F(x,s) \wedge \neg F(x,do(a,s)) \supset \alpha_F(x,a,s)\\&\neg F(x,s) \wedge F(x,do(a,s)) \supset \beta_F(x,a,s)\,.\end{aligned}$$

In the above frame axioms, the action a is universally quantified. This means that if it changes value of a fluent F, then α_f or β_f provides explanation for this change. The number of axioms needed in order to describe all the likely changes in the truth value of fluents is $2 \times F$.

2.1.5 The solution of Reiter

Reiter proposed a simple solution [107] to the frame problem. We present it with an example. Consider a robot r that holds an object x and a new fluent *bomb*(b) (b is a bomb). Assume that the following two positive frame axioms hold

$$\begin{aligned}&fragile(x,s) \supset broken(x,do(drop(r,x),s))\\&nextTo(b,x,s) \supset broken(x,explode(b),s))\,,\end{aligned}$$

The first axiom states that the object x that the robot holds in situation s will be broken in the next situation if the robot drops it. The second axiom states that if the object x is next to a bomb in the situation s, then in the next situation it will break if the bomb explodes. The above axioms can rewritten as follows

$$\begin{aligned}&[(\exists r)\{a = drop(r,x) \wedge fragile(x,s)\}\\&\vee(\exists b)\{a = explode(b) \wedge nextTo(b,x,s)\}]\\&\supset broken(x,do(a,s))\,.\end{aligned}$$

The above axioms mention two actions which if executed will result in then the fluent *broken* being true. In order for the execution of those actions to be possible, some preconditions must hold. Thus the above axioms must be rewritten as follows:

$$\begin{aligned}&Poss(a,s) \wedge [(\exists(r,x))\{a = drop(r,x) \wedge fragile(x,s)\}\\&\vee(\exists b)\{a = explode(b) \wedge nextTo(b,x,s)\}]\\&\supset broken(x, do(a,s))\,.\end{aligned}$$

Finally, the positive axiom takes the form

$$Poss(a,s) \wedge (\gamma^{+})_{broken}(a,s) \supset broken(x, do(a,s))\,.$$

In a similar way we can take the negative frame axiom

$$\begin{aligned}&Poss(a,s) \wedge [(\exists(r,x))a = repair(r,x)]\\&\supset \neg broken(x, do(a,s))\,.\end{aligned}$$

and rewrite it as:

$$Poss(a,s) \wedge (\gamma^{-})_{broken}(a,s) \supset \neg broken(x, do(a,s))\,.$$

This form can be generated from axioms of the form:

$$\begin{aligned}&Poss(a,s) \wedge (\gamma^{+})_F(a,s) \supset F(do(a,s))\\&Poss(a,s) \wedge (\gamma^{-})_F(a,s) \supset \neg F(do(a,s))\,.\end{aligned}$$

Consequently for each fluent F, we derive an axiom of the form

$$Poss(a,s) \supset [F(do(a,s)) \equiv (\gamma^{+})_F(a,s) \vee F(s) \wedge \neg(\gamma^{-})_F(a,s)\,.$$

The above axiom means that fluent F is true in a situation (which resulted from the execution of action a in a situation s), only when the conditions $(\gamma^{+})_F(a,s)$ are true (thus after the execution of action a, the fluent F becomes true) or the fluent F is true in the situation s and the conditions, which make it false after the execution of action a are false. (this mean that the $(\gamma^{-})_r(a,s)$ is false).

This solution is very simple because it only requires $F + A$ axioms (one axiom for each fluent and one for each action).

2.2 The Ramification Problem

The ramification problem is a very hard problem that arise in robotics, software engineering and databases. We introduce this problem by mean of examples. Suppose we are interested in maintaining a database that describes a simple circuit (figure 1), which has two switches and one lamp. The circuit's behavior is described by the following integrity constraints.

$$up(s_1) \wedge up(s_2) \equiv light \quad (1)$$
$$\neg up(s_1) \supset \neg light \quad (2)$$
$$\neg up(s_2) \supset \neg light \quad (3)$$

The first constraint means that when the two switches are up the lamp must be lit, while the second and third constraints mean that if a switch is not up then lamp must not be lit.

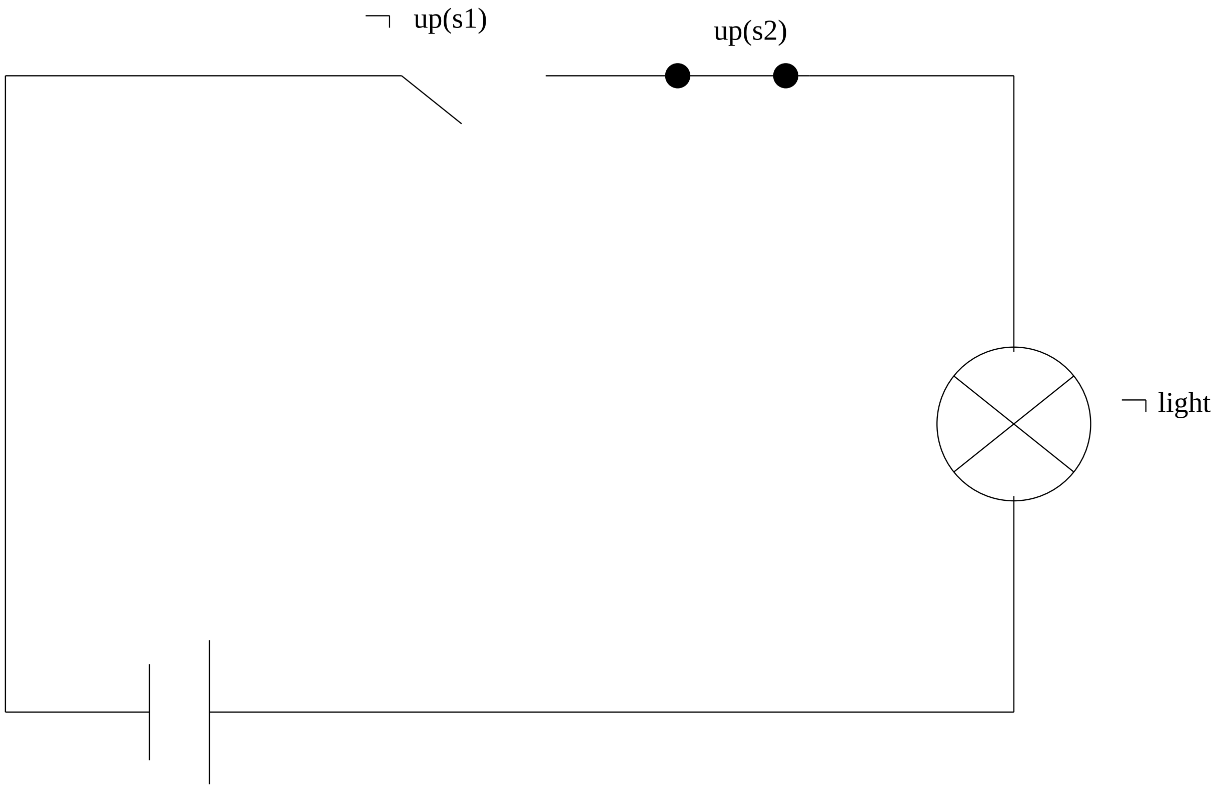

Figure 1: Simple electric circuit

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

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

$$toggle\_switch(s) \supset \neg up(s) \quad if \quad up(s)\,.$$

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 as direct 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 the indirect effects exists 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 [81].

2.2.1 Minimal Change Approach

The simplest of the techniques suggested in the literature is the *minimal-change* approach [131]. It suggests that, when an action occurs in a situation S, one needs to find the consistent situation S' which has the fewer changes from the situation S. S' is that situation that is closer to S than any other situation. The formal definition of the minimal change successor is:

Definition 2.1 *Consider three situations, S, S', S''. Then S' is said to be close to S than S'' if and only if $S' \setminus S \not\subseteq S'' \setminus S$. The notation that is used is $S' \preceq_S S''$.*

After the execution of an action the next situation is minimal the change successor.

Consider the example that was presented above and assume that the situation of the circuit is $\{\neg up(s_1),\ up(s_2), \neg light\}$. The action $toggle - switch(s_1)$ has as effects the situation $S' = \{up(s_1), up(s_2), \neg light\}$. This situation is inconsistent because it violates the first integrity constraint. The two situations which are consistent are: $S_1 = \{up(s_1), up(s_2), light\}$ and $S_2 = \{up(s_1), \neg up(s_2), \neg light\}$. We have that: $S_1 \setminus S' = \{light\}$, $S_2 \setminus S' = \{\neg up(s_2)\}$. Thus, none of the following two propositions hold: $S_1 \preceq_{S'} S_2$ or $S_2 \preceq_{S'} S_1$. Consequently, there exist two minimal-change successors.

As we can observe, when the action $toggle - switch(s_1)$ takes place, there are two possible indirect effects. One is to turn on the light and the other to toggle down the switch s_2. According to the minimal change approach the two effects are similar and there is no way to separate them. It is more reasonable to turn on the lamp than to toggle down switch s_2. More knowledge should be available in order to discriminate between the two possible situations.

2.2.2 Categorizing Fluents

As it was reported in the previous section it is necessary the employ more knowledge in order to be able to discriminate between the indirect effects of action $toggle - switch$.

In the above example the desirable indirect result is to turn on the lamp and not to toggle down the switch. In order to select this, we must be able to discriminate between fluents *up* and *light*. One way to do this [62, 63, 64] is to seperate fluents into those whose truth value change as direct effect of an action and those whose truth value as indirect effect. The former are named *primary* fluents and the latter *secondary* fluents. In the above example primary fluents are $F_p = \{up(x) : x \in \{s1, s2\}\}$ and secondary are $F_s = \{light\}$. Now a minimal change successor situation is defined as

Definition 2.2 *A situation S' is the minimal change successor of situation S if and only if for every other situation S''*
$\|S' \setminus S\| \cap F_p \not\subseteq \|S'' \setminus S\| \cap F_p$
or
$\|S' \setminus S\| \cap F_p = \|S'' \setminus S\| \cap F_p$ *and*
$\|S' \setminus S\| \cap F_p \not\subseteq \|S'' \setminus S\| \cap F_s$

The symbol $\| \ \|$ is the number of elements of a set that contains only positive elements (e.g. $\|\{f, \neg f\}\| = \|\{f\}\|$). With the above definition, the situations that has the fewer changes in primary fluents is selected. [1].

In the above example the situation S_1 will be selected as minimal change successor and not S_2 because $\|S_1 \setminus S\| \cap F_p = \{\}$ while $\|S_2 \setminus S\| \cap F_p = \{up(s2)\}$. S_1 does not contain indirect effects in primary fluents. S_2 contains $up(s_2)$.

However, it is not always feasible to separate fluents in primary and secondary. Many times it is possible to have fluents that can be primary for some actions and secondary for others. In this case the categorization of fluents cannot work satisfactorily.

For example consider the circuit which appears in figure 7. There are three switches s_1, s_2 and s_3. Fluents are $F = \{up, light, relay\}$. The rules for the execution of actions are:

$$\begin{aligned} toggle - switch(x) &\supset \{up(x)\} \ change \ to \ \{\neg up(x)\} \\ toggle - switch(x) &\supset \{\neg up(x)\} \ change \ to \ \{up(x)\} \end{aligned}$$

These rules mean that if a switch is closed(open) and action toggle-switch is performed then the switch will open(close, resp). The integrity constraints are

[1]The situations that contain indirect effects which do not change the truth value many primary fluents.

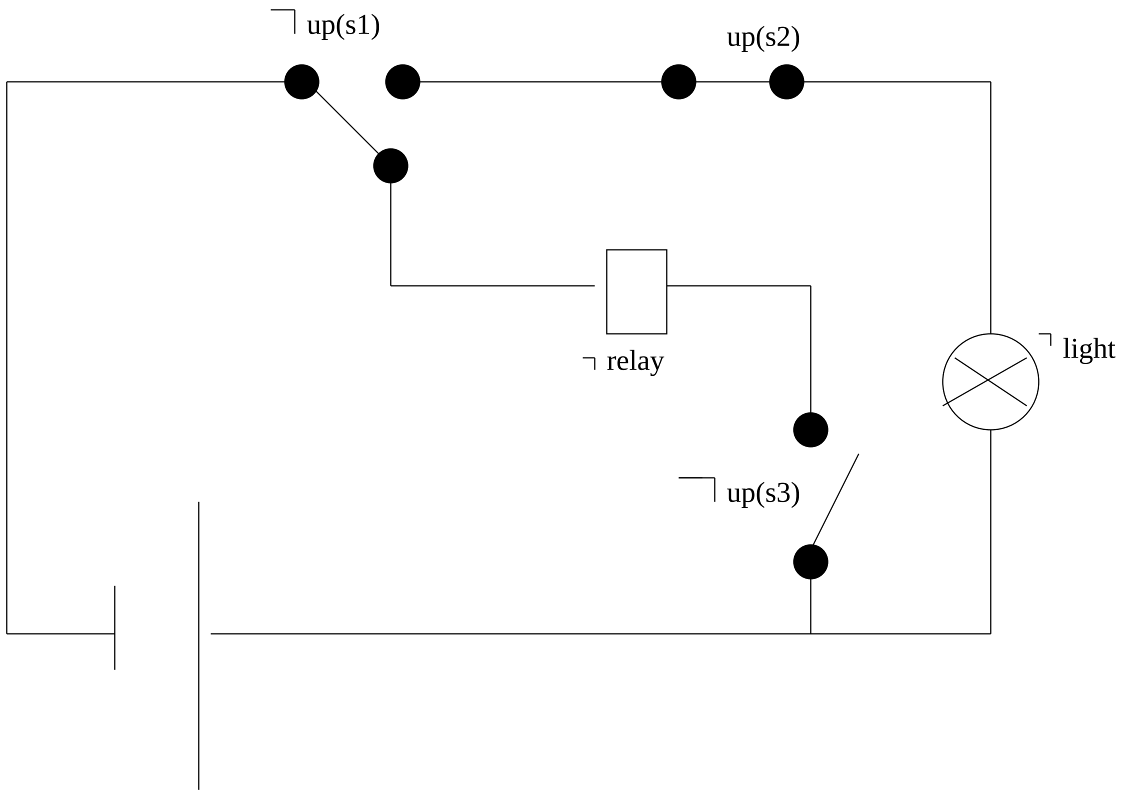

Figure 2: Complex Circuit

$$light \equiv up(s_1) \wedge up(s_2)$$
$$relay \equiv \neg up(s_1) \wedge up(s_3)$$
$$relay \supset \neg up(s_2)$$

The fluents $up(s_1), up(s_3)$ are primary. The fluents *light* and *relay* are secondary. The problem exists with fluent $up(s_2)$. On the one hand it is secondary because when *relay* is true, it changes the situation of switch s_2 from $up(s_2)$ in $\neg up(s_2)$. On the other, it is primary because it can change its situation as the direct effect of action $toggle\_switch(s_2)$.

If it is considered as primary then if the situation of the circuit is $S = \{\neg up(s1), up(s2), \neg up(s3), \neg light, \neg relay\}$ then the action $toggle\_switch(s_3)$ will have as result the situation $S' = \{\neg up(s1), up(s2), up(s3), \neg light, \neg relay\}$. This situation is not consistent. Hence it must be changed. There exist three likely new situations

$$S_1 = \{\neg up(s1), \neg up(s2), up(s3), \neg light, relay\}$$

$$S_2 = \{up(s1), up(s2), up(s3), light, \neg relay\}$$
$$S_3 = \{up(s1), \neg up(s2), up(s3), \neg light, \neg relay\}$$

We have

$$\|S_1 \setminus S'\| \cap F_p = \{up(s_2), relay\} \cap F_p = \{up(s_2)\}$$
$$\|S_2 \setminus S'\| \cap F_p = \{up(s_1), light\} \cap F_p = \{up(s_1)\}$$
$$\|S_3 \setminus S'\| \cap F_p = \{up(s_1), up(s_2)\} \cap F_p = \{up(s_1), up(s_2)\}$$

As we can observe

$$S_1 <_{S'} S_3$$
$$S_2 <_{S'} S_3$$
$$S_1 =_{S'} S_2 \quad (for \quad F_p)\,.$$

For the secondary fluent we have

$$\|S_1 \setminus S'\| \cap F_s = \{up(s_2), relay\} \cap F_s = \{relay\}$$
$$\|S_2 \setminus S'\| \cap F_s = \{up(s_1), light\} \cap F_s = \{light\}$$

Thus

$$S_1 =_{S'} S_2 \quad (for \quad F_p)\,.$$

In this case there are two minimal change successors S_1 and S_2.

If $up(s_2)$ is considered as secondary then there also are two minimal change successors.

As it appears the double role of $up(s2)$ has as result the failure of the above solution (categorization) to produce a unique solution. A solution is to create a third category of fluents, tertiary fluents that will have still smaller priority than secondary. In that case, the fluent $up(s_2)$ belongs to the new category. Now the following holds

$$S_1 <_{S'} S_2 \quad (for \quad F_p)\,.$$

In complicated systems it is very difficult to separate of the fluents in these three categories because a large number of dependences between direct and indirect effects must be examined.

2.2.3 Causal Relationalships

The production of more knowledge in order to enable the categorization of fluents is very complicated and difficult. Whether it changes or not a fluent as consequence of the execution of an action, it depends on the content of database. For example the action $toggle - switch(s_1)$ has as indirect effect the light if and only if the content of database before the execution is $\neg up(s_1), up(s_2)$. In any other case the action $toggle - switch(s_2)$ has not as indirect effect to light the lamp (e.g. $\neg up(s_2)$ holds). We need a frame which will allows the production of the indirect effects based on the content of database. This means that the new frame must encapsulate the dependencing that exists between indirect effects of an action and the content of database.

Causal relationships [21, 82, 43, 124, 125, 27, 29, 30, 31, 37, 38, 40, 41, 44, 54, 60, 65, 66, 67, 117, 123, 135] encapsulate the dependences that exists between the content of databese and the indirect effects of an action. As we observe from the above example a causal relationship has the form

$$A \ \ causes \ \ B \ \ if \ \ C$$

Causal relationships are constituted by two parts. The first part (C) (which is named context) is a fluent formula). If the fluent formula is true then could executed the causal relationship. The second part is a simple result (an atomic fluent) which is the indirect effect.

For example consider the circuit that depicted in figure 1. Then

$$up(s_1) \ \ causes \ \ light \ \ if \ \ up(s_2)$$

In this example "if up (s2)" is the condition (first part of causal relationship) which when is true then the effect $up(S_1)$ has as indirect effect $light$ (second part the causal relationship).

Causal relationship can produce indirect effects after the execution of an action and after the production of the direct effects (from the application of concrete rules). Such a relation functions always on a pair (S,E), where S is a concrete situation and E are the all direct and indirect results an action.

As we may observe, there exists discrimination between a context and particular effect. The necessity of this discrimination becomes apparent in the following example. Consider circuit of figure 3. In this example, there are the followings causal relationships.

$$up(s1) \ \ causes \ \ up(s2) \ \ if \ \ \top$$

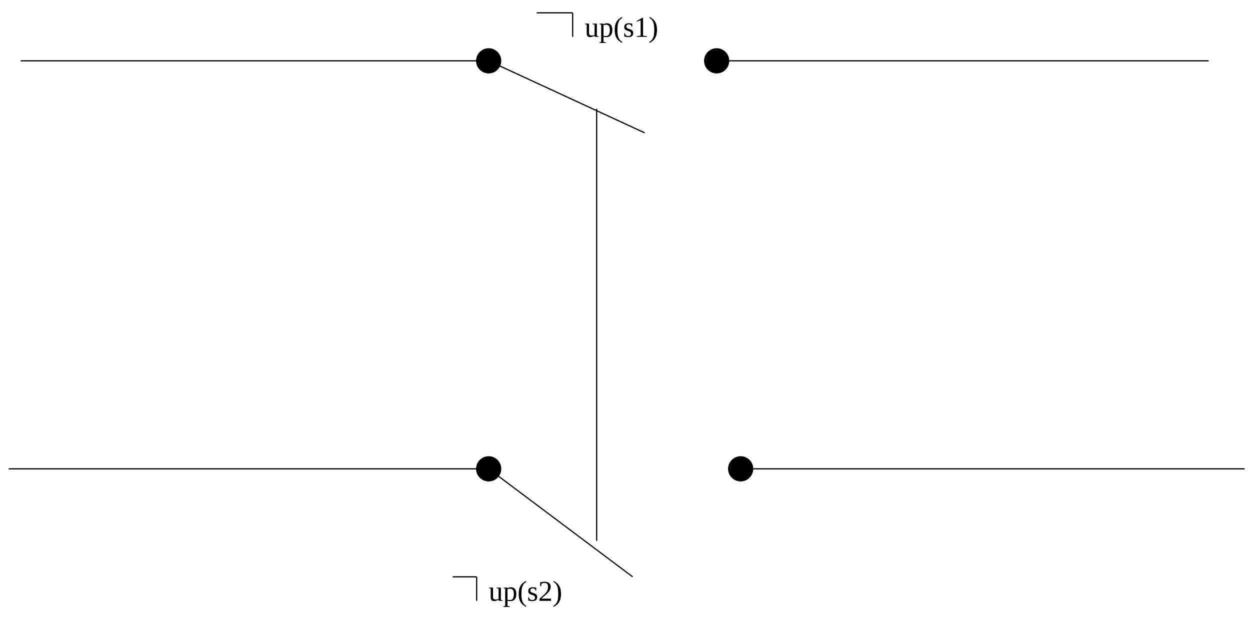

Figure 3: causal relationship

$$
\begin{aligned}
&up(s2) \;\; causes \;\; up(s1) \;\; if \;\; \top \\
&\neg up(s1) \;\; causes \;\; \neg up(s2) \;\; if \;\; \top \\
&\neg up(s2) \;\; causes \;\; \neg up(s1) \;\; if \;\; \top
\end{aligned}
$$

Assume that the circuit is in the situation $\{\neg up(s1), \neg up(s2)\}$ when the action $toggle\_switch(s_1)$ takes place. The direct effect will be $up(s_1)$ and the new situation will be $S' = \{up(s_1), \neg up(s_2)\}$. Now the set E is $E = \{up(s_1)\}$. Applying the first rule the final situation will be $S'' = \{up(s_1), up(s_2)\}$ and the effects will be $E = \{up(s_1), up(s_2)\}$.

Assume that the circuit is in the situation $\{up(s1), up(s2)\}$ and that the action $toggle\_switch(s_2)$ takes place. The direct result will be $\neg up(s2)$ and the new situation is $S' = \{up(s1), \neg up(s2)\}$ and the set of effects is $E = \{\neg up(s2)\}$. The situation S' is the same in both cases. However the triggering effects E are different. Finally, in the second case, the situation is $S'' = \{\neg up(s1), \neg up(s2)\}$ and set of triggering events is $E = \{\neg up(s1), \neg up(s2)\}$.

From the above examples it appears that it is necessary to seperate context from triggering effects. If they are not separated then it is possible to have a change which would reverse some previous effects (direct or indirect) and finally cancel an action.

A causal relationship r has the form

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

where ϕ is a formula, ϵ is triggering event and ρ is ramification(indirect) effect. This rule means that if in a situation S the fluent formula Φ is true then if the effect ϵ is true then the fluent ρ must be true.

The causal relationship r is applicable in a situation S and in a set of effects E if and only if $\epsilon \in E$ and the $\Phi \wedge \neg\rho$ are true in S. This means that ϵ belongs in E, hence can cause ρ to become true because the fluent formula Φ is true in S (this means that $\Phi \wedge \neg\rho$ holds, thus Φ is true). In this case the execution of the causal relationship r in (S, E) leads to (S', E') where $S' = (S \backslash \{\neg\rho\} \cup \{\rho\})$ and $E' = (E \backslash \{\neg\rho\} \cup \{\rho\})$. Notice that if ρ is true then the execution of the rule does not leads to a new situation.

If with the application of a set of causal relationships R from (S, E) the world is led to (S', E'), then this is symbolized by $(S, E) \leadsto_R (S', E')$. An action could have a lot of direct and indirect effects. These in turn can cause a lot of other indirect effects. The process $(S, E) \leadsto_R (S', E')$ is continued until the world is in a consistent situation.

Causal relationships are applied without a pre-determined sequence. For example, if the following causal relationships are definied,

$$\begin{array}{lllll} e_1 & causes & \rho_1, & if & \Phi_1 \\ e_2 & causes & \rho_2, & if & \Phi_2 \\ e_3 & causes & \rho_3, & if & \Phi_3 \end{array}$$

then if the (S, E) holds where

$$\begin{array}{l} S = \{\Phi_1, \Phi_2, \Phi_3, \neg\rho_1, \neg\rho_2, \neg\rho_3\} \\ E = \{e_1, e_2, e_3\} \end{array}$$

we have

$$\begin{array}{l} (S, E) \leadsto_R (S', E') \\ where \\ S' = \{\rho_1, \rho_2, \rho_3, ...\} \\ E = \{\rho_1, \rho_2, \rho_3, ...\} \end{array}$$

As we observe in the above example all causal relationships are applicable. If execute all three causal relationships then the final situation is S'.

This means that if we execute the same causal relationships in different order, then the final situation will be the same. Theilsher [21] has proved the above result.

However this does not mean when changes the order of execution (causal relationships) will be executed the same causal relationships. For example if the causal relationship r_1 executes before r_2 then perhaps the execution of r_2 will not be possible in the new situation. Consider the above example with some changes

$$\begin{array}{llll} e_1 & causes & \rho_1, & if \;\; \Phi_1 \\ e_2 & causes & \rho_2, & if \;\; \neg\rho_1 \\ e_3 & causes & \rho_3, & if \;\; \Phi_3 \end{array}$$

Assume that (S, E) holds where

$$S = \{\Phi_1, \Phi_3, \neg\rho_1, \neg\rho_2, \neg\rho_3\}$$
$$E = \{e_1, e_2, e_3\}$$

As we observe in the situation S are applicable the first and the third causal relationships. If we execute first the $e_1 \quad causes \quad \rho_1, \quad if \quad \Phi_1$ then the second causal relationship is not applicable in the new situation because ρ_1 holds. In that case the final situation is

$$S' = \{\Phi_1, \Phi_3, \rho_1, \neg\rho_2, \rho_3\}$$

If we executed first the second causal relationship then we could execute the other two. In that case the final situation is

$$S' = \{\Phi_1, \Phi_3, \rho_1, \rho_2, \rho_3\}$$

This means that perhaps there are two or more consistent situations as final situations. Which of these we finally produce is dependent on the order of the execution of the causal relationships.

It is obvious that there is need for some mechanism which will allow the execution of causal relationship in a deterministic order. For this reason, a binary relation I (influence relation) is used. If $(f_1, f_2) \in I$, this means that if the truth value of f_1 changes then this could change also the truth value of f_2. This relation suggests a deterministic order for the execution of the causal relationships. The relation I provides a global categorization of fluents. In the above example with the simple circuit we have that

$$\begin{aligned}
&(up(s_1), light) \in I\\
&(up(s_2), light) \in I\\
&(up(s_1), up(s_2)) \notin I\\
&(up(s_2), up(s_1)) \notin I
\end{aligned}$$

Thus when a switch is up the lamp could be lit but it cannot toggle down another switch. For the creation of the causal relationships many algorithms have been proposed which take into account the relation I. The result is the creation of causal relationships that do not cause undesirable results.

Now we present some other solutions which based in the causal relationships.

2.2.4 The solution of McCain and Turner

The method proposed by in McCain and Turner's in [82] uses a standard language of propositional logic, based on a fixed set of atoms. A situation for the language is represented by a maximal consistent set of literals (fluents).

Background knowledge is given in the form of state constraints and causal laws. A standard example of a state constraint is $eating \rightarrow Alive$ (standard implication). McCain and Turner propose causal rules of the form

$$\psi \Rightarrow \phi\,,$$

where ψ and ϕ are fluent formulas. Informally, this rule expresses a relation of determinism between the states of affairs that make ψ and ϕ true. To begin with, the causal laws are treated as inference rules; later they will be recast to rules called s-conditionals.

The standard derivability relation $\vdash$ of propositional logic is extended to take into account the inference rules. Let Γ be a set of formulas and C be a set of inference rules. Γ is said to be closed under C if for every rule $\psi \Rightarrow \phi \in C$, if $\psi \in \Gamma$ then $\phi \in \Gamma$. For any formula ψ,

$$\Gamma \vdash_C \psi$$

means that ψ belongs to the smallest set of formulas containing Γ that is closed closed under C.

The standard framework in which the ramification problem is addressed is one in which background knowledge is given in the form of state constraints. For this framework, the problem was specified by Winslett [131] using the following definition.

Definition 2.3 *For any situation S, any direct effect E, and any set B of formula constraints, $Res_W^B(E,S)$ is the set of situations S' such that*

1. *S' satisfies $E \cup B$*
2. *No other situation that satisfies $E \cup B$ differs from S on fewer atoms, where "fewer" is defined in terms of set inclusion.*

For example assume the following:

$$\begin{aligned} S &= \{Alive, Eat\} \\ E &= \{\neg Alive\} \\ B &= \{Eating \supset Alive\} \end{aligned}$$

Thus

$$Res_B^W(E,S) = \{\{\neg Alive, \neg eating\}\}$$

The ramification is $\neg eating$. The above definition is right but it does not describe "exactly" how we could produce the ramifications. Assume the example with the simple circuit

$$\begin{aligned} S &= \{up(s_1), \neg up(s_2), \neg light\} \\ E &= \{up(s_2)\} \\ B &= \{up(s_1) \wedge up(s_2) \supset light, \neg up(s_1) \supset \neg light\} \end{aligned}$$

Then,

$$Res_B^W(E,S) = \{\{up(s_1), up(s_2), light\}, \{\neg up(s_1), up(s_2), \neg light\}\}$$

As we observe we have two possible situations as in the case of the minimal change approach.

McCain and Turner [82] propose the following reformulation of the above definition, in order to take inference rules into account.

Definition 2.4 *For any situation S, any direct effect E, and any set C of inference rules, $Res_C(E,S)$ is the set of situations S' such that $S' = \{L : (S \cap S') \cup E \vdash_C L\}$.*

Consider the first example of the people who eat.

$$\begin{aligned} S &= \{eat, alive\} \\ E &= \{\neg alive\} \\ C &= \{\neg alive \Rightarrow \neg eat\} \end{aligned}$$

Then

$$Res_C(E, S) = \{\{\neg alive, \neg eat\}\}$$

Consider the example with the simple circuit

$$\begin{aligned} S &= \{up(s_1), \neg up(s_2), \neg light\} \\ E &= \{up(s_2)\} \\ B &= \{up(s_1) \wedge up(s_2) \supset light, \neg up(s_1) \supset \neg light\} \end{aligned}$$

Thus

$$Res_C(E, S) = \{\{up(s_1), up(s_2), light\}\}$$

Notice that

$$\{\neg up(s_1), up(s_2), \neg light\} \notin Res_C(E, S)$$

because there is no rule such that $\neg light \supset \neg up(s_1)$. Thus the new definition is more deterministic because we have only one possible situation while in the previous solution we have two possible situations. Also

$$Res_C(E, S) \subseteq Res_B^W(E, S)$$

The trouble with the two previous definitions is that they cannot be recast in semantic terms by replacing the derivability relation $\vdash$ with $\models$. To remedy this a conditional logic C_{flat} is defined. C_{flat} is an extension of S5 modal logic.

The vocabulary of the C_{flat} language consists of a fixed set of atoms (fluents). The formulas of the language are formed from its atoms and expressions of the form $(\psi \Rightarrow \phi)$. Here expressions of the above form are called s-conditionals, and they may be read as: "the truth of ψ determines the truth of ϕ".

The second formalism by McCain and Turner [83] is inspired by the work of Pearl [92, 93]. The underlying propositional signature is described by three pairwise-disjoint sets: A set of action names, a nonempty set of fluent names, and a nonempty set of time names (corresponding to a segment of the integers). The atoms of the language are expressions of the forms a_t and f_t, where a, f , and t are action, fluent, and time names, respectively. Intuitively, is true only if action a occurs at time t, and f_t is true only if the fluent f holds at time t. A fluent formula is a propositional combination of fluent names.

A causal law is an expression of the form $\Psi \Rightarrow \Phi$, where Ψ and Φ are formulas of the underlying propositional language. Ψ is called the antecedent and Φ is called the consequent of the causal law. A causal law $\Psi \supset \Phi$ can mean both

- the fact that Ψ causes the fact that Φ, and
- necessarily, if Ψ then the fact that Φ is caused.

A causal theory is a set of causal laws. An situation S for a propositional language is identified with the set of literals L such that $S \models L$. For every causal theory D and situation S, let

$$D^S = \{\Psi \| for \;\; some \;\; \Phi, \; \Phi \rightarrow \Psi \;\; and \;\; S \models \Phi\}.$$

A situation S is causally explained according to D iff $S = \{L : D^S \models L\}$, where L is understood to stand exclusively for literals.

For example consider the circuit of the figure 1. Then, there are the following causal laws.

$$\begin{aligned}
&toggle\_switch(s_1) \wedge \neg up(s_1) \supset up(s_1)\\
&toggle\_switch(s_1) \wedge up(s_1) \supset \neg up(s_1)\\
&toggle\_switch(s_2) \wedge \neg up(s_2) \supset up(s_2)\\
&toggle\_switch(s_2) \wedge \neg up(s_2) \supset \neg up(s_2)\\
&up(1) \wedge up(s_2) \supset light\\
&\neg up(s_1) \supset \neg light\\
&\neg up(s_2) \neg light
\end{aligned}$$

As we can observe the four first laws are dynamic because they are evaluated when the corresponding action takes place. The rest are static because they are executed every time. The dynamic laws encapsulate the direct effects of an actions , while the static ones encapsulate the indirect effects of an action. Consider the situation

$$S = \{\neg up(s_1), up(s_2), \neg light\}$$

Assume that the action *toggle_switch*(s_1) take place. Then the first rule must be evaluated. The new situation is

$$S = \{up(s_1), up(s_2), \neg light\}$$

Now we must evaluate the static law. $up(1) \wedge up(s_2) \supset light$. The final situation is

$$S = \{up(s_1), up(s_2), light\}$$

An interesting feature of this approach that the timeline represented by adding a subscript to the fluents allows delayed effects, although the authors mention this only in passing. For example is possible an action which executed at time point t to have effects at time point $t + y$. This is very important in the case of temporal databases as we explain in the following chapter. The author use the causal relationship only for producing the next situation (in the next time point). This works well in the static world as the world of the circuit, in which nothing changes until an action take place. As we explain in the Chapter 3 this is very strict in dynamic worlds.

Also, it is necessary that an automatic way for the production of these causal laws. In Chapter 4, we propose an algorithm for the production of the causal laws.

2.2.5 Lin's Solution

Another proposal using explicit causality was presented by Lin [68]. Using a Caused predicate, directionality of causation can be encoded within the situation calculus framework. Lin later shows how this approach can be used to deal with nondeterminism.

Time progresses through the execution of actions, that is successive applications of the *do* function, leading to a branching time-structure.

In addition to the standard situation calculus notation a ternary predicate Caused is introduced.

Definition 2.5 *$Caused(f, true, S)$ denotes that the fluent f is caused to be true in situation S.*

From the above definition we can produce the following corollary

Corollary 2.1 *The following always hold.*

$$\begin{array}{l} Caused(f, true, S) \rightarrow Holds(f, S) \\ Caused(f, false, S) \rightarrow \neg Holds(f, S) \end{array}$$

Lin proposes the following frame axiom

$$\begin{array}{l} Poss(a, S) \rightarrow ((\exists v) Caused(f, v, do(a, s)) \rightarrow \\ {[Holds(f, do(a, S)) \leftrightarrow Holds(f, S)]}) \end{array}$$

From this the pseudo successor state axiom can be derived:

$$\begin{array}{l} Poss(a, S) \rightarrow \{Holds(f, do(a, S)) \leftrightarrow \\ \qquad Caused(f, true, do(a, S)) \vee \\ \qquad Holds(f, S) \wedge \neg Caused(f, false, do(a, S))\}\,. \end{array}$$

A formula $\Phi(S)$ is called a simple state formula about S if Φ does not mention $Poss, Caused$ or any situation term other than S.

Lin gives the following definitions for the direct and indirect effects of an action and for the preconditions of an action

Definition 2.6 *For each action $A(\vec{x})$, the direct effects are described by the following axioms*

$$\begin{array}{l} Poss(A(\vec{x}), S) \rightarrow \\ \qquad [\Phi(S) \rightarrow Caused(F(\vec{y}), v,\ do(A(\vec{x}), S))]\,, \end{array}$$

where F is a fluent, $\Phi(S)$ is a simple state formula about S and $v \in \{true, false\}$.

Definition 2.7 *Formalize all causal rules(which include the indirect effects) by axioms of the form:*

$$\Phi(S) \wedge Caused(f_1, v_1, S) \wedge ... \wedge\ Caused(f_n, v_n, S) \rightarrow\ Caused(F(\vec{x}), v, S)\,,$$

where F is a fluent, and $\Phi(S)$ is a simple state formula about S.

For example consider the simple circuit of figure 1. Then the direct effects are determined by the following rules

$$\begin{aligned}
&Poss(toggle - switch(x), S) \supset \\
&up(x, S) \supset Caused(up(x), false, do(toggle - switch(x), S)), \\
&Poss(toggle - switch(x), S) \supset \\
&\neg up(x, S) \supset Caused(up(x), true, do(toggle - switch(x), S)),
\end{aligned}$$

The precondition of the action $toggle - switch(x)$ is the object x to be a switch.

$$Poss(toggle - switch(x)) \supset switch(x)$$

The indirect effects are encapsulated by the following rules

$$\begin{aligned}
&Caused(up(s_1), true, do(a, S)) \wedge Caused(up(s_2), true, do(a, S)) \supset Caused(light, true, do(a, S)), \\
&Caused(\neg up(s_1), true, do(a, S)) \supset Caused(\neg light, true, do(a, S)), \\
&Caused(\neg up(s_2), true, do(a, S)) \supset Caused(\neg light, true, do(a, S))
\end{aligned}$$

Lin formalized all other domain knowledge by axioms of the form $\forall_S C(S)$, where $C(S)$ is a simple state formula about S. All the above give us a starting theory T.

This starting theory may contain several unnecessary rules. For example

$$\begin{aligned}
&Caused(\neg up(s_1), true, do(a, S)) \wedge Caused(\neg up(s_2), true, do(a, S)) \supset Caused(\neg light, true, do(a, S)), \\
&Caused(\neg up(s_1), true, do(a, S)) \supset Caused(\neg light, true, do(a, S)), \\
&Caused(\neg up(s_2), true, do(a, S)) \supset Caused(\neg light, true, do(a, S))
\end{aligned}$$

As we can observe the first rule overlaps with the other two. Thus we must eliminate them. The procedure of the elimination of the extra rules is the minimization of the starting theory T.

Also, the starting theory may contain many different rules about the preconditions of one action. For example, assume the fluents f, f_1, f_2, f_3 and the action A

$$\begin{aligned}
&Poss(A) \equiv f \wedge f_1 \\
&Poss(A) \equiv f_2 \wedge f_3
\end{aligned}$$

As we can observe, in order for the execution of the action A to be possible, the following must hold

$$f \wedge f_1 \wedge f_2 \wedge f_3$$

Thus the above two propositions are replaced by

$$Poss(A) \equiv f \wedge f_1 \wedge f_2 \wedge f_3$$

This procedure is called maximization of $Poss$.

The following algorithm addresses as the ramification problem

1. Start with a theory T that includes all effect axioms and state constraints.
2. Let $T' = \{Minimize\ \ Caused\ \ in\ \ \mathrm{T}\}$
3. Circumscribe Caused in T with all other predicates fixed.
4. Maximize Poss in T' to obtain the final action theory.
5. Replace the occurrences of Caused in the pseudo successor state axiom to get the following form $Caused(F(\vec{x}), v, S) \leftrightarrow \Psi$, where Ψ is a formula not containing the predicate Caused.
6. For each action A, maximize the relation $Poss(A(\vec{x}), S)$ with $Holds(f, S)$ fixed but Caused and $Holds(f, S') \wedge S' \neq S$ allowed to vary.

Consider the example where shooting someone results in that he stops walking.

$$\begin{aligned}
&Poss(start - walk, S) \rightarrow Caused(walking, true, do(start - walk, S)),\\
&Poss(end - walk, S) \rightarrow Caused(walking, false, do(end - walk, S)),\\
&Poss(shoot, S) \rightarrow Caused(dead, false, do(shoot, s)),\\
&Poss(start - walk, S) \rightarrow walking(S),\\
&Poss(end - walk, S) \rightarrow walking(S),\\
&dead(S) \rightarrow Caused(walking, false, S)\,.
\end{aligned}$$

Lin allows preconditions (Φ) and triggers in the form of conjunctions. The postcondition is limited to a literal. This means that his dependency laws cannot model nondeterminism in its current form. Nondeterminism of actions, however, is handled [69].

One interesting aspect of this method is that it is the first general approach to the ramification problem that does not use a fixpoint computation to determine the resulting states.

2.2.6 Sandewall's Solution

A method for addressing the ramification problem is proposed by Sandewall in [118]. The idea is that different approaches should be assessed with respect to an underlying semantics which formally defines the intended conclusions for a class of scenario descriptions. The purpose of the underlying semantics is to define the set of intended models, and thereby the set of intended conclusions, in a precise, simple and intuitively convincing fashion.

The method for handling ramifications that serves as the basis on which other approaches are judged has much in common with Thielscher's method [123] in the sense that it is basically a fixpoint approach. The details of the causal chains, however, are captured in the function N , there by avoiding the need for a fixpoint computation.

Let R be the set of possible states in the world, formed as the Cartesian product of the finite range sets of a finite number of state variables(fluents). Also, let **E** be the set of possible actions, and let the main next-state function $N(E, r)$ be a function from $E \times R$ to non-empty subsets of R. The function N is intended to indicate the set of possible result states if the action E is performed when the world is in state r. The assumption $N(E, r) \neq \emptyset$ expresses that every action E can always be executed in every starting state r.

A binary non-reflexive causal translation relation C between states is introduced; if $C(r, r')$ then r' is said to be a successor of r. A state r is said to be stable iff it does not have any successor. The set R_c of admitted states is chosen as a subset of R all of whose members are stable with respect to C.

Furthermore, an action invocation relation $G(E, r, r')$ is a relation where $E \in \mathbf{E}$ is an action, r is the state where the action E is invoked, and r' is the new state where the instrumental part of the action has been executed. For every E and r there must be at least one r' such that $G(E, r, r')$, that is, every action is always invocable.

An action system is a tuple $\langle R, E, C, R_c, Gi \rangle$. For any state $r \in R_c$, consider a state r_1 such that $G(E, r, r_1)$, and a sequence of states $r_1, r2, ..., r_k$ where $C(r_i, r_{i+1})$ for every i between 1 and $k - 1$, and $r_k \in R_c$ is a stable state. Such a sequence is called a transition chain, and r_k is considered as a result state of the action E in the situation r.

The intention is that it shall be possible to characterize the resulting state r_k in terms of E and r, but without referring explicitly to the details of the intermediate states. The following assumptions (or something similar) are needed in order to make this work as intended.

Definition 2.8 *If three states r, r_i and r_{i+1} are given, the pair r_i, r_{i+1} is said to respect r iff $(r_i(f) \neq r_{i+1}(f)) \rightarrow (r_i(f) = r(f))$, for any state variable f that is defined in R. Then, an action system $\langle R, E, C, R_c, G \rangle$ is said to be respectful iff for every $r \in R_c$ and every $E \in \mathbf{E}$, r is respected by every pair r_i, r_{i+1} in every transition chain, and the last element of the chain is a member of R_c.*

This condition amounts to a "write-once" or "single-assignment" property: if the action E is performed in state r, the world may go through a sequence of states, but in each step from one state to the next, there cannot be changes in state variables which have already been changed previously in the sequence, nor can there be any additional change in a state variable that has changed in the invocation transition from r to r1. This definition can be compared to Lin's definition of stratified systems in [68], or the definitions of applicability and successor axiom by Thielscher [124, 125, 123].

As a consequence of these definitions, if $\langle R, E, C, R_c, G \rangle$ is a respectful action system, and $r \in R_c, E \in \mathbf{E}$ and $G(E, r, r')$ holds for some r', then all transition chains that emerge from (E, r) are finite and cycle-free. The set of states will be denoted $N(E, r)$. $N(E, r)$ is derived from the elementary relations G and C in the action system.

Respectful action systems are intended to capture the basic intuitions of actions with indirect effects which are due to causation, as follows. Suppose the world is in a stable state r, and an action E is invoked. The immediate effect of this is to set the world in a new state, which is not necessarily stable. If it is not, then one allows the world to go through the necessary sequence of state transitions until it reaches a stable state. That whole sequence of state transitions is viewed as the action, and the resulting admitted state is viewed as the result state of the action.

One has to remember that this is a method used for assessments. This makes a comparison with more detailed methods somewhat problematic. Some general observations can, however, be made about the assessment method, for example that it handles causal cycles.

Sandewall uses a fixpoint-oriented approach, but the results of the cascades are captured by the N function. Using N, the approach can be considered to be a non-fixpoint method.

The fact that C is a relation between only two states and that change isn't remembered makes it impossible to distinguish between precondition and trigger. So, just as is the case in Geffner's [26, 27] and McCain and Turner's approaches, preconditions are not expressible.

2.2.7 Fluents Calculus

In this section we present the Fluents Calculus [127, 14, 50, 126, 51, 48, 119], which has been proposed for the solution of the ramification problem. First we give certain definitions for fluents calculus.

The definition of fluents does not change from the definition that we have given for the situation calculus. A situation in fluents calculus is a term, which contains all their fluents (or negation) that is true.

For the representation of a situation the operator $\circ$ is used. This operator is binary. In the example with the simple electric circuit in order to represent the situation at which the fluents $up(s_1), \neg up(s_2), \neg light$ are true, we have $up(s_1) \circ \neg up(s_2) \circ \neg light$.

The order of fluents is not important. Thus $up(s_1) \circ \neg up(s_2) \circ \neg light = \neg up(s_2) \circ \neg light \circ up(s_1)$. The operator $\circ$ has the following properties:

$$\begin{aligned}
&\forall x, y, z. \quad (x \circ y) \circ z = x \circ (y \circ z) \quad (associativy)\\
&\forall x, y. \quad x \circ y = y \circ x \quad (commutativy)\\
&\forall x. \quad x \circ \emptyset = x \quad (unit \;\; element)\\
&\forall x. \neg\neg x = x \quad (double \;\; negation)
\end{aligned}$$

The symbol $\emptyset$ denotes the unit element for the operator $\circ$. This set is very useful when we want to process empty sets of fluents. The four above axioms are called ACIN. In addition to the above axioms the following are also defined for each n-place function f and predicate P, for each $1 \leq i \leq n$.

$$\begin{aligned}
&x = x\\
&x = y \supset y = x\\
&x = y \wedge y = z \supset x = z\\
&x_i = y \supset f(x_1, ...x_i, ...x_n) \equiv f(x_1, ...y, ...x_n)\\
&x_i = y \supset [P(x_1, ...x_i, ...x_n) \equiv P(x_1, ...y, ...x_n)],
\end{aligned}$$

All the variables are considered to be global. All the above axioms are used in equalities but in order to be able to use the above theory for solving the ramification and frame problems, it is necessary to prove inequalities of form $up(s_1) \circ light \neq up(s_2) \circ light$. This inequality is based on the fact that $s_1 \neq s_2$. In order to prove these inequalities, we make the hypothesis that starting from different fluent, the terms that are produced will never be equal. The above axioms are collectively referred to as EUNA for brevity (extended unique name assumption). For ease in the representation of a situation $S = l_1 ... l_n$ we use the symbolism $T_S = l_1 \circ .... \circ l_n$.

From EUNA axioms we can produce the following conclusions:

$$\begin{aligned}
&\forall x, z. [up(x) \circ z = up(s_1) \circ up(s_2) \circ \neg light] \supset\\
&(x = s_1 \wedge z = up(s_2) \circ \neg light) \vee (x = s_2 \wedge z = up(s_1) \circ \neg light)
\end{aligned}$$

The following have been established in [14, 50]

$$\begin{aligned}
&If \quad A \subseteq B, \quad then \quad EUNA \models \exists z. T_A \circ z = T_B, \quad else \quad EUNA \models \forall z. T_A \circ z \neq T_B\\
&If \quad A \subseteq B, \quad then \quad EUNA \models \forall z. [T_A \circ z = T_B \equiv z = T_{B \setminus A}]\\
&If \quad A \cap B = \{\}, \quad then \quad EUNA \models \forall z. [z = T_A \circ T_B \equiv z = T_{A \cup B}]
\end{aligned}$$

Now we present the solution to the ramification problem. Assume the following set of causal relationship

$$a_1[x_1] = a_1 \quad transforms \;\; C_1 \;\; into \;\; E_1, ... \; a_n[x_n] = a_n \quad transforms \;\; C_n \;\; into \;\; E_n$$

We define the ternary predicate $Action(a, c, e)$, which is true if and only if there exists action law with action a, with condition T_c^{-1} and effect T_e^{-1} [2] ,

$$Action(a, c, e) \equiv \bigvee \exists x_i[a = a_i \wedge c = T_{c_i} \wedge e = T_{e_i}]$$

Consider the previous example with the simple circuit. In this case,

$$\begin{aligned} Action(a, c, e) \equiv \exists x[a = toggle(x) \wedge (c = \neg up(x) \wedge e = up(x) \\ \vee c = up(x) \wedge e = \neg up(x)] \end{aligned}$$

Notice that the predicate *Action* produces the direct effects of an action. In order to solve the ramification problem we must define some other predicates which produce the indirect effects. Assume the set of causal relationships:

$$\begin{aligned} &\epsilon_1 \quad causes \;\; \rho_1 \;\; if \;\; \Phi_1 \\ &..... \\ &\epsilon_n \quad causes \;\; \rho_n \;\; if \;\; \Phi_n \,. \end{aligned}$$

We define the predicate $Causal(l_\epsilon, l_\rho, s)$ as follows:

$$Causal(l_\epsilon, l_\rho, s) = \bigvee \exists \vec{xi}[l_\epsilon = \epsilon_i \;\; \wedge l_\rho = \rho_i \wedge Holds(\Phi_i, s)]$$

The above axiom means that if in a situation s the formula Φ_i is true and the triggering fluent ϵ_i is true, then ρ_i will become true (which is the indirect result of the execution of an action).

Assume the following causal relationships:

[2] The fluent formula T_c^{-1} is the negation of the fluent formula T_c (resp. T_e^{-1}).

$$\begin{array}{llll} up(s_1) & causes & light & if \;\; up(s_2) \\ up(s_2) & causes & light & if \;\; up(s_1) \\ \neg up(s_1) & causes & \neg light & if \;\; T \\ \neg up(s_2) & causes & \neg light & if \;\; T \end{array}$$

Then the predicate $Causal$ is:

$$\begin{aligned} Causal(l_\epsilon, l_\rho, s) \equiv\; & l_\epsilon = up(s_1) \wedge l_\rho = light \wedge Holds(up(s_2), s) \\ & \vee\, l_\epsilon = up(s_2) \wedge l_\rho = light \wedge Holds(up(s_1), s) \\ & \vee\, l_\epsilon = \neg up(s_1) \wedge l_\rho = \neg light \\ & \vee\, l_\epsilon = \neg up(s_2) \wedge l_\rho = \neg light \end{aligned}$$

The predicate $Causal$ encapsulates all the indirect effects. Thielsher [124, 125] define the predicate $Causes(s, e, s', e')$, which means that starting from the situation s and having as effects e (direct and indirect) of some action up to this moment then we transition from the situation s in s' with effects e'.

$$\begin{aligned} & Causes(s, e, s', e') \equiv \\ & \quad \exists l_\epsilon, l_\rho[Causal(l_\epsilon, l_\rho, s) \wedge \exists v.\epsilon_i \circ v = e \\ & \quad \exists z[\exists \rho_i \circ z = s \wedge s' = z \circ \rho_i] \\ & \quad [\forall w.\neg\rho_i \circ w \neq e \wedge e' = e \circ \rho_i \\ & \quad [\exists w.[\neg\rho_i \circ w = e \wedge e' = w \circ \rho_i]] \end{aligned}$$

As we may observe, in a situation s in which up to this moment the effects are e, then if in them exists the ϵ_i which as indirect effect the ρ_i, then if ρ_i is not contained in the situation s, we have a transition to a new situation s' which contains the ρ_i and does not contain $\neg\rho_i$. Also, if the set e contains $\neg\rho_i$ then we remove it and add the ρ in e'. The predicate $Causes(s, e, s', e')$ is true if and only if from the situation s and with the effects e with the evaluation of causal relationships, there is a transition to the situation s' with effects e'.

In the example with the simple electric circuit we have

$$\begin{array}{l} Causes(up(s_1) \circ up(s_2) \circ \neg light, up(s_1), up(s_1) \circ up(s_2) \circ light, up(s_1) \circ light) \\ Causes(up(s_1) \circ up(s_2) \circ \neg light, up(s_2), up(s_1) \circ up(s_2) \circ light, up(s_2) \circ light) \\ \ldots\ldots \end{array}$$

The predicate $Causes$ does not solve the ramification problem because it shows only a transition from a situation into another, which contains some indirect effects of an action. It does not ensure that the new situation is consistent. In order to solve the ramification problem it is necessary to find a consistent situation. For this reason we import the predicate $Acceptable(s)$, which is true if and only if the situation s is consistent. As we have already mantioned in order to solve the ramification problem it is necessary to find a consistent situation after the execution of action. In order to find this situation, one may have to go through a lot of intermediate situations. In other words it could be a sequence of situation $s_1, s_2, ..., s_n$, with s_n being the consistent situation. For each situation s_i except the s_n, there is a transition $Causes(s_i, e_i, s_{i+1}, e_{i+1})$ to the next. Consequently we need a predicate that will incorporate all this intermediate transitions. We defined the predicate $Ramify_s(s, e, s', e')$ as follows:

$$\begin{aligned}
&Ramify_s(s,e,s',e') \equiv \forall\Pi[\ \forall s_1, e_1 . \Pi(s,e,s',e') \\
&\qquad\qquad \wedge\ [\forall s_1, e_1, s_2, e_2, s_3, e_3 \\
&[\Pi(s_1,e_1,s_2,e_2) \wedge Causes(s_2,e_2,s_3,e_3) \supset \Pi(s_1,e_1,s_3,e_3)]] \supset \Pi(s,e,s',e')] \\
&\wedge Acceptable(s')
\end{aligned}$$

As we observe the predicate $Ramify_s(s, e, s', e')$ is true if and only if the (s, e, s', e') belongs to the transitive closure of $Causes$ (which gives us the transition from one situation to another and the situation s' is consistent). Finally we define the predicate $Ramify(s, e, s')$ which is true if starting from the situation s and with effects e we can reach in a consistent situation s'. This means that the predicate $Ramify(s, e, s')$ is true if there exists such a e' such that (s, e, s', e') belongs in the transitive closure of $Ramify_s$ and $Causes$.

$$\begin{aligned}
&Ramify(s,e,s') \equiv \forall\Pi[\ \forall s_1, e_1, s_2, e_2 . Ramify_s(s_1,e_1,s_2,e_2) \supset\ .\Pi(s_1,e_1,s_2,e_2) \\
&\qquad\qquad \wedge\ [\forall s_1, e_1, s_2, e_2, s_3, e_3, s_4, e_4 \\
&[\Pi(s_1,e_1,s_2,e_2) \wedge Causes(s_2,e_2,s_3,e_3) Ramify_s(s_3,e_3,s_4,e_4)\ \supset \Pi(s_1,e_1,s_4,e_4)]]\ \supset \exists e' . \Pi(s,e,s',e')] \\
&\wedge Acceptable(s')
\end{aligned}$$

The predicate $Ramify$ allows us to find a consistent situation when an action take place (with direct effects which are described by the $Action$). In the example with the circuit switch we have that

$$Ramify(\neg up(s_1) \circ up(s_2) \circ \neg light,\ up(s_1), up(s_1) \circ up(s_2) \circ light)$$

2.3 The Qualification problem

In the previous section we examined the ramification problem which refers to determining the indirect effects of actions. Related to it is the qualification problem which refers to determining the preconditions which must hold in order for the execution of an action to be possible.

We introduce the problem with an example. Assume that a driver wants to start his/her car. In order to achieve this, there must exist petrol in the tank. However this condition is not enough in order to allow the driver to drive his car. S/he will have to check whether there exists a potato in the exhausts pipe. Even when this is checked s/he cannot be sure that the car will start. It could be the case that while the driver is entering the car, somebody places a potato in the exhaust and thus the car cannot brought to operation.

In the example that we presented above, there are the following rules

$$\begin{aligned} \neg gas &\supset \neg run \\ potato\_in\_tail &\supset \neg run\,. \end{aligned}$$

In this case, the conditions $\neg gas$ and *potato in tail* may $disqualify$ the action *run*. When one of the two conditions is true the action *run* will not be executable.

$$\begin{aligned} run \quad \underline{inexecutable} \quad \underline{after} \ \neg gas \\ run \quad \underline{inexecutable} \quad \underline{after} \ potato\ in\ tail\,. \end{aligned}$$

More specifically, when the formula $F = \neg gas \vee (potato\ in\ tail)$ is true the action *run* cannot be executed. In this case formula F disqualifies the action *run*.

$$F \supset disq(run)$$

Generalizing for each action a we define a fluent F_a, which when true disqualifies the action a.

Independently of the fact that some conditions are unlikely to hold, it is important that one be able to forbid the action if there conditions happen to hold. It is very important when this happens, to be able to forbidden the execution of action run. This means that we cannot ignore the second condition.

As we may observe, when an action executes, it may have as effect to disqualify some other action (for example the action *put a potato* disqualifies the action *run*). The problem of determining the context in which an action is allowed to execute is the **qualification problem** [35, 128, 82].

Now we present the qualification problem with another example. Consider a room in which there are boxes. Each box can only be at one place at a time. Also no box can be moved in a place that is occupied by another box except if it is placed on it. The precondition that enables the movement of a box which is in place l to a new place l' is that the new place is not occupied.

$$poss(move(x,l')) \supset clear(l')\,.$$

A placw is considered to be free if there is no object on it.

$$poss(on(x,l)) \supset disqualified(move(y,l)) \quad (1)$$

The above condition tells us that when we move a box x to the place l then we cannot move another box to that place. The effect of action $move(x,l)$ is

$$move(x,l) \supset on(x,l)\,.$$

The above example becomes even more difficult when two boxes are connected (between themselves) with a constant contact, as appears in figure 4.B. In this case, the above rules are not right. Assume that the initial situation is

$$on(A,l_1)\,, on(B,l_2)\,, on(C,l_3)\,.$$

The action $move(A,l_2)$ cannot be executed according to rule (1), because another box is at that place. This however is not right because when the box A is moved then the box B will be moved too (because the boxes are connected). Thus, if A is moved to place l_2 then box B will be also moved to a new place l_4. This means that the action $move(A,l_2)$ has as indirect result(ramification) the movement of box B.

$$move(A,l_2) \supset on(B,l_4)\,.$$

In this case rule (1) should change as follows

$$on(x,l) \land y \neq x \land \neg connected(x,y) \land on(y,l') \supset disqualified(move(x,l')) \quad (2)$$

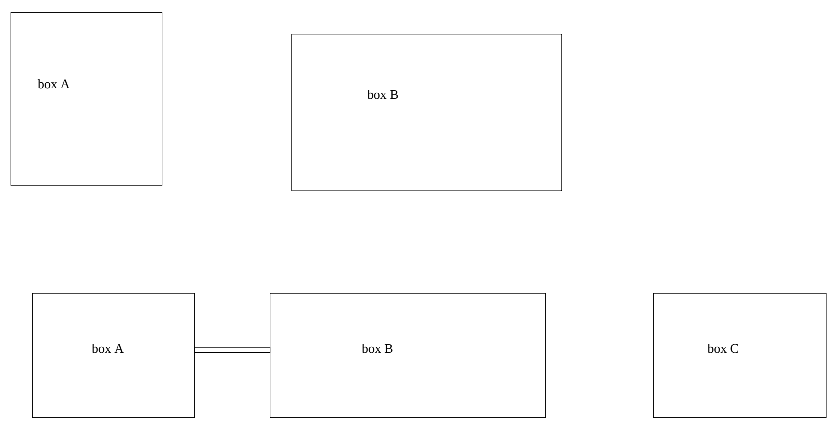

Figure 4: Qualification - Movement the box

The execution of the action $move(A, l_2)$ is allowed because the indirect effect that it has to release the place l_2. The conclusion is that there does not always exist a closer seperation of the ramification and qualification problems.

However rule (2) does not ensure that the action $move(A, l_5)$ will not be executed when this do not allow. For example, consider the action $move(A, l_5)$ which has as effect $on(B, l_3)$. The second action cannot be executed because in place l_3 exists the box C. Thus the rule (2) must be changed so as to include also all the indirect results of action *move*. This would have as result the number of disqualification rules to increase exponentially as the ramifications increase. For this reason, the default solution of the qualification problem which assumes that for each action, we determine explicitly all the qualification rules, cannot be applied.

2.3.1 The solution of nearest world

Various solutions has been proposed for the qualification problem. The first solution was proposed by the Ginsberg [35] and it is an extension of the nearest possible worlds approach, the solution that had been proposed for the ramification problem.

According to this solution each action will be executed. After the execution, if there is a consistent situation which includes all the effects of the action(direct and indirect), then the action is accepted.If no such situation exists, then the action will be rejected and the situation remains unchanged. The search for this situation becomes possible with the use of the algorithm of the nearest world that was described in the previous section.

Thus, in the above example when the action $move(A, l_5)$ is executed, it has as effect $on(B, l_3)$. Hence, it will not be executed because there is no exist consistent situation that would include this effect $(on(B, l_3))$. In the example with the connected boxes the movement is possible because when we move the box A then the box B will be moved too. Thus, if we execute the action, there is a consistent situation which includes the effects of the action (if the position of box A and B is different from l_3 which is the position of box C).

2.3.2 Explicit Solution

Some other proposals [82, 128] suggest that we must define explicitly the preconditions of each action. These preconditions must hold in order to allow the execution of the action. In the example with the car in order to start the car, none of the following must hold.

$$\begin{aligned}
&\exists x.in(x) \\
&tank - empty \\
&low - battery \\
&engine - problem\,.
\end{aligned}$$

We can collect these condition into the following expression:

$$\exists x.in(x) \vee tank - empty \vee low - battery \vee engine - problem \supset disq(run)\,.$$

Generalizing, for each action a we define a formula of the form

$$F^a = \bigvee F_i \equiv disq(a)\,.$$

In order for an action to be executed this formula must be false. Thus,

$$\bigwedge \neg F_i = true\,.$$

When a action a has as effect

$$a \quad causes \;\; \neg F_i \;\; if \;\; \phi\,.$$

If $\neg(F^a - F_i)$ [3] is true then

[3]This mean that the part of fluent formula F^a which remain if we "minus" from it the F_i (when we tell minus mean to delete from F^a all fluents which belong in the F_i).

$$a \quad causes \;\; \neg disq(a) \;\; if \;\; \neg(F^a - F_i)\,.$$

For example assume that the following hold

$$\begin{aligned}
&\exists x.in(x)\\
&\neg tank - empty\\
&\neg low - battery\\
&\neg engine - problem\,.
\end{aligned}$$

Then the formula F^{run} is true. Hence $disq(run)$ is true. The execution of action $clear-tail$ has as effect $\neg in(x)$. Thus the formula F^{run} is false and $\neg disq(run)$ become true.

$$clear - tail \quad causes \;\; \neg disq(run) \;\; if \;\; \neg(tank - empty \;\vee low - battery \vee engine - problem)\,.$$

2.3.3 Fluent Calculus and the Qualification Problem

In the fluent calculus, in order to address the qualification problem we define the following predicate as proposed in [14]:

$$\begin{aligned}
&Qualified([])\\
&Qualified([a^*|a]) \equiv Qualified(a^*) \wedge\\
&\exists s, s'[Result(a^*, s) \wedge Holds(\neg disq(a), s) \;\; \wedge Successor(s, a, s')
\end{aligned}$$

The predicate $Qualified$ shows when a sequence of action is executable. The above definition is recursive. Obviously $Qualified([])$ is true. The predicate $Qualified([a^*|a]$ is true when the action sequence $a^*|a$ is executable. The predicate $Result(a^*, s)$ is true when the situation s is the situation which is the result of the execution of action sequence a^*. The predicate $Successor(s, a, s')$ is true when $s' = do(a, s)$ holds. In order for $Qualified([a^*|a]$ to hold, the action sequence a^* must be executable (this is represented by $Qualified(a^*)$ and the action a must not be disqualified in s, the situation resulting from the execution of action sequence a^*.

2.4 Qualified Ramifications

As it became apparent from the previous section it is difficult to untie the ramification and qualification problems. A solution [82, 128, 129] is the extension of causal relations in order to include qualifications as well. More specifically, each causal rule changes from

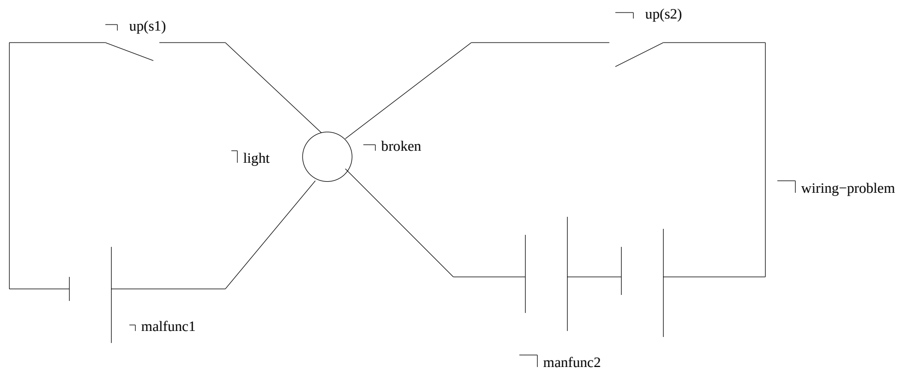

Figure 5: Qualifications - A complex circuit

$$\begin{array}{l} its \ \ previous \ \ form: \epsilon \quad causes \ \ \rho \ \ if \quad \Phi \\ to \ \ a \ \ new \ \ form: \\ \epsilon \quad causes \ \ \rho \ \ if \quad \Phi \wedge \neg ab_C \, . \end{array}$$

where ab_C is new fluent that when true disqualifies the rule C. This means that when ab_C is true then the execution of action ϵ does not have as effect ρ.

Consider the circuit which appears in figure 5. This circuit includes two individual circuits. The right circuit has a big battery. When the switch s_2 goes up then lamp will light up if there is no problem in the wire of circuit and the battery functions correctly. Thus, there is the rule

$$up(s_2) \supset broken \, .$$

This rule is evaluated, if the following does not hold

$$ab_2 \equiv malfunc_2 \vee wiring - problem \, .$$

The ab_2 is the fluent that disqualifies the above rule. We change the rule as follows

$$\neg ab_2 \supset [up(s_2) \supset broken] \, .$$

Also, if the left switch goes up then the lamp will light up if not broken and the wire and the first battery do not have any problem.

$$\begin{aligned}&ab_1 \equiv broken \vee malfunc_2 \vee wiring - problem\\&\neg ab_1 \supset [light \equiv up(s_1)]\,.\end{aligned}$$

In addition the above rules, there also exists the rule:

$$broken \vee malfunc_2 \vee wiring - problem \supset \neg light\,.$$

From the three above rules we produce the following causal relations.

$$\begin{array}{l}
up(s_1) \quad causes \quad light \quad if \quad \neg ab_1\\
\neg up(s_1) \quad causes \quad \neg light \quad if \quad \neg ab_2\\
up(s_2) \quad causes \quad broken \quad if \quad \neg ab_2\\
broken \quad causes \quad \neg light \quad if \quad T\\
malfunc_1 \quad causes \quad \neg light \quad if \quad T\\
wiring - problem \quad causes \quad \neg light \quad if \quad T\\
broken \quad causes \quad ab_1 \quad if \quad T\\
malfunc_1 \quad causes \quad ab_1 \quad if \quad T\\
wiring - problem \quad causes \quad ab_1 \quad if \quad T\\
\neg broken \quad causes \quad \neg ab_1 \quad if \quad \neg malfunc_1 \vee \neg wiring - problem\\
\neg malfunc_1 \quad causes \quad \neg ab_1 \quad if \quad \neg broken \vee \neg wiring - problem\\
\neg wiring - problem \quad causes \quad \neg ab_1 \quad if \quad \neg broken \vee \neg malfunc_1\\
malfunc_2 \quad causes \quad ab_2 \quad if \quad T\\
wiring - problem \quad causes \quad ab_2 \quad if \quad T\\
\neg malfunc_2 \quad causes \quad \neg ab_2 \quad if \quad \neg wiring - problem\\
\neg wiring - problem \quad causes \quad \neg ab_2 \quad if \quad \neg malfunc_2\,.
\end{array}$$

These rules cannot solve the ramification problem if the qualification problem is not addressed as well. For example, assume that initial situation is that the two switches are down, $\neg up(s_2)$ and $\neg up(s_1)$. Assume that the two actions $toggle\_switch(s_2)$ and $toggle\_switch(s_1)$ execute. Then there exist two possible worlds. Firstly, the lamp is broken. Consequently the action $toggle\_switch(s_1)$ (which has as direct effect $up(s_1)$) does not have as indirect effect that of turning on the lamp. Secondly, is the case that the wire or the second battery are spoiled. Thus, the action $toggle\_switch(s_2)$ does not have as indirect effect to break the lamp. Consequently the other action has as effect to turn on the lamp.

The conclusion is that global minimization is not suitable approach for the solution of ramification problem in the presence of the qualification problem.

In order to solve the problem we change each integrity rule as follows:

$$ab_C \supset C\,.$$

This means that an integrity rule will not be in effect in each situation. In order for the rule to be in effect its preconditions must hold.

In the above example we have that in order for the rule $up(s_2) \supset broken$ to be in effect the precondition $\neg ab_2$ must be true. Consider the situation of the circuit which approved in figure 5 , $S = \{\neg manfunc_1, \neg manfunc_2, \neg broken, \neg wiring, \neg up(s_1), \neg up(s_2)\}$. In the situation S, $\neg ab_1$, $\neg ab_2$ are true. Hence, the execution of action $toggle\_switch(s_2)$ will have as direct effect $up(s_1)$. Also the precondition of the rule $up(s_2) \supset broken$ is true (because $\neg ab_2$), thus, the indirect effect will be *broken*. Consequently, in the end there will exist a unique likely situation.

2.5 Summary

The world represented in a database is not static. It changes continuously. The changes occur as results of database transactions. An atomic database transaction can be considered as an action. So, we can say that the changes in a database occur as results of actions. These actions change stored data in the database, and thus they may affect integrity constraints which determine the consistent states of the database. A database is consistent when all integrity constraints are satisfied.

An action may have direct and indirect effects. There are infamous problems which arise by the execution of an action, the **frame, ramification** and **qualification** problems.

Some fluents are affected by some actions while others are not. The problem of determining which fluents are **not** affected when an action is executed is called the **frame problem** and was introduced by MacCarthy [81] in 1969. The main proposals for the solutions of frame problem are symarized below.

The simplest solution is the **monotonic** approach [81] . This solution suggests two kinds of axioms, namely *action axioms* and *frame axioms.* The action axioms specify the fluents that hold after the execution of an action and frame axioms specify the fluents which do not change after an action. The biggest problem with the monotonic approach is that we must determine explicity that some object does not change its situation when an action takes place. Usually, when an action takes place, it changes very few objects while most remain unaffected. The Default approach [104] proposes that it is necessary to provide axioms only for those predicates which tchange after one concrete action while for the rest there exists a default axiom. This shows that for each predicate which has not been declared that it changes after an action, the predicate remains as is.

For the solution of the frame problem, the use of the situation calculus [81] has been suggested. The situation calculus is a second-order language that represents the changes which occur in a domain of interest, as results of actions. One possible evolution of the world is a sequence of actions and is represented by a first-order term, called a situation. The initial situation S_0 is a situation at which no action has occurred yet. A binary function, $do(a, s)$ yields the situation resulting from the execution of an action a while in situation s.

Many solutions based on situation calculus have been suggested (Pednault [91], Hass [45] and Reiter [107]). The most important ones are those proposed by Reiter [107].

Pednault [91] proposed a set from positive and negative frame axioms (one for each pair action-fluent), moreover for each fluent $f(x, s)$

$$\begin{aligned} &\epsilon_f^+(A, s) \supset f(x, do(A, s)) \\ &\epsilon_f^-(A, s) \supset \neg f(x, do(A, s)), \end{aligned}$$

where $\epsilon_f^+, \epsilon_f^+$ are fluents formulas. When the $\epsilon_f^+(A, s)$ is true then in the situation $s' = do(A, s)$ the fluent f is true (respectively for the $\neg f$ and $\epsilon_f^-(A, s)$). The other solution that has been proposed by Hass [45], is based on a similar idea.

Reiter [107] suggests that for each fluent f there are two frame axioms

$$\begin{aligned} &Poss(a, s) \wedge (\gamma^+)_f(a, s) \supset f(do(a, s)) \\ &Poss(a, s) \wedge (\gamma^-)_f(a, s) \supset \neg f(do(a, s)). \end{aligned}$$

These axioms may be integrated in one

$$Poss(a, S) \supset [f(do(a, S)) \equiv (\gamma^+)_f(a, S) \vee f(s) \wedge \neg(\gamma^-)_f(a, S).$$

This means that a fluent f is true in a situation $S' = do(a, S)$ which came up after the execution of action a in a situation S only if f is false in S and the preconditions $(\gamma^+)_f$ (which make f true) hold in S', or f is true in S and the preconditions $(\gamma^-)_f$ (which make f false) do not hold in S'. An action a can execute in a situation S only if its preconditions (poss(a,S)) hold.

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 [81].

The simplest of the techniques suggested in the literature is the *minimal-change* approach [131]. It suggests that, when an action occurs in a situation S, we need to find a consistent situation S' which has the fewer changes from the situation S(S' is closer to S than any other situation).

Another solution is the *categorization* of fluents [62]. The fluents are categorized as primary and secondary. A primary fluent may change only as a direct effect of an action, while a secondary one 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 other this solution is not satisfactory.

A fluent can change or remain unchanged after an action. This depends on the context in which an action takes place. *Causal relationships* [82, 124, 125] capture this dependence between an action and an indirect effect. A causal relationship has the form

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

where ϵ is an action, ρ is the indirect effect and Φ is the context. 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 exists between context Φ and fluent ρ.

The problem of determining the context in which an action is allowed to execute is the **qualification problem**.

Several solutions have been proposed for the qualification problem. The most prominent ones are the *minimal-change* approach [131] and the *explicit-solution* [82, 128].

According to the minimal-change approach each action will be executed. After the execution, if there is a consistent situation which includes all the effects of the action (direct and indirect), then the action is accepted.If no such situation exists then the action will be rejected and the situation remains as it was before the execution.

The explicit solution suggests that, for each action a, we must determine a formula F^a which, when true, prohibits action a from executing. The formula F^a has the form

$$F^a = \bigvee F_i \supset disq(a)\,,$$

where each F_i is a fluent. When any of the F_i is true, the action a can not executed.

In this section we describe the infamous problems *frame*, *ramification* and *qualification* in the case that the time is absent. In the following section we describe the problem when the time is present.

3 Action Theories in Temporal Databases

3.1 Introduction to the Action Theories in Temporal Databases

In temporal databases all predicates and relationships exist over time. This means that the truth value of each fluent f depends on the time point at which it refers. As a consequence all the situations exist over time. A situation has a start time point and a finish time point. In a temporal database the evolution of a world is not just a sequence of situations $S_0, S_1, \ldots, S_n$ but a sequence of triples $(S_0, 0, t_0), (S_1, t_0, t_1), \ldots, (S_n, t_{n-1}, t_n)$, where the first element of each triple is the situation, the second is the situation start time and the third is the end time point of the situation. Notice that an evolution of the world in a conventional database may correspond to many different evolutions of a respective temporal database. Also when an action takes place, this happens at a specific time point.

The situation calculus [81] cannot represent the relationships which exist among fluents, situations, actions and time.

The situation calculus provide a very good context for the formalization of action theories in the conventional databases. Most of the solutions of the frame, ramification and qualification problems are based on the situation calculus. Thus, it is desirable to extend to the situation in order to encapsulate time. After that we may use it for the addressing of the frame, ramification and qualification problem in the temporal databases.

3.2 Situation Calculus and Time

A first proposal for encapsulating time in the Situation Calculus is due to Pinto and Reiter [101, 102].

More concretely they have defined a time line which correspond to a sequence of situations, beginning from the initial situation. Every time point corresponds to only one situation. All the situations are universally ordered. Actions that lead to different situations that happen at concrete time points. For example, $occurs(A, t)$ mean that action A happens at time moment t. Also, $holdsAT(f, t)$, which means that the fluent f is true time moment t.

The time line that was defined above corresponds to the real facts. This creates an asymmetry between situation calculus and this linear time line. In order to erase this asymmetry a branching structure of time and the predicate **actual** are introduced. With a branching time structure several histories of situations are created. The predicate actual(s) means that the situation s is true (it has really happened). The actual situations from a linear time line end at the present moment.

The properties of the predicate actual are

$$actual(S_0)$$

$$(\forall a, s).actual(do(a, s)) \supset actual(s) \wedge Poss(a, s)$$
$$(\forall a_1, a_2, s).actual(do(a_1, s)) \wedge actual(do(a_2, s)) \ \supset a_1 = a2 \,.$$

The first property shows that the initial situation is always true. Second that if a situation which resulted from the execution of an action in a situation s is true then the preconditions which allowed the execution of action a must hold in the situation s. The third property means that only one action can be executed in a situation. This is a weakness because you do not allow the simultaneous/parallel/concurrencing execution of more one of action. This leads the global ordering of the actions(as long as it concerns their execution sequence).

Certain examples of application of situation calculus with time are present below

3.2.1 Explaining Observations

Assume a car which is parking in the morning.

$$holds(Parked, S_m) \wedge actual(S_m) \,.$$

Later it is not parked. Then

$$actual(S_e) \wedge \neg holds(Parked, S_e) \wedge S_m < S_e \,.$$

This means that somebody have stolen the car or some driver took it from there. Hence

$$Poss(a, s) \supset holds(Parked, do(a, s)) \equiv$$
$$[holds(Parked, s) \wedge a \neq Steal \wedge a \neq TowAway\ ] \vee a = Park \,.$$

From the above we conclude that

$$(\exists s).(S_m \leq S_e) \wedge [occurs(Steal, s) \vee \ occurs(TowAway, s)] \,.$$

3.2.2 External Events

The Situation Calculus can express some facts which happen periodically. For example assume a bus passes from a stage every 15 minutes, between hours 08:00 and 24:00. Thus

$$occurs(BusArrives, t) \equiv (\exists n).0 \leq n \leq 64 \ \wedge t = 800 + 100 \times n/4 \,.$$

As a conclusion it comes out that

$$occurs(BusArrives, t) \supset occurs(BusArrives, t+1) \,.$$

3.3 The Frame and Qualification problems in Temporal Databases

We introduce these problems with an example [86]. Assume that a driver can not drive his car for four hours after drinking alcohol. This is expressed by a constraint of the form:

$$occurs(drink, t) \supset \neg occurs(drive, t_1) \ \wedge t_1 < t + 4h \,, \quad (1)$$

where t and t_1 are temporal variables and the predicate $occur(drink, t)$ means that the action drink is executed at time t. Also if a driver drinks alcohol, he is assumed drunk for four hours, and if he causes an accident then he is illegal until he gets a pardon. These are expressed by the following constraints:

$$\begin{array}{l} occurs(drink, t) \supset hold(drunk, t_1) \ \wedge t_1 < t + 4h \\ occurs(drink, t) \supset hold(\neg drunk, t_1) \wedge t_1 > t + 4h \\ occurs(car\_accident, t) \supset hold(illegal, \infty) \\ occurs(get\_pardon, t) \supset hold(\neg illegal, \infty) \end{array}$$

Now we can rewrite the (1) as follows:

$$holdAT(drunk, t) \supset disq(drive, t)$$

In a temporal database we need to describe the effects of an action not only to the next situation but possible for many future situations. In the above example, the action drink has the effect that the driver is drunk for the following four hours and cannot drive during this time (the time he is drunk). In these four hours, a number of other actions may execute leading to many different situations. In all these situations the action "drink alcohol" has the effect *drunk* (and also the action drive is disqualified).

The solutions which have been proposed for the frame problem in conventional databases cannot solve the problem in temporal databases because they determine effects of an action for the next situation. In the Temporal Databases the effects of an action start to hold from a specific time point and end in another specific time point.

The above weakness can be alleviated by constructing a correspondence between situations and actions to the time. Figure 6 depicts this correspondence. There are three parallel axes. The first is the situation axis, the second is the time axis and the third is the action axis. We assume that all actions are instantaneous. When an action occurs, the database changes into a new situation. For example, at time t_1 when action a_1 occurs, the situation changes from S_0 to $S_1 = do(a_1, S_0)$. We introduce two predicates $occurs_{A,T}(a,t)$ and $occurs_{S,T}(S,t)$ which correspond the actions and situations to a specific time stamp. Also, we introduce two functions $start(S)$ and $end(S)$ which show the start and the end of situation S.

The frame problem in temporal databases becomes more complex in the case the effects of an action may change the effects of another action in the current and the future situations. This happens between action *car_accident* and *get_pardon* (for the fluent *illegal*).

As the above relations show there is a dependence between the actions drive, accident, get-pardon and the fluents *drunk* and *illegal*. We represent this dependence with the predicate $duration(a,f,t,t')$ which when true, it means that the action a executes at time t and affects the fluent f at time t'. For our example we have that

$$\begin{array}{l}
duration(car\_accident, illegal, t, t') \;\; for \;\; all \;\; t, t' \;\;\; s.t. \;\; t' > t \\
duration(get\_pradon, illegal, t, t') \;\; for \;\; all \;\; t, t' \;\;\; s.t. \;\; t' > t \\
duration(drink, drunk, t, t') \;\; for \;\; all \;\; t, t' \;\;\; s.t. \;\; t < t' < t + 4h
\end{array}$$

The above relations take the form

$$\begin{array}{l}
\epsilon^{+}_{drunk}(t') \equiv occur_{A,T}(drink(x), t) \;\; \wedge duration(drink(x), drunk, t, t') \\
\epsilon^{-}_{drunk}(t') \equiv \neg \; \epsilon^{+}_{drunk}(t') \\
\epsilon^{+}_{illegal}(t') \equiv occur_{A,T}(car\_accindent(x), t) \;\; \wedge duration(car\_accident(x), illegal, t, t') \\
\epsilon^{-}_{illegal}(t') \;\; \equiv \neg \epsilon^{+}_{illegal}(t') \vee \;\; [occur_{A,T}(give\_pardon(x), t) \wedge \\
duration(get\_pardon(x), illegal, t, t')]
\end{array}$$

The $\epsilon^{-}_{drunk}(t')$ expresses the preconditions which when true, the fluents drunk is false at time point t' . The $\epsilon^{+}_{drunk}(t')$ expresses the preconditions which when true, the fluent drunk is true at time t'. For each fluent f we define the two preconditions ϵ^{+}_{f} and ϵ^{-}_{f}. As we observe the $\epsilon^{-}_{drunk}(t')$ is true when the preconditions $\epsilon^{+}_{drunk}(t')$ is false while the $\epsilon^{-}_{illegal}(t')$ is true when the $\epsilon^{+}_{illegal}(t')$ is false. This mean that the following case

$$\begin{aligned}\epsilon^-_{drunk}(t') \vee \epsilon^-_{drunk}(t') &\equiv FALSE\\ \epsilon^-_{illegal}(t') \ \vee \epsilon^-_{illegal}(t') &\equiv FALSE\end{aligned}$$

cannot occur [4]. This ensures that even if we do not have knowledge which of the two fluents $(f, \neg f)$ hold one of the two fluents is true. For example for the $(drunk, \neg drunk)$ the $\neg drunk$ holds if there is no knowledge for the opposite, while for the $(illegal, \neg illegal)$ the $\neg illegal$ is true if there is no knowledge for the opposite. For each pair $(f, \neg f)$ we decide which of the two holds by default. If the f holds then we change the

$$\begin{aligned}&\epsilon^+_f \quad as\\ &\epsilon^+_f \equiv \epsilon^+_f \vee \neg\epsilon^-_f\,.\end{aligned}$$

Else we change the

$$\begin{aligned}&\epsilon^-_f \quad as\\ &\epsilon^-_f \equiv \epsilon^-_f \vee \neg\epsilon^+_f\,.\end{aligned}$$

There is case to hold the following

$$\begin{aligned}\epsilon^-_{drunk}(t') \wedge \epsilon^-_{drunk}(t') &\equiv TRUE\\ \epsilon^-_{illegal}(t') \ \wedge \epsilon^-_{illegal}(t') &\equiv TRUE\,.\end{aligned}$$

In that case we must determine which of the two propositions $(\epsilon^-_f, \epsilon^+_f)$ give us more recent knowledge. In order to achieve that, we must "discover" the more recently action which influences the fluent f. We change the propositions $(\epsilon^-_f, \epsilon^+_f)$ as follows

$$\begin{aligned}&\epsilon^-_f(t, t')\\ &\epsilon^+_f(t, t')\end{aligned}$$

which mean that when the proposition $\epsilon^-_f(t, t')$ is true in time point t', the more recent action, which influences("negatively") the fluent f, has been executed at time point t (resp. for $\epsilon^+_f(t, t')$). In the above example we have

[4]If this cases occur mean that none of f and $\neg f$ is true.

$$\epsilon^+_{drunk}(t,t') \equiv occur_{A,T}(drink(x),t) \ \wedge duration(drink(x), drunk, t, t')$$
$$\epsilon^-_{drunk}(0,t') \equiv \neg\epsilon^+_{drunk}(t')$$
$$\epsilon^+_{illegal}(t,t') \equiv \ occur_{A,T}(car\_accindent(x),t) \wedge duration(car\_accident(x), illegal, t, t')$$
$$\epsilon^-_{illegal}(t,t') \equiv \neg\epsilon^+_{illegal}(t') \vee [occur_{A,T}(give\_pardon(x),t) \wedge$$
$$duration(get\_pardon(x), illegal, t, t')]$$

As we observe the first argument of propositions $\epsilon^-_f, \epsilon^+_f$ is produced by the predicate duration. Now we defined the two following predicate

$$E^-_f(t') \equiv \ [\epsilon^-_f(t_1,t') \wedge \ \neg\epsilon^+_f(t_2,t')] \vee [\epsilon^-_f(t_1,t') \wedge \ \epsilon^+_f(t_2,t') \wedge (t_1 > t_2)]$$
$$E^+_f(t') \equiv \ [\epsilon^+_f(t_1,t') \wedge \ \neg\epsilon^-_f(t_2,t')] \vee [\epsilon^+_f(t_1,t') \wedge \ \epsilon^-_f(t_2,t') \wedge (t_1 > t_2)]$$

If $E^-_f(t')$ is true then the fluent $\neg f$ becomes true at time point t' (resp. the $E^+_f(t')$). If $E^-_f(t')$ is true then the proposition $\epsilon^-_f(t_1,t')$ is true and the $\epsilon^+_f(t_2,t')$ is false or both are true but the action which influences the $\epsilon^-_f(t_1,t')$ takes place more recently than the action which influences the $\epsilon^+_f(t_2,t')$ $(t_1 > t_2)$. (Respectively for the proposition $E^+_f(t')$). In the above example assume the following execution of actions

$$occur(car\_accident, 4)$$
$$occur(get\_pardon, 7)$$

Suppose that at time point 8 the following hold

$$\epsilon^-_{illegal}(7,8) \equiv occur(get\_pardon, 7) \ \wedge duration(get\_pardon, illegal, 7, 8)$$
$$\epsilon^+_{illegal}(4,8) \equiv occur(car\_accident, 4) \ \wedge duration(car\_accident, illegal, 4, 8)$$

From the above we conclude that the $E^-_{illegal}(8)$ is true and $E^+_{illegal}(8)$ is false because $7 > 4$. This mean that the fluent $\neg illegal$ is true at time point 8.

Consider the following execution of actions

$$occur(car\_accident, 4)$$
$$occur(get\_pardon, 7)$$
$$occur(car\_accident, 9)\,.$$

At time point 10 the following are true

$$\epsilon^{-}_{illegal}(7,10) \equiv occur(get\_pardon, 7) \ \wedge duration(get\_pardon, illegal, 7, 10)$$
$$\epsilon^{+}_{illegal}(9,10) \equiv occur(car\_accident, 9) \ \wedge duration(car\_accident, illegal, 9, 10)$$

By the above we conclude that the $E^{-}_{illegal}(10)$ is false and $E^{+}_{illegal}(10)$ is true because $9 > 7$. This means that the fluent *illegal* is true at time point 10.

The algorithm for the production of the proposition $\epsilon^{-}_{f}(t,t')$, $\epsilon^{+}_{f}(t,t')$ is the following:

Algorithm for the production of $\epsilon^{-}_{f}(t,t')$, $\epsilon^{+}_{f}(t,t')$

```
1. For every time point $t'$, for every fluent $f$ do
    (a) If $occur_{A,T}(a_1,t_1).....occur_{A,T}(a_n,t_n)$ is the sequence of actions which have been executed then
    (b) Select the action $a_i$ s.t. $occur_{A,T}(a_i,t_i)$ and the predicate $duration(a_i,f,t_i,t')$ are true and for all $t_k$ s.t. $occur_{A,T}(a_k,t_k)$ and the predicate $duration(a_k,f,t_k,t')$ are true then $t_i > t_k$.
    (c) Set $\epsilon^{+}_{f}(t,t') = occur_{A,T}(a_i,t_i) \wedge duration(a_i,f,t_i,t')$.
2. do the same for fluent $\neg f$.
```

The following algorithm addresses the frame problem in temporal database

Algorithm for addressing the frame problem

```
1. For every time point $t'$ do
2. if $E^{+}_{f}(t')$ is true then $f$ is true.
3. else $E^{-}_{f}(t')$ is true then $\neg f$ is true.
```

Theorem 3.1 *The above algorithm addressine the frame problem.*

Proof: Assume that there is a time point t' at which the fluent f is true but the algorithm returns it false.

In order for the fluent f to be true at time point t' there must exist an action a which has been executed at time point t ($t \leq t'$). As an effect the fluent f is true at time point t' and between t and t' no action has executed which to has as an effect to make false the fluent f. These mean that the predicate $duration(a, f, t, t')$ is true. Thus

$$\epsilon_f^+(t,t') \equiv occur_{A,T}(a,t) \ \wedge duration(a,f,t,t')\,,$$

is true. Assume that the more recent action a_1 which influences negatively the fluent f takes place at time point $t_1 < t$ [5]. We have that

$$\begin{aligned}
\epsilon_f^+(t,t') &\equiv occur_{A,T}(a,t) \ \wedge duration(a,f,t,t') \\
\epsilon_f^-(t_1,t') &=\equiv occur_{A,T}(a_1,t_1) \ \wedge duration(a_1,f,t_1,t')
\end{aligned}$$

Because $t_1 < t$ we conclude that

$$\begin{aligned}
E_f^+(t') &= TRUE \\
E_f^-(t') &= FALSE
\end{aligned}$$

Thus the algorithm makes the fluent f true [6]. A contradiction.

■

Now we address the qualification problem in a temporal database. Assume that for an action a the $disq(a)$ is the formula which disqualifies the action a (in a conventional database). The following algorithm produces a formula for the case of a temporal database.

Algorithm for addressing the qualification problem

1. Transform each disqualified formula in its CNF form. Now each disqualified formula has the form $C_1 \wedge C_2 \wedge ... \wedge C_n$, where C_i is a disjunction.

2. For each i from 1 to n do

```
(a) assume $C_i = f_1 \vee ... \vee f_m$
    for each j from 1 to m do
    if $f_j$ then replace it from $E_f^+$
    else if $\neg f_j$ replace it from $E_f^-$
```

[5]In the opposite case the fluent f is false at the time point t'.

[6]We have assumed that the algorithm makes the fluent f false.

The above algorithm produces for each action a a formula $disq(a, t')$. When this formula is true the action a cannot be executed.

Consider the above example. For the action *drive* the above algorithm gives

$$disq(drive, t') \equiv E^{+}_{drunk}(t')$$

Consider the following execution

$$\begin{array}{l} occur(drink, 5) \\ occur(drive, 7) \end{array}$$

At time point 7 the following hold

$$\epsilon^{+}_{drunk}(5,7) \equiv occur_{A,T}(drink, 5) \wedge duration(drink, drunk, 5, 7)$$

The $\epsilon^{+}_{drunk}(5,7)$ is true because the $occur_{A,T}(drink, 5)$ and $duration(drink, drunk, 5, 7)$ are true $(5 < 7 < 5 + 4)$. The $\epsilon^{-}_{drunk}(0,7) \equiv \neg\epsilon^{+}_{drunk}(7)$ is false thus

$$E^{+}_{drunk}(7) = TRUE$$

This mean that the action *drive* cannot executed at time point 7.

Assume now the following execution

$$\begin{array}{l} occur(drink, 5) \\ occur(drive, 11) \end{array}$$

At time point 11 the following hold

$$\epsilon^{+}_{drunk}(11) \equiv occur_{A,T}(drink, 5) \wedge duration(drink, drunk, 5, 11)$$

The $\epsilon^{+}_{drunk}(5,11)$ is false because the $occur_{A,T}(drink, 5)$ is true but the $duration(drink,$ $drunk, 5, 11)$ is false (because $11 \notin [5, 5+4]$). The $\epsilon^{+}_{drunk}(11) \equiv \neg\epsilon^{+}_{drunk}(11)$ is true thus

$$E^{+}_{drunk}(7) = FALSE$$

This mean that the action *drive* will be executed at time point 11.

Theorem 3.2 *The above algorithm solves the qualification problem.*

Proof: Assume that at time point t, the action a must not be executed but it is executed. In order for this to happen the fluent formula $disq(a)$ must be true but the algorithm returns it false. The only case this can happen is that one fluent f to be true (or false) and the formula E_f^+ is false and E_f^- is true. This mean that the algorithm return that f is false (resp. true). Thus we can conclude that the formula E_f^+ and E_f^- cannot solve the frame problem [7]. This is contradiction because we have proven 3.1 that they solve it.

■

The solution of the qualification problem is result on the idea that the formula E_f^+ and E_f^- solve the frame problem.

3.4 The Ramification Problem in Temporal Databases

In this section we present the ramification problem in the context of temporal databases. We introduce this problem by means of an example. Assume that when public employees commit misdemeanors, then for the next five months they are considered illegal, except if they are granted a pardon. When public employees are illegal, then they must be suspended and cannot receive a promotion for the entire time interval over which they are considered illegal. Also, when public employees are suspended, they cannot receive a salary until the end of the suspension period. Public employees are evaluated for their work. If they receive a bad grade, then they are considered "bad employees" and cannot receive a promotion until they are evaluated positively. Otherwise, they are considered to be "good employees" and may take a bonus if not suspended. As we can observe, there are four actions (*misdemeanor*, *take_pardon*, *good_grade*, *bad_grade*) and seven fluents (*good_employee*, *bad_employee*, *illegal*, *take_salary*, *take_bonus*, *take_promotion*, *suspended*). The direct effects of the four actions are expressed by the following propositions[8]:

$$
\begin{aligned}
&occur(misdemeanor(p), t) \supset illegal(p, 5m) \quad (1)\\
&occur(take\_pardon(p), t) \supset \neg illegal(p, \infty) \quad (2)\\
&occur(bad\_grade(p), t) \supset \neg good\_employee(p, \infty) \quad (3)\\
&occur(good\_grade(p), t) \supset good\_employee(p, \infty) \quad (4)\,,
\end{aligned}
$$

where t is temporal variable and the predicate $occur(misdemeanor(p), t)$ denotes that the action $misdemeanor(p)$ is executed at time t. Also, we have and the following integrity constraints which describe the indirect effects of the four actions.

[7]Because the do not determine right the truth value of the fluent f.

[8]Quantifiers are omitted. Propositions are assumed to be implicitly universally quantified over their temporal and non-temporal arguments.

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

In a temporal context, we need to describe the direct and indirect effects of an action not only in the immediately resulting next situation but possibly for many future situations as well. In the above example, the action *misdemeanor*(p) has as indirect effect that the public worker is in suspension for the next five months. In these five months, the action *good_grade* may occur but even if this happens, the employee still cannot receive a promotion. This means that the world being modelled may change from one situation to another while the direct and/or indirect effects of action still hold. Also, in this time span other actions may occur leading to many different situations. Furthermore, five months after the execution of the action *misdemeanor* the situation may change without an action taking place (because the public worker is no longer considered illegal). Hence, fluents cannot be assumed to persist.

Previous proposals about addressing the ramification problem have used so called *causal relationships* [21, 82, 43, 124, 125]. They come short in 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 fluents assumption (i.e., no fluent may change truth value without an action taking place). The same weakness characterizes all other solutions of the ramification problem in conventional databases (e.g., [131, 62, 63, 64, 14, 50, 51]).

The above weakness can be alleviated by constructing a correspondence between situations and actions with time. Some proposal for that has been done by [82, 102, 105, 87, 88, 86]. We suggest the correspondence that appears in figure 6. There are three parallel axes: the situations axis, the time axis and the actions axis. For now, we assume that all actions are instantaneous. When an action takes place, the database changes into a new situation.

3.5 Previous Work in the Ramification Problem in Temporal Databases

The most prevalent previous works are those by Reiter [105], Reiter and Pinto [101, 102] and by Kakas [52, 53]. 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

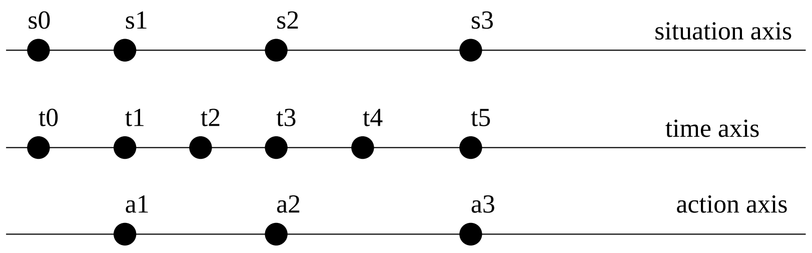

Figure 6: The correspondence among Time-Actions-Situations

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 rather than the ramification problem. However the work of Reiter sets the basis for encapsulating time in the situation calculus. In the section 3.6.1, we propose a further extension of the situation calculus based on Reiter's proposal.

Kakas [52, 53] 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).

$$\begin{array}{l}
L \quad holds \quad at \quad T \\
A \quad happens \quad at \quad T \\
A \quad initiates \quad F \quad when \quad C, \\
A \quad terminates \quad F \quad when \quad C, \\
L \;\; whenever \quad C \\
A \;\; needs \;\; C\,.
\end{array}$$

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 the example with the circuit we have the following dynamic axioms:

$$\begin{array}{l}
toggle\_switch(s_1) \quad initiates \quad up(s_1) \;\; when \;\; \neg up(s_1) \\
toggle\_switch(s_1) \quad initiates \quad \neg up(s_1) \;\; when \;\; up(s_1) \\
toggle\_switch(s_1) \quad terminates \quad up(s_1) \;\; when \;\; up(s_1) \\
toggle\_switch(s_1) \quad terminates \quad \neg up(s_1) \quad when \;\; \neg up(s_1) \\
toggle\_switch(s_2) \quad initiates \quad up(s_2) \;\; when \;\; \neg up(s_2) \\
toggle\_switch(s_2) \quad initiates \quad \neg up(s_2) \;\; when \;\; up(s_2)
\end{array}$$

$$\begin{array}{l} toggle\_switch(s_2) \;\; terminates \;\; up(s_2) \;\; when \;\; up(s_2) \\ toggle\_switch(s_2) \;\; terminates \;\; \neg up(s_2) \;\; when \;\; \neg up(s_2) \end{array}$$

Also we have the following static rules

$$\begin{array}{l} light \;\; whenever \;\; up(s_1) \wedge up(s_2) \\ \neg light \;\; whenever \;\; \neg up(s_1) \vee \neg up(s_2) \end{array}$$

Assume that the initial situation of circuit at time 0 is

$$S_0 = \{up(s_1), \neg up(s_2), \neg light\}$$

At this time we have that

$$\begin{array}{l} up(s_1) \;\; holds \;\; at \;\; 0 \\ \neg up(s_2) \;\; holds \;\; at \;\; 0 \\ \neg light \;\; holds \;\; at \;\; 0 \end{array}$$

Assume the following execution of actions:

$$\begin{array}{l} toggle\_switch(s_2) \;\; happens \;\; at \;\; 4 \\ toggle\_switch(s_1) \;\; happens \;\; at \;\; 8 \end{array}$$

The language E is based in the idea of persistence. This mean that no fluent changes its true value until some action causes this change. In our example, the situation does not change until the time point 4. At time point 4 the following axioms will be evaluated

$$\begin{array}{l} toggle\_switch(s_2) \;\; initiates \;\; up(s_2) \;\; when \;\; \neg up(s_2) \\ toggle\_switch(s_2) \;\; terminates \;\; \neg up(s_2) \;\; when \;\; \neg up(s_2) \end{array}$$

Now the new situation is

$$S_1' = \{up(s_1), up(s_2), \neg light\}$$

Now the following static rule is evaluated

$$light \ \ whenever \ \ \ up(s_1) \wedge up(s_2)$$

The new situation now is

$$S_1 = \{up(s_1), up(s_2), light\}$$

The fluents which hold are

$$\begin{array}{l} up(s_1) \ \ holds \ \ at \ \ 4 \\ up(s_2) \ \ holds \ \ at \ \ 4 \\ light \ \ holds \ \ at \ \ 4 \end{array}$$

The situation does not change until time point 8. Then the situation changes again.

In E, one cannot declare effects that persist over a time span as in the aforementioned example where, if someone drinks then s/he is drunk for the subsequent five hours. In order to achieve this, it is necessary for an action to occur after five hours. This means that the users must explicitly determine all the indirect and direct effects. Also, E cannot represent delayed effects, as e.g., if someone drinks alcohol then s/he becomes drunk half an hour later and remains drunk for the next five hours. We consider these assumptions rather strong and examine the problem in a strictly more general setting. The languages E works satisfactorily only when the world which described is based on the persistence of fluents (like the circuit).

Related also is the language VHDL integrated circuit design used for symbolic temporal behavior. This language is based on the assumption that fluents persist until their truth value is changed. This mean that in the case of driver we must describe the direct and indirect effects as follows:

$$\begin{array}{l} if \ \ (occur(drink(p), t) \quad then \\ \qquad drunk(p) \\ \qquad wait \ \ for \ \ 5 \ \ month \\ \qquad \neg(drunk(p))\,. \end{array}$$

As we observe from this description, the user must determine explicitly all direct and indirect effects. Also, the description becomes more difficult if an action can change the effects of another action. In the above example, assume that there is an action *drink_glycose* which has as direct effect $\neg drunk$. In that case, we need to describe an interrupt in order to ensured the effects of the second action. The language VHDL, as E, works satisfactory only when the world which described is based in the persistence of fluents.

The **Temporal Action Logic** is logic which encapsulates time. TAL consists of a compact surface language $L(SD)$ which can be translated into a first order language $L(FL)$.

Definition 3.1 *The $L(SD)$ consists of*

1. *A set of observation statements. Each observation statement shows that some fluents are true at some timepoints.*
2. *A set of occurrences. Each occurrence shows the time point and the action which take place at the specific time point.*
3. *A set of action laws which describe the direct effects of the action and the time intervals at which these effects hold.*

Consider the example which we borrow from [43], with a car which starts its operation. Initially the engine and the car lights are off, the fan and radio are turned on but since they are not connected to the battery until the key is turned there is neither fresh air nor any music. Assume that the time start at 0 and between time point 3 and 4 the driver turns the key. We have

$$\begin{array}{ll}
obs1 & [0](\neg engine \wedge \neg light \neg air \neg music) \\
occ1 & [3,4]TurnKey \\
acs1 & [3,4]TurnKeyleadsto[3,4]engine := true \wedge \\
 & \quad [3,4]lights := true \wedge \\
 & \quad ([3,4]fan \rightarrow [3,4]air := true) \wedge \\
 & \quad ([3,4]radio \rightarrow [3,4]music := true)
\end{array}$$

The first element describes the initial situation of the world. The second element shows that the action *turnKey* takes place at the time points between 3 and 4. The third element show that if the action *TurnKey* takes place in the $[3,4]$, then the fluents *engine* and *lights* must become true between the time points 3 and 4.

TAL has been extended in order to encapsulate the indirect effects of an action. The language $L(SD)$ is extend as follows

Definition 3.2 *The language $L(SD)$ in addition two the three sets of the definition 3.1 it introduces a set of dependency laws of the form*

$$\forall t_1, \bar{u}[\gamma \rightarrow ([t]\alpha \gg [t']\beta].$$

The dependency law means that "if γ is true and α becomes true at time point t then the β become true at time point t'.

Assume the example of shooting someone. Assume that st the time point 0 the person is alive and walking. At time point between $[1, 2]$ someone shoots him.

$$\begin{array}{ll} obs1 & [0]alive = true \wedge walking = true \\ occ1 & [1,2]Shoot \\ acs1 & [1,2]Shoot \leadsto [1,2]alive := false \\ dep1 & \forall t_1([t_[1]alive = false \gg [t_1]walking = false \end{array}$$

The problem with TAL is that it assumes that the persistence of fluents holds. In the above example with the public worker, if we use TAL we must describe explicitly when the no persistence effects end. In order to achieve that, we must define an action and specify the time point of its execution which shows the end of no persistence effects. For example assume that a public worker does a misdemeanor at t then we must determine that after 5 months$(t + 5)$ the new action "end-illegal" which has as effect not illegal must be executed. This assumption create many new problem. Assume the following execution

$$\begin{array}{l} occur(misdemeanor(p), 3) \\ occur(take\_pardon(p), 5) \\ occur(misdemeanor(p), 7) \end{array}$$

The first action has the non persistent effect that the fluent *illegal* holds for 5 month. 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 will cancel the effect of 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 for each no persistent effect of each action.

The Event Calculus has been proposed by [84, 59]. In the event calculus the time is discrete. We defined the following predicates and relations

$$\begin{array}{l} Initiates(a, f, t) \\ Terminates(a, f, t) \\ Initially_p(f) \\ Initially_N(f) \\ t_1 < t_2 \\ Happens(a, t) \\ Happens(t_1, a, t_2) \\ HoldsAt(f, t) \\ Clipped(t_1, f, t_2) \\ declipped(t_1, f, t_2) \\ Releases(a, f, t) \end{array}$$

The first predicate means that the fluent f starts to holds 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 the time point t_1 is before the 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 the fluent f holds at time point t, the ninety means that fluent f ceases to hold between times t_1 and t_2, the tenth mean that the fluent f starts to hold between times t_1 and t_2. The last predicates mean that the fluent f is not subject to inertia after action a at time point t.

The event calculus is very similarly 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 points(the predicates $Clipped(t_1, f, t_2)$, $declipped(t_1, f, t_2)$, $Happens(t_1, a, t_2)$).

3.6 Extended Situation Calculus

We extend the temporal situation calculus as follows:

- We define functions $start(a)$ and $end(a)$, where a is an action. The former function returns the time moment at which the action a starts while the latter returns the time moment at which it finishes.

- We define functions $start(S)$ and $end(S)$ for situations. They return the time moments at which situation S begins and ends respectively.
- We define the functional fluent $f_\alpha(a)$ as $current\_moment - start(a)$, i.e., the duration of execution of action a until the present moment.
- Time is discrete and isomorphic to the set of natural numbers.
- Each fluent f is represented as $f(t')$, which means that the fluent f is true in the time interval $[current\_moment, current\_moment + t']$. $\neg f(t')$ means that the fluent f is false in the time interval $[current\_moment, current\_moment + t']$. As time progresses, the value of t' is decreased by one time unit.
- We define the function $FluentHold(S, t)$ which returns the set of all fluents which are true of time moment t.
- We define as temporal situation a situation which contain fluents of the form $f(t)$. This means that a temporal situation contain information not only about the truth value of the fluent but also how long they will be true. The function $FluentHold$ is a $n \to 1$ function from the domain of temporal situations to the domain of the situations [9]. The domain of temporal situations is infinite while the domain of situations is finite because if F is the number of fluent there are 2^F different situations. For the rest of the thesis the term situation will mean temporal situation except if we explicitly specify the opposite.
- Actions are ordered as follows:

  For instantaneous actions

  $$\begin{array}{l} a_1 < a_2 < .... < a_n \quad , when \\ start(a_1) < start(a_2) < ..... < start(a_n) \end{array}$$

  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.

  For actions with duration, $a_1 < a_2$ when $end(a_1) < start(a_2)$. Two actions a_1, a_2 will be executed concurrently when $start(a_1) \leq start(a_2) \leq end(a_1) \leq end(a_2)$ holds.

  We assume that all actions which execute at the same time moment will be executed concurrently.
- We define the function do as $do \ : \ action^n \times situation \longrightarrow situation$. $do(\{a_1, a_2, .., a_n\}, S) = S_1$ means that the actions $a_1, a_2, .., a_n$ execute concurrently in the situation S and the result is the situation S_1.
- For two situations S_1, S_2, $S_1 < S_2$, when $end(S_1) \leq start(S_2)$. It is not necessarily the case that $S_2 = do(\{a_{i1}, a_{i2}, ...\}, ..., do(\{a_{j1}, ...\}, S_1)...)$.

[9] $FluentHold(\{f_1(10), f_2(34)\}, 12) = FluentHold(\{f_1(2), f_2(3)\}, 12) = \{f_1, f_2\}$.

- We extend predicate $poss(a, S)$ as follows:
$poss(\{a_1, a_2, ...a_n\}, S) = \bigwedge_{i=1,...,n} poss(a_i, S)$. This means that the actions $\{a_1, a_2, ...a_n\}$ can execute concurrently if and only if the preconditions of each action are true.
- We define naturals actions as action which execute in pre-determined time points if their preconditions hold.
- We define as a legal (consistent) situation, a situation in which all integrity constraints are satisfied.

3.6.1 Fundamental Axioms

We use the axioms which have been defined by Reiter [105]

$$\begin{array}{lr} S_0 \neq do(\{a_1, ..., a_n\}, S) & (1) \\ do(\{a_1, ..., a_n\}, S) = do(\{a_1^{'}, ..., a_n^{'}\}, S^{'}) \supset & \\ \qquad\qquad \{a_1, ..., a_n\} = \{a_1^{'}, ..., a_n^{'}\} \wedge S = S^{'} & (2) \\ start(a(t)) = t & (3)\,. \end{array}$$

Reiter has also defined one other axiom, namely

$$(\forall P).P(S_0) \wedge (\forall \{a_1, ..., a_n\}, S)\ [P(S) \supset\ P(do(\{a_1, ..., a_n\}, S)] \supset (\forall S)P(S)\,.$$

This is an inductive axiom which means that each situation is the result of the execution of a sequence of actions. Thus, if the initial situation is known we can determine which fluent is true in each situation. This axiom does not hold in our case because the transition from one situation to the next does not necessarily happen after the execution of an action. In order for the above axiom to hold, we could define a natural action a_f for each fluent f. The only direct effect of the action a_f is that the fluent f becomes false ($f(0)$ [10]). This means that when an action a has as effect $f(10)$ the action a_f will execute 10 time moments later. Natural actions do not affect the world being modelled. They are employed to ensure that the transition from one situation to the next is the result of the execution of some action (natural or not). The transition from one situation to the next happens when the truth value of at least one fluent changes. By the inclusion of natural actions no fluent can change its truth value without some action taking place.

[10]This mean that the fluent f is true for the 0 next time points. Thus is false

3.7 symmary

In chapter 3 we address the *frame, ramification* and *qualification* problems in temporal databases and we propose one solution for the frame and qualification problems. Also, we present the most important previous work on the ramification problem in temporal databases and we extended the situation calculus in order to encapsulate time.

In temporal databases all predicates and relationships exist over time. This means that the truth value of each fluent f depends on the time point at which it refers. As a consequence all the situations exist over time. A situation has a start time point and a finish time point. In a temporal database the evolution of a world is not just a sequence of situations $S_0, S_1, \ldots, S_n$ but a sequence of triples $(S_0, 0, t_0), (S_1, t_0, t_1), \ldots, (S_n, t_{n-1}, t_n)$, where the first element of each triple is the situation, the second is the situation start time and the third is the end time point of the situation. Notice that an evolution of the world in a conventional database may correspond to many different evolutions of a respective temporal database. Also when an action takes place, this happens at a specific time point.

In a temporal database we need to describe the effects (direct and indirect) of an action not only to the next situation but possible for many future situations. This happen because in the temporal databases the effects refered in the time interval while in the convertional databases the effects refered only in the next situation. This means that the world being modelled may change from one situation to another while the direct and/or indirect effects of action still hold. Also, in this time span other actions may occur leading to many different situations. Hence, fluents cannot be assumed to persist. All solutions which have been proposed for the three above problems based on the idea of the persistent of fluents. Hence these solutions cannot solve the ramification problem in the case of temporal databases.

The above weakness can be alleviated by constructing a correspondence between situations and actions with time. We suggest the correspondence that appears in figure 6. There are three parallel axes: the situations axis, the time axis and the actions axis. For now, we assume that all actions are instantaneous. When an action takes place, the database changes into a new situation.

The situation calculus provide a very good context for the formalization of action theories in the conventional databases. Most of the solutions of the frame, ramification and qualification problems are based on the situation calculus. Thus, it is desirable to extend to the situation in order to encapsulate time. At section 3.6 we propose an extension of situation calculus which encapsulate the time. Our extension admir us to discrebe the actions and their effects when the time is presnt.

In the next chapter we use this extension in order to solve the ramification problem in the temporal databases.

4 The Ramification Problem in Temporal Databases - The Solutions

4.1 Sequential execution

In this section we present a solution for the ramification problem in temporal databases, when the actions execute sequentially. This solution extends the solution which has been proposed by McCain and Turner [82].

Each action A is represented as $A(t)$ which mean that the action A is executed at time t. Each fluent f is represented as $f(t')$, which means that the fluent f is true in the time interval $[curent\_moment, current\_moment + t']$. $\neg f(t')$ means that the fluent f is false in the time interval $[curent\_moment, current\_moment + t']$. As time progresses, the value of t' is decreased by one time unit.

For each action A we define one axiom

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

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

$$\begin{aligned} G(t,t') &\supset f(t') \\ B(t,t') &\supset \neg f(t')\,, \end{aligned}$$

where $G(t,t')$ is a formula which when true (at time point t) causes fluent f to become true at the next t' time moments (respectively for $B(t,t')$). These axioms encapsulate the indirect effects of an action. The former of axioms 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 correspondent fluent is false.

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 satisfied in the new situation. Thus the static rules must be produced in such way that when an integrity constraint is not satisfied in a situation produced by the 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

$$\begin{aligned} &C_1 \wedge ... \wedge C_n \;\; where \\ &C_i \equiv f_{1i} \vee ... \vee f_{mi} \end{aligned}$$

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

$$\neg[\bigvee f_p \;\; s.t \;\; p \in \{1i, ..., mi\} \;\; and \;\; p \neq ki] \vee FL \supset f_{ki}$$

where FL is a fluent formula and $ki \in \{1i, \ldots, mi\}$. After the execution the 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 satisfied 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 dependencies between fluents (In section 4.1.1 we describe the binary relationship with details).

We need $O(A + 2 * F)$ axioms, where A is the number of actions and F the number of fluents. Additionally, it may be necessary to define one default axiom for each pair $(f, \neg f)$. The default axiom has one of the following forms: $f(0) \wedge \neg f(0) \supset f(t)$ or $f(0) \wedge \neg f(0) \supset \neg f(t)$. The default axioms are evaluated when $f(0) \wedge \neg f(0)$ holds [11]. It is not necessary to define default axiom for each pair $(f, \neg f)$, if there is no case in which $(f(0) \wedge \neg f(0))$ holds.

Consider again the example with the public worker: if a public employee commits a misdemeanor, then for the next five months s/he is considered illegal, except if s/he receive a pardon. When a public employee is illegal, then s/he must be suspended and cannot take promotion for the entire time interval over which s/he is considered illegal. Also when a public employee is suspended s/he cannot take his/her salary until the end of the suspension period. Each public employee is evaluated for his/her work. If s/he receive a bad grade, then s/he is assumed to be a bad employee and s/he cannot take promotion until s/he receives a good grade. If s/he receives a good grade, then s/he is assumed to be a good employee and s/he may take a bonus if s/he not suspended. Also assume that a public worker is not illegal if there does not exist information that proves s/he is illegal, is not suspended if there does not exist information that proves s/he is suspended and takes his/her salary if there does not exist information that proves the opposite. This helps us define the default axioms. As we observe we have four actions $misdemeanor, take\_pardon, good\_grade, bad\_grade$ and seven fluents $good\_employee, bad\_employee, illegal, take\_salary$, $take\_bonus$, $take\_promotion$, *suspended*. The direct effects of the four actions are expressed in propositional form by the following constraints[12]:

$$occur(misdemeanor(p), t) \supset illegal(p, 5m) \quad (1)$$

[11] In that case mean that the fluent f and its negation are false. The default axioms determine which of two fluent f or $\neg f$ assumed true in that case

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

$$occur(take\_pardon(p), t) \supset \neg illegal(p, \infty) \quad (2)$$
$$occur(bad\_grade(p), t) \supset \neg good\_employee(p, \infty) \quad (3)$$
$$occur(good\_grade(p), t) \supset good\_employee(p, \infty) \quad (4)\,,$$

where t is a temporal variable and the predicate $occur(misdemeanor(p), t)$ denotes that the action $misdemeanor(p)$ is executed at time t. Also we have the following integrity constraints which describe the indirect effects of the four actions.

$$illegal(p, t_1) \supset suspended(p, t_1) \quad (5)$$
$$illegal(p, t_1) \supset \neg take\_promotion(p, t_1) \quad (6)$$
$$suspended(p, t_1) \supset \neg take\_salary(p, t_1) \quad (7)$$
$$\neg good\_employee(p, t_1) \supset \neg take\_promotion(p, t_1) \quad (8)$$
$$\neg suspended(p, t_1) \wedge good\_employee(p, t_2) \supset take\_bonus(p, min(t_1, t_2) \ (9)$$
$$\neg good\_employee(p, t_1) \supset \neg take\_bonus(p, t_1) \quad (10)$$
$$\neg suspended(p, t_1) \supset take\_salary(p, t_1) \quad (11)\,.$$

In a temporal database we need to describe the direct and indirect effects of an action not only in the immediately resulting next situation but possibly for many future situations as well. In the above example, the action $misdemeanor(p)$ has the indirect effect that the public worker is in suspension for the next five months. In these five months the action $good\_grade$ may occur but even if that happens the employee cannot take promotion. This means that the world changes situations while the direct and indirect effects of some action still hold. In the above example the dynamic axioms are the (1) - (4) while the static axioms are

$$illegal(p, t_1) \supset suspended(p, t_1)$$
$$illegal(p, t_1) \vee \neg good\_employee(p, t_2) \ \supset \neg take\_promotion(p, max(t_1, t_2)$$
$$suspended(p, t_1) \supset \neg take\_salary(p, t_1)$$
$$\neg suspended(p, t_1) \wedge good\_employee(p, t_2) \supset take\_bonus(p, min(t_1, t_2))$$
$$\neg good\_employee(p, t_1) \supset \neg take\_bonus(p, t_1)$$
$$\neg suspended(p, t_1) \supset take\_salary(p, t_1)\,.$$

We have the following default axioms

$$illegal(p, 0) \wedge \neg illegal(p, 0) \supset illegal(p, \infty)$$
$$take\_salary(p, 0) \wedge \neg take\_salary(p, 0) \supset take\_salary(p, \infty)$$
$$suspended(p, 0) \wedge \neg suspended(p, 0) \supset suspended(p, \infty)\,.$$

4.1.1 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:

$$\begin{array}{ll} (a) & G_f \supset K_f \\ (b) & G_f \equiv K_f\,, \end{array}$$

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 figure 7. The integrity constraints specifying the behavior of this system are expressed as the following formulae:

$$\begin{array}{ll} (a) & light \equiv up(s1) \wedge up(s2) \\ (b) & relay \equiv \neg up(s1) \wedge up(s3) \\ (c) & relay \supset \neg up(s2) \end{array}$$

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 to 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 figure 7. The action $toggle\_switch(s_1)$ has as indirect effect to light the lamp and not to toggle the switch s_2. We observe that it is not reasonable to include the fluent pairs $(light, (up(s1))$,$(light, (up(s2))$, $(relay, (up(s1))$,$(relay, (up(s3))$ in I. The truth values of fluents *light* and *relay* cannot change as the direct effect of an action, so they cannot affect the truth values of other fluents.

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') *to* 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') to I. If f' can change its truth value as direct effect of an action then add (f', f) to I.

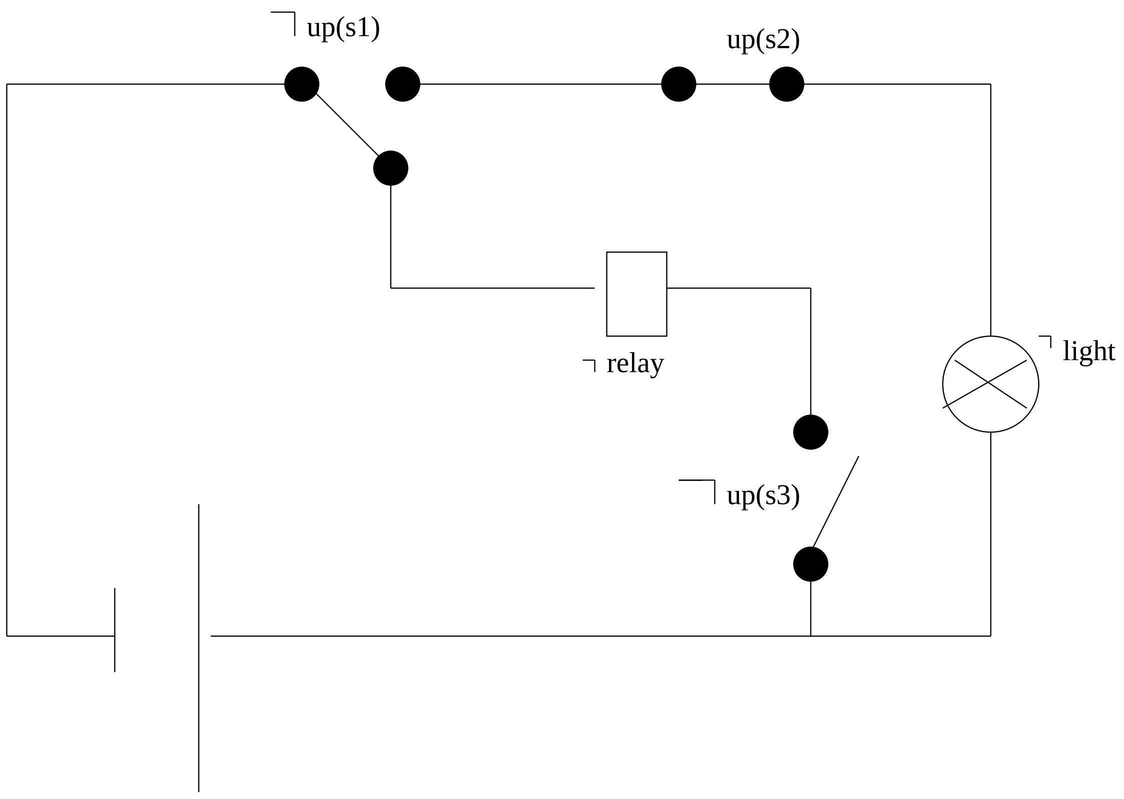

Figure 7: Thielscher's Circuit

In our example, the above change is right if and only if each of the fluents *light* and *relay* appear in a single constraint of the form $G_f \equiv K_f$. For example, consider the circuit in

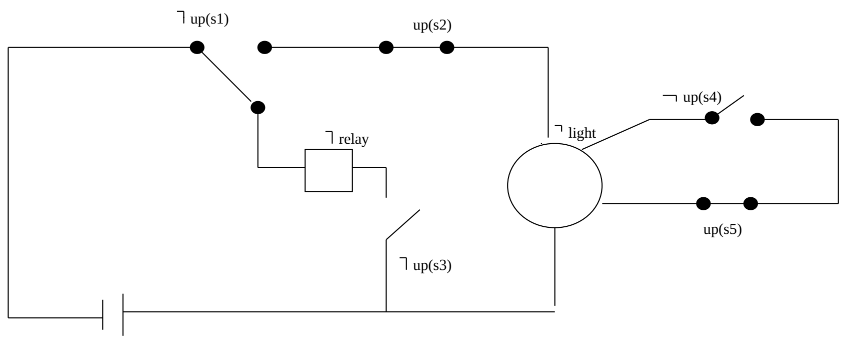

Figure 8: A more complex circuit

figure 8. The integrity constraints specifying the behavior of this system are expressed as the following formulae:

$$\begin{array}{ll}(a) & light \equiv up(s1) \wedge up(s2),\\ (b) & light \equiv up(s4) \wedge up(s5),\\ (c) & relay \equiv \neg up(s1) \wedge up(s3),\\ (d) & relay \supset \neg up(s2)\end{array}$$

Applying the procedure described above yields:

$$\begin{array}{l}((up(s1), light), (up(s2), light),\ (up(s4), light), (up(s5), light),\\ (up(s1), relay), (up(s3), relay),\ (relay, up(s2)) \in I\,.\end{array}$$

Assume that the circuit is in the situation depicted in figure 8. Then, after the execution of action $toggle\_switch(s_4)$, because $(up(s4), light) \in I$, the fluent $light$ changes from $\neg light$ to $light$. Because $(light, up(s1)), (light, up(s2)) \notin I$, the fluents $up(s1), up(s2)$ do not change. This means that the circuit will be in situation $\neg up(s1), up(s2), up(s4), up(s5), \neg up(s3), \neg relay$, $light$, which violates the rule (a). Assume now that the integrity constraints specifying the behavior of this system are expressed as the following formulae:

$$\begin{array}{ll}(a) & light \equiv (up(s1) \wedge up(s2)) \vee (up(s4) \wedge up(s5))\\ (b) & relay \equiv \neg up(s1) \wedge up(s3),\\ (c) & relay \supset \neg up(s2)\end{array}$$

In the above specification of constraints, the fluent $light$ is only in one constraint of type $G_f \equiv K_f$ and algorithm 2 behaves correctly. As we observe the circuit of the figure 8 consists of two smaller circuits. The first consists of the switches s_1, s_2 and the lamp, while the second of the switches s_4, s_5 and the lamp. The reasonable behavior is that the lamp lights up when one of the two circuit is closed. This is ensured by the second set of integrity constraints. The first set of integrity constraints ensures that when one circuit is closed then the second must be closed. This is not reasonable.

Theorem 4.1 *Let $A \supset B$ be a constraint and $C_1 \wedge .... \wedge C_n$ its CNF form. Both algorithms produce I in such 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

$$\begin{aligned}&\gamma_1 \vee ....\gamma_n\\ &\beta_1 \vee ..... \vee \beta_m\,,\end{aligned}$$

respectively.

Assume that

$$\begin{aligned}\gamma_1 &= f_{1_{\gamma_1}} \wedge ..... \wedge f_{n_{\gamma_1}}\\ &\dots\\ \gamma_n &= f_{1_{\gamma_n}} \wedge ..... \wedge f_{n_{\gamma_n}}\\ \beta_1 &= f_{1_{\beta_1}} \wedge ..... \wedge f_{n_{\beta_1}}\\ &\dots\\ \beta_m &= f_{1_{\beta_m}} \wedge ..... \wedge f_{n_{\beta_m}}\end{aligned}$$

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

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

This equivalent with the following:

$$\begin{aligned}&C_1 \wedge ..... \wedge C_w, \quad \mathit{where}\\ &(1) \quad w = n_{\gamma_1} * ...... * n_{\gamma_n} * n_{\beta_1} * ..... * n_{\beta_m}\\ &(2) \quad C_i = f_{i_1} \vee ...... \vee f_{i_n} \vee f_{i_{n+1}} \vee ... \vee f_{i_{n+m}}, \qquad \mathit{where}\end{aligned}$$

$$
\begin{aligned}
&f_{i_1} \in \gamma_1,\\
&\ldots\\
&f_{i_n} \in \gamma_n\\
&f_{i_{n+1}} \in \beta_1\\
&\ldots\\
&f_{i_{n+m}} \in \beta_m
\end{aligned}
$$

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

$$
\begin{aligned}
&\{(f_{i_1}, f_{i_{n+1}}), \ldots\ldots(f_{i_1}, f_{i_{n+m}})\} \subseteq I\\
&\ldots\\
&\{(f_{i_n}, f_{i_{n+1}}), \ldots\ldots(f_{i_n}, f_{i_{n+m}})\} \subseteq I
\end{aligned}
$$

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

■

Theorem 4.2 *Let $A \equiv B$ be a constraint and $C_1 \wedge .... \wedge C_n$ its CNF form. The algorithm 1 generates I in such 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 integrity constraint $A \equiv B$ is equivalent to $(A \supset B) \wedge (B \supset A)$. Suppose that

$$
\begin{aligned}
&C_1 \wedge \ldots \wedge C_w\\
&C'_1 \wedge \ldots \wedge C'_w
\end{aligned}
$$

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

$$C_1 \wedge \ldots \wedge C_w \wedge C'_1 \wedge \ldots \wedge C'_w$$

In order to transform into CNF form the above proposition we must examine if there is some pair (C_i, C_j) for which C_i "subsume" C_j $(C_i \equiv C_j \vee C'_i)$, 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.

■

The theorem 4.2 does not hold for algorithm 2 because it eliminates some pairs which the algorithm 1 produces. For example assume that the following two integrity constraints:

$$A \equiv B$$
$$A' \equiv B'$$

Assume that $A = f \wedge A_1$ and $A' \equiv f \wedge A_1'$, where fluent f cannot change as direct effect of an action. Then for all fluents f_i which there are in B or in B' and could change their truth value as direct effect of an action there is a pair $(f_i, f)i \in I$, while $(f, f_i) \notin I$. Then if an action has as effect the to become true the B, then A must become true. In order for this to happen f must become true. After that, B' must be true. But the fluent f cannot affect the truth value of any fluent in B'.

An example of such a situation is the following:

$$\begin{array}{ll} (a) & light \equiv up(s1) \wedge up(s2), \\ (b) & light \equiv up(s4) \wedge up(s5), \\ (c) & relay \equiv \neg up(s1) \wedge up(s3), \\ (d) & relay \supset \neg up(s2) \end{array}$$

(a) is equivalent to the successive formulas

$$\begin{array}{l} [light \supset up(s_1) \wedge up(s_2)] \wedge [up(s_1) \wedge up(s_2) \supset light] \\ (\neg light \vee (up(s1) \wedge up(s2))) \wedge (\neg up(s_1) \vee \neg up(s_2) \vee light), \\ (\neg light \vee up(s1)) \wedge (\neg light \wedge up(s2)) \;\; \wedge (\neg up(s_1) \vee \neg up(s_2) \vee light) \end{array}$$

The last formula is in CNF form. As we observe the rule has three parts

$$\begin{array}{ll} first: & \neg light \vee up(s1) \\ second: & \neg light \wedge up(s2) \\ third: & \neg up(s_1) \vee \neg up(s_2) \vee light \end{array}$$

If we apply the second algorithm, then from the first part the set I contains only the pair $(up(s_1), light)$ and not $(light, up(s_1))$. As we observe from the second integrity constraint the fluent light could change its truth value as result of the changing of truth value of the fluents $up(s_4)$ or $up(s_5)$. For the second rule we have

$$[light \supset up(s_4) \wedge up(s_5)] \wedge [up(s_4) \wedge up(s_5) \supset light]$$

The problem is that the following must hold:

$$up(s_4) \wedge up(s_5) \supset light \;\; (from \;\; the \;\; second \;\; integrity \;\; constraint) \;\; and$$
$$light \supset up(s_1) \wedge up(s_2) \;\; (from \;\; the \;\; first \;\; integrity \;\; constraint)$$

In that case we use the algorithm 1.

Above we identified the following situation as problematic for algorithm 2.

$$A \equiv B \quad and \quad C \supset B \;\; are \;\; ICs \;\; and \;\; A \neq C$$

where A,B,C are fluent formulas and B contains fluents which may change their truth value only as indirect effect of an action. The following result shows a weaker condition under which algorithm 2 is guaranteed to lead to an I which preserves consistency.

Theorem 4.3 *For each integrity constraint $A \equiv B$ if and fluents A_1 and B_1 which belong in the A and B respectively and could change their truth values only as indirect effects, the 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 in the A and B respectively and could change their 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 $D \neq B$.

Suppose that the initial situation satisfied the integrity constraint, and suppose that an update in the database occurs. There are two cases. First this update does not influence any fluents which belong to A and B. Then the integrity constraint is satisfied in the new situation. Second, this update changes the truth value of some fluents which belong to A or B. This change refers to fluents which belong to $A \setminus A_1$ or $B \setminus B_1$ because the fluents which belong to A_1 and B_1 could change their truth values only as indirect effects of an action and none 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 the algorithm 2.

4.1.2 Algorithms for the production of static rules

The static rules encapsulate the indirect effects of the execution of each action. The indirect effects exist because of the presence of integrity constraints. Hence, it is reasonable to produce the static rules from the integrity constraints. Initially the set of rules is $R = \{False \supset f,\ \ False \supset \neg f : for\ \ each\ \ fluent\ \ f\}$. Static rules are produced as follows:

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

2. For each i from 1 to n do

   (a) assume $C_i = f_1 \vee .... \vee f_m$

   ```
       For each j from 1 to m do
       For each k from 1 to m, and $k \neq j$, do
       if $(f_j, f_k) \in I$ then
         $R = R \cup (\neg f_j \quad causes \quad f_k \quad if \quad \bigwedge \neg f_l), l = 1, ..m, l \neq j, k.$
   ```

3. For each fluent f_k the rules [13] in R have the following form

$$\bigwedge f_i \quad causes \quad f_k \quad if \quad \Phi$$
$$\bigwedge f'_i \quad causes \quad \neg f_k \quad if \quad \Phi'.$$

   We successively incorporate these rules into exactly two, one for f_k and one for $\neg f_k$, as follows: if the current rules are

$$G \supset f_k$$
$$K \supset \neg f_k$$

   then we change them into

$$\begin{aligned} & G' \supset f_k \\ & K' \supset \neg f_k \\ & where \\ & G' = G \vee (\bigwedge f_i \wedge \Phi) \\ & K' = K \vee (\bigwedge f'_i \wedge \Phi'). \end{aligned}$$

[13]Notice that for a fluent there could be more than one causal relationship. This happens in the case that the fluent f_k appears in more than one integrity constraint. At this step we integrate all these causal relationships in 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 in $f_1 \wedge f_2 \wedge f_4 \supset f_3$ and $(f_1 \wedge f_2 \wedge f_4) \vee (f_5 \wedge f_6 \wedge f_4) \supset f_3$.

4. For each rule $G_p \supset f_p$, we replace each fluent f with $f(t)$, as we have defined above. The static rule has the form $G_p(t) \supset f_p(1)$.

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

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

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

Now we show how the algorithm works in the above example. First, we must construct the set I using the algorithm which has been proposed in section 4.1.1. All the integrity constraints (IC) have the form $A \supset B$. We have that

$$(illegal, suspended) \in I \quad (from \quad IC \quad 5)$$
$$(illegal, \neg take\_promotion) \in I \quad (from \quad IC \quad 6)$$
$$(suspended, \neg take\_salary) \in I \quad (from \quad IC \quad 7)$$
$$(\neg good\_employee, \neg take\_promotion) \in I \quad (from \quad IC \quad 8)$$
$$(\neg suspended, \neg take\_bonus) \in I \quad (from \quad IC \quad 9)$$
$$(good\_employee, \neg take\_bonus) \in I \quad (from \quad IC \quad 9)$$
$$(\neg good\_employee, take\_salary) \in I \quad (from \quad IC \quad 10)$$
$$(\neg suspended, take\_salary) \in I \quad (from \quad IC \quad 11)$$

The transformation of integrity constraints into CNF form yields:

$\neg illegal(p, t_1) \vee suspended(p, t_1)$ (5)
$\neg illegal(p, t_1) \vee \neg take\_promotion(p, t_1)$ (6)
$\neg suspended(p, t_1) \vee \neg take\_salary(p, t_1)$ (7)
$good\_employee(p, t_1) \vee \neg take\_promotion(p, t_1)$ (8)
$suspended(p, t_1) \vee \neg good\_employee(p, t_1) \vee \neg take\_bonus(p, t_1)$ (9)
$good\_employee(p, t_1) \vee \neg take\_bonus(p, t_1)$ (10)
$suspended(p, t_1) \vee take\_salary(p, t_1)$ (11)

This is step 1 of the algorithm. In step 2 we have

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

In the step 2 we estimate all causal relationships. In the step 3 we have that

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

In step 3 we construct for each fluent the fluents formula which makes the fluent true. Notice that perhaps there are many causal relationships which affect the same fluents. We integrate these causal relationship in this step. For example see the second and fourth causal relationships from step 2. In step 4

$$\begin{aligned}
&R = \{illegal(p,t_1) \supset suspended(p,1),\\
&illegal(p,t_1) \vee \neg good\_employee(p,t_2) \supset \ \neg take\_promotion(p,1)),\\
&suspended(p) \supset \neg take\_salary(p,1),\\
&\neg suspended(p,t_1) \wedge good\_employee(p,t_2) \supset take\_bonus(p,1),\\
&\neg good\_employee(p,t_1) \supset \neg take\_bonus(p,1),\\
&\neg suspended(p,t_1) \supset take\_salary(p,1)\}
\end{aligned}$$

Finally in step 5 (which must execute at each time point at which the static rules evaluate) we have the following four static rules:

$$\begin{aligned}
&R = \{illegal(p,t_1) \supset suspended(p,t_1),\\
&illegal(p,t_1) \vee \neg good\_employee(p,t_2) \supset \ \neg take\_promotion(p,max(t_1,t_2))\\
&suspended(p) \supset \neg take\_salary(p,t_1),\\
&\neg suspended(p,t_1) \wedge good\_employee(p,t_2) \supset take\_bonus(p,min(t_1,t_2)),\\
&\neg good\_employee(p,t_1) \supset \neg take\_bonus(p,t_1),\\
&\neg suspended(p,t_1) \supset take\_salary(p,t_1)\}
\end{aligned}$$

In step 5, for each static rule, we find the maximum time for which its body is true. For example in the second rule, the body is true when $illegal(t_1) \vee good\_employee(t_2)$ is true. This mean that we take the maximum of times t_1, t_2 for which $good\_employee$ is true. For the fourth rule, the body is true when $\neg suspended(t_1) \wedge good\_employee(t_2)$ is true. This means that we must take the minimum of times t_1, t_2.

By the production of the static rule we have that

Theorem 4.4 *When a static rule is executable at least one integrity constraint is not satisfied.*

Proof: Assume that a static rule

$$G_f(...) \supset f$$

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

$$\begin{aligned}
&G_f \equiv G_f^1 \vee \ldots G_f^w\\
&where\\
&G_f^i \equiv (\neg \bigwedge f_j(t_j, j = 1,..m)
\end{aligned}$$

Each G_f^i is produced by the integrity constraint $C_1 \wedge .... \wedge C_n$ such that there is a $C_i \equiv f \vee f_1 \vee \ldots \vee f_m$. Thus when the G_f^i is true it must be the case that all fluents $f_j, j = 1, ..m$ must be false. The above static rule is only executable when f is false. In that case $C_i \equiv f \vee f_1 \vee \ldots \vee f_m$ is false, thus the integrity constraint $C_1 \wedge .... \wedge C_n$ is not satisfied.

■

We study the case where the set of integrity constraints is satisfiable. For example if there are six integrity constraints of the form

$$\begin{aligned} & f_1 \supset f_2 \\ & f_2 \supset f_3 \\ & f_3 \supset \neg f_1 \\ & \neg f_1 \supset \neg f_2 \\ & \neg f_2 \supset \neg f_3 \\ & \neg f_3 \supset f_1 \end{aligned}$$

then there is no situation in which the above 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 [14]. By the above six constraints we produce six static rules which will be evaluated one after the other for ever. This happens because there is no consistent situation and thus always at least one static rule will be executable.

Theorem 4.5 *Consider a set of static rules G. Then if for a fluent f_1 there is a sequence*

$$\begin{aligned} & G_{f_2}(...) \supset f_2(...) \quad where \quad G_{f_2} \equiv f_1 \vee G'_{f_2} \\ & G_{f_3}(...) \supset f_3(...) \quad where \quad G_{f_3} \equiv f_2 \vee G'_{f_3} \\ & \ldots \\ & G_{f_n}(...) \supset f_n(...) \quad where \quad G_{f_n} \equiv f_{n-1} \vee G'_{f_n} \\ & G_{\neg f_1}(...) \supset \neg f_1(...) \quad where \quad G_{\neg f_1} \equiv f_n \vee G'_{\neg f_1} \\ & G_{f_{n+1}}(...) \supset f_{n+1}(...) \quad where \quad G_{f_{n+1}} \equiv \neg f_1 \vee G'_{f_{n+1}} \\ & \ldots \\ & G_{f_{n+m}}(...) \supset f_{n+m}(...) \quad where \quad G_{f_{n+m}} \equiv f_{n+m-1} \vee G'_{f_{n+m}} \\ & G_{f_1}(...) \supset f_1(...) \quad where \quad G_{f_1} \equiv f_{n+m} \vee G'_{f_1} \end{aligned}$$

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

then the set of rules is unsatisfiable.

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

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

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

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

$$\begin{array}{l} f_{n+m} \\ f_{n+m} \vee G'_{f_1} \supset f_1 \\ \hline f_1 \end{array}$$

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

■

For the rest of the thesis we assume that the set of the integrity constraints is satisfiable (this means that there is no sequence as in theorem 4.5). *This will be ensured if no static rule is executable in the initial situation at time point 0(by the theorem 4.4)).*

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

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

Each C_{ik} is a disjunct. 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.

Also in order for the set of the static rule R to be consistent, for each pair $(f, \neg f)$ it must hold that $G_f \wedge B_f \equiv FALSE$, when $G_f \supset f,\ \ B_f \supset \neg f$. In other cases, 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 4.6 *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.*

Proof: Assume that $G_f \wedge B_f$ is true. Also assume that initially the fluent f is true. Then the static rule $B_f \supset \neg f$ must be executed because the fluent formula B_f is true. After the execution the fluent $\neg f$ is true but the the static rule $G_f \supset f$ must be executed because the fluent formula G_f is true. After the execution the fluent f is true but the the static rule $B_f \supset \neg f$ must be executed because the fluent formula B_f is true. Thus the two above rules will be executed one after the other for infinite.

■

4.1.3 Algorithms for the Evaluation of Dynamic and Static Rules

In this section we present an algorithm for the evaluation of dynamic and static rules

1. After the execution of an action a evaluate the dynamic rule which refers to this action

   $$a(..) \supset f_1(t'_1) \wedge \ldots f_n(t'_n)$$

   and set $E = \{f_1, \ldots f_n\}$ [15].

2. Evaluate the default axioms

3. For all static rules except these that have as heading a fluent whose its negation belongs in the set E (this means that $G(..) \supset f(..)$ and $\neg f(..) \notin E$) do:

   (a) Repeat until no change occurs.

   (b) Evaluate the static rules corresponding to the fluents which are false at time point t.

   (c) If a fluent $f(t)$ becomes true after the evaluation of a static rule, then set $\neg f(0)$. (the negation is false).

4. For all static rules that have as heading a fluent whose negation belongs in the set E (this means that $G(..) \supset f(..)$ and $\neg f(..) \notin E$) do:

   (a) If none of these rules is executable then return the current situation, otherwise else repeat until no change occurs.

   (b) Evaluate the static rules corresponding to the fluents which are false at time point t.

   (c) If a fluent $f(t)$ becomes true after the evaluation of a static rule, then set $\neg f(0)$. (the negation is false).

5. For each time point t at which no action takes place do

   (a) Evaluate the default axioms

      i. Repeat until no change occurs.

      ii. Evaluate the static rules corresponding to the fluents which are false at time point t.

      iii. If a fluent $f(t)$ becomes true after the evaluation of a static rule, then set $\neg f(0)$. (the negation is false).

[15] The set E constrains the direct effects of action a

As we can observe in the above algorithm, if at time point t an action a take place then try to find a consistent situation which contains all the direct effects of a. If no such consistent situation [16] exists then try to find a consistent situation which does not contain all direct effects of the a. Notice that when the above algorithm terminates there are no executable static rules.

Consider the example with the public worker. We have four dynamic rules (1-4) as we have described in the previous section. Also we have produced the six static rules

$$\begin{aligned}
&illegal(p,t_1) \supset suspended(p,t_1),\\
&illegal(p,t_1) \vee \neg good\_employee(p,t_2) \supset \neg take\_promotion(p,max(t_1,t_2)),\\
&suspended(p,t_1) \supset \neg take\_salary(p,t_1),\\
&\neg suspended(p,t_1) \wedge good\_employee(p,t_2) \supset take\_bonus(p,min(t_1,t_2))\\
&\neg good\_employee(p,t_1) \supset \neg take\_bonus(p,t_1),\\
&\neg suspended(p,t_1) \supset take\_salary(p,t_1).
\end{aligned}$$

Assume now that we have a public worker p and the initial situation is

$$\begin{aligned}
&S_0 = \{\neg take\_bonus(p,\infty), take\_salary(p,\infty), \neg take\_promotion(p,\infty),\\
&\neg suspended(p,\infty), \neg good\_employee(p,\infty), \neg illegal(p,\infty)\}\,.
\end{aligned}$$

Time starts at 0 and has the granularity of months. Assume that the following actions occur at the following time points:

$$\begin{aligned}
&occur(good\_grade(p),2)\\
&occur(misdemeanor(p),4)\\
&occur(bad\_grade(p),6)\\
&occur(good\_grade(p),8)\\
&occur(misdemeanor(p),10)\\
&occur(take\_pardon(p),12)\,.
\end{aligned}$$

At time point 2 the action $good\_grade(p)$ executes. From the algorithm for the evaluation of dynamic and static rules, after the evaluation of dynamic rule (4) we have the situation

[16]This happens when a static rule which has as conclusion an effect which is inconsistent with the direct effects is executable and none other static rule which has as conclusion an effect which is consistent with the direct effects is executable. In that case the resulting situation is inconsistent (because by the production of the static rules, we have that when a static rule is executable in a situation S there is an integrity constraint which is not satisfiable in S). Thus the evaluation of the static rules must go on until no static rule be executable.

$$S'_1 = \{\neg take\_bonus(p,\infty), take\_salary(p,\infty),\ \neg take\_promotion(p,\infty),$$
$$\neg suspended(p,\infty), good\_employee(p,\infty), \neg illegal(p,\infty)\}\,.$$

As we csn observe the following static rule will be evaluated:

$$\neg suspended(p,\infty) \wedge good\_employee(p,t_2) \supset\ take\_bonus(p,\infty)\,,$$

After the evaluation of static rule we have the situation

$$S_1 = \{take\_bonus(p,\infty), take\_salary(p,\infty),\ \neg take\_promotion(p,\infty),$$
$$\neg suspended(p,\infty), good\_employee(p,\infty), \neg illegal(p,\infty)\}\,.$$

This situation does not change until time point 4, when the second action ($misdemeanor(p)$) takes place. From the algorithm for the evaluation of dynamic and static rules, after the evaluation of dynamic rule (1) we have the situation

$$S'_2 = \{take\_bonus(p,\infty), take\_salary(p,\infty),\ \neg take\_promotion(p,\infty),$$
$$\neg suspended(p,\infty), good\_employee(p,\infty), illegal(p,5)\}\,.$$

As we may observe the following static rules will be evaluated

$$illegal(p,5) \supset suspended(p,5)$$
$$suspended(p,5) \supset \neg take\_salary(p,5)$$

After the evaluation of static rules we have the situation

$$S_2 = \{take\_bonus(p,\infty), \neg take\_salary(p,5),\ \neg take\_promotion(p,\infty),$$
$$suspended(p,5), good\_employee(p,\infty), illegal(p,5)\}\,.$$

This situation does not change until the time point 6, when the third action ($bad\_grade(p)$) executes. From the algorithm for the evaluation of dynamic and static rules, after the evaluation of dynamic rule (3) we have the situation

$$S'_3 = \{take\_bonus(p,\infty), \neg take\_salary(p,5),\ \neg take\_promotion(p,\infty),$$
$$suspended(p,5), \neg good\_employee(p,\infty), illegal(p,5)\}\,.$$

The following static rule will be evaluated:

$$\neg good\_employee(p,\infty) \supset \neg take\_bonus(p,\infty)$$

After the evaluation of static rule we have the situation

$$S_3 = \{\neg take\_bonus(p,\infty), \neg take\_salary(p,3),\ \neg take\_promotion(p,\infty),$$
$$suspended(p,3), \neg good\_employee(p,\infty), illegal(p,3)\}\,.$$

This situation does not change until the time point 6, when the fourth action (*good_grade*) takes place. From the algorithm for the evaluation of dynamic and static rules after the evaluation of dynamic rule (4) we have the situation

$$S'_4 = \{\neg take\_bonus(p,\infty), \neg take\_salary(p,1),\ \neg take\_promotion(p,\infty),$$
$$suspended(p,1), good\_employee(p,\infty), illegal(p,1)\}\,.$$

No static rule will be evaluated. Thus the situation does not change. At time point 9 does not executed any action but the situation change because $take\_salary(p,0) \wedge take\_salary(p,0)$, $suspend(p,0) \wedge \neg suspend(p,0)$ and $illegal(p,0) \wedge \neg illegal(p,0)$ holds. Thus the following default axioms are evaluated

$$\neg suspended(p,0) \wedge suspended(p,0) \supset \neg suspended(p,\infty)$$
$$\neg take\_salary(p,0) \wedge take\_salary(p,0) \supset take\_salary(p,\infty)$$
$$illegal(p,0) \wedge \neg illegal(p,0) \supset \neg illegal(p,\infty)$$

The new situation is

$$S'_5 = \{\neg take\_bonus(p,\infty), take\_salary(p,\infty),\ \neg take\_promotion(p,\infty),$$
$$\neg suspended(p,\infty), good\_employee(p,\infty), \neg illegal(p,\infty)\}\,.$$

Now the static rule

$$\neg suspended(p,\infty) \wedge good\_employee(p,\infty) \supset \; take\_bonus(p,\infty)\,,$$

will be evaluated. After the evaluation of the static rule the situation is

$$S_5 = \{take\_bonus(p,\infty), take\_salary(p,\infty),\; \neg take\_promotion(p,\infty),$$
$$\neg suspended(p,\infty), good\_employee(p,\infty), \neg illegal(p,\infty)\}\,.$$

At time point 10 the action *misdemeanor*(p) executes, thus the situation changes into

$$S'_6 = \{take\_bonus(p,\infty), take\_salary(p,\infty),\; \neg take\_promotion(p,\infty),$$
$$\neg suspended(p,\infty), good\_employee(p,\infty), illegal(p,5)\}\,.$$

Now the following static rules will be evaluated

$$illegal(p,5) \supset suspended(p,5)$$
$$suspended(p,5) \supset \neg take\_salary(p,5)$$

After the evaluation of static rules the situation is

$$S_6 = \{take\_bonus(p,\infty), \neg take\_salary(p,5),\; \neg take\_promotion(p,\infty),$$
$$suspended(p,5), good\_employee(p,\infty), \neg illegal(p,5)\}\,.$$

The last action (*take_pardon*) occurs at time point 12. The new situation is

$$S'_7 = \{take\_bonus(p,\infty), \neg take\_salary(p,3),\; \neg take\_promotion(p,\infty),$$
$$suspended(p,3), good\_employee(p,\infty), \neg illegal(p,\infty)\}\,.$$

Finally the situation changes again at time point 15, because $\neg suspended(p,0) \wedge suspended(p,0)$ and $\neg take\_salary(p,0) \wedge take\_salary(p,0)$ holds. Thus the following default axioms are evaluated

$$\neg suspended(p,0) \wedge suspended(p,0) \supset \neg suspended(p,\infty)$$
$$\neg take\_salary(p,0) \wedge take\_salary(p,0) \supset take\_salary(p,\infty)$$

Now the new situation is

$$S_8 = \{take\_bonus(p,\infty), take\_salary(p,\infty),\ \neg take\_promotion(p,\infty),$$
$$\neg suspended(p,\infty), good\_employee(p,\infty), \neg illegal(p,\infty)\}\,.$$

This is the end of execution.

As we observe from the set R, for each pair $(f, \neg f)$ it holds that $G_f \wedge K_f \equiv FALSE$, when $G_f \supset f$, $\ K_f \supset \neg f$. More specifically the set of static rules is

$$\begin{aligned}
R = \{&illegal(p,t_1) \supset suspended(p,t_1),\\
&illegal(p,t_1) \vee \neg good\_employee(p,t_2) \supset\ \neg take\_promotion(p,max(t_1,t_2)\\
&suspended(p,t_1) \supset \neg take\_salary(p,t_1),\\
&\neg suspended(p,t_1) \wedge good\_employee(p,t_2) \supset take\_bonus(p,min(t_1,t_2)),\\
&\neg good\_employee(p,t_1) \supset \neg take\_bonus(p,t_1),\\
&\neg suspended(p,t_1) \supset take\_salary(p,t_1),\\
&good\_employee(p,t_1) \supset take\_promotion(p,t_1),\\
&False \supset \neg suspended(p,t_1),\\
&False \supset take\_promotion(p,t_1),\\
&False \supset illegal(p,t_1),\\
&False \supset \neg illegal(p,t_1),\}
\end{aligned}$$

As we observe, for the fluents for which there is no static rule we add the rule $false \supset f$, because they cannot become true by the static rule but only by the dynamic rules (this means that the truth value changes only as direct effect of some action). Now we have

$$\begin{aligned}
&(\neg suspended(p,t_1) \wedge good\_employee(p,t_2)) \wedge \neg good\_employee(p,t_1)\\
&\qquad for\ \ (take\_bonus, \neg take\_bonus)\\
&suspended(p,t_1) \wedge \neg suspended(p,t_1)\\
&\qquad for\ \ (take\_salary(p,t_1), \neg take\_salary(p,t_1))\\
&illegal(p,t_1) \wedge False \quad for\ \ (suspended(p,t_1), \neg suspended(p,t_1))
\end{aligned}$$

$$\begin{aligned}&illegal(p) \vee \neg good\_employee(p) \wedge False\\ &\qquad\qquad for \quad (\neg take\_promotion(p), take\_promotion(p)\\ &False \wedge False \quad for \quad (illegal(p), \neg illegal(p))\end{aligned}$$

The assumption $G_f \wedge K_f \equiv FALSE$ is very important in order to ensure that, always, after the execution of action there is a consistent situation. Now we show with an example that if this assumption does not hold there is no consistent situation after some sequence of action execution.

Consider the example with the public worker and assume that there is another integrity constraint specifying that when a public worker is a good employee then s/he can take promotion. Now the set of static rules is

$$\begin{aligned}&R = \{illegal(p,t_1) \supset suspended(p,t_1),\\ &illegal(p,t_1) \vee \neg good\_employee(p,t_2) \supset \ \neg take\_promotion(p, max(t_1,t_2)\\ &suspended(p,t_1) \supset \neg take\_salary(p,t_1),\\ &\neg suspended(p,t_1) \wedge good\_employee(p,t_2) \supset take\_bonus(p, min(t_1,t_2)),\\ &\neg good\_employee(p,t_1) \supset \neg take\_bonus(p,t_1),\\ &\neg suspended(p,t_1) \supset take\_salary(p,t_1),\\ &good\_employee(p,t_1) \supset take\_promotion(p,t_1)\}\end{aligned}$$

As we can observe, for the pair $(take\_promotion(p), \neg take\_promotion(p)$ the above assumption does not hold, because

$$good\_employee(p,t_1) \wedge \ (illegal(p,t_1) \vee \neg good\_employee(p,t_2))$$

can be true when $good\_employee(p) \wedge illegal(p)$ holds.

Assume now that we have a public worker p and the initial situation is

$$\begin{aligned}&S_0 = \{\neg take\_bonus(p,\infty), take\_salary(p,\infty), \neg take\_promotion(p,\infty),\\ &\neg suspended(p,\infty), \neg good\_employee(p,\infty), \neg illegal(p,\infty)\}\,.\end{aligned}$$

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

$$\begin{aligned}&occur(misdemeanor(p), 4)\\ &occur(good\_grade(p), 6)\,.\end{aligned}$$

At time point 4 after the execution of the action $misdemeanor(p)$ we have the situation

$$S_1' = \{\neg \mathit{take\_bonus}(p,\infty),\ \mathit{take\_salary}(p,\infty), \neg \mathit{take\_promotion}(p,\infty),$$
$$\neg \mathit{suspended}(p,\infty), \neg \mathit{good\_employee}(p,\infty),\ \mathit{illegal}(p,\infty), \}$$

After the evaluation of the static rules we have

$$S_1 = \{\neg \mathit{take\_bonus}(p,\infty),\ \neg \mathit{take\_salary}(p,\infty), \neg \mathit{take\_promotion}(p,\infty),$$
$$\mathit{suspended}(p,\infty), \neg \mathit{good\_employee}(p,\infty),\ \mathit{illegal}(p,\infty), \}$$

At time point 6, after the execution of the action *good_grade*(p) we have the situation

$$S_2' = \{\neg \mathit{take\_bonus}(p,\infty),\ \neg \mathit{take\_salary}(p,\infty), \neg \mathit{take\_promotion}(p,\infty),$$
$$\mathit{suspend}(p,\infty), \mathit{good\_employee}(p,\infty),\ \mathit{illegal}(p,\infty), \}$$

Now the static rule $\mathit{good\_employee}(p,\infty) \supset \mathit{take\_promotion}(p,\infty)$ must be evaluated and after that $\mathit{take\_promotion}(p,\infty)$ must hold. But if $\mathit{take\_promotion}(p,\infty)$ holds then we must examine if the static rule $\mathit{illegal}(p,\infty) \vee \neg \mathit{good\_employee}(p,t_1) \supset \neg \mathit{take\_promotion}(p,\infty)$ must be evaluated. We observe that we must evaluate this static rule. As we may observe the two static rules will be evaluated one after the other for ever. This means that there is no consistent situation. This happens because there is a mistake in the integrity constraints, which has as result that the above assumption does not hold.

The algorithm can run without the above assumption but we must determine the preconditions of each action in order to avoid the above problem.

The following theorem establishes the termination of the algorithm.

Theorem 4.7 *At each time unit, the algorithms terminate at a finite number of step.*

Proof: 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 $S_t^0, S_t^1, ...S_t^k, ...$ [17]

In this proof the term situation refers to the set of the truth values of the fluents. Thus the transition from one situation to the next happens only when a fluent changes its truth value. [18]. Notice that because a static rule is evaluated only when the corresponding fluent is false, it is not possible for a static rule $G(t,t') \supset f(t')$ to be evaluated when the fluent f is true

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

[18]This mean that each $S_t^i = \mathit{FluentHold}(S_m, t)$, for a temporal situation S_m

at point t. The static rule will be evaluated when f becomes false. Thus, for the transaction from one situation that the other it is necessary to there is a fluent f which changes from f to $\neg f$.

If F is the number of fluents then there are 2^F different situations (because the transaction from one situation to the next happens only if a fluent changes its truth value.) This means that in the above sequence, there are two identical situations because of the infinite loop. Without loss of generality we assume $S_t^l = S_t^k$.

Thus in the sequence $S_t^l, ..., S_t^k$ 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.

Assume that

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

Assume that first f' holds. Then we must evaluate the rule $B(t, t'') \supset \neg f'$ and after the $G(t, t') \supset f'$. This means that 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 the theorem 4.5 describes. In that case the integrity constraints are unsatisfiable. A contraction (the initial situation satisfies all integrity constraints).

Hence, in each case we reach a contradiction and thus the algorithm terminate at a finite number of step.

■

The following theorem establishes that we always end up with a consistent situation.

Theorem 4.8 *The above algorithm always return a legal situation.*

Proof: In order for the algorithm to be correct must always terminate in a consistent situation. This means that all integrity constraints must be satisfied at this situation.

Assume that integrity constraint Law_j is not satisfied in one situation. Assume that the CNF of this law is $C_1 \wedge .... \wedge C_n$. Then it must be the case that one of the $C_1, ..., C_n$ is false. Assume that $C_i = f_1 \vee .... \vee f_m$ is false. Then all fluents $f_j, j = 1, ..m$ are false. Assume that f_k and f_p are two of these for which $(f_k, f_p) \in I$ [19]. Then for f_p it must be the case that: $G_p(t) = G' \vee (\neg \bigwedge f_j(t_j, j = 1,..m, j \neq p)$. If all fluents $f_j, j = 1,..m$ are false then $(\neg \bigwedge f_j, j = 1,..m, j \neq p)$ is true. Thus $G_p(t, t')$ is true and $t' \geq 1$. This means that the static rule $G_p(t, t') \supset f_p(t')$ must be evaluated and thus, the f_p is true. A contradiction.

■

The complexity of the above solution is $O(A + 2 * F))$ laws, where A is the number of action an F the number of fluents..

4.1.4 The ramification problem when the direct and indirect effects of an action refer only to future situations

The ramification problem becomes more complex when the direct and indirect effects of an action do not hold for the next time moment but after some time moments. For example, assume that in the above example the action *good_grade* has as direct effect to characterize the public worker as good employee after two months (respectively for the action *bad_grade*). In that case the above representation of fluents cannot encapsulate the direct and indirect effects of the actions *good_grade*, *bad_grade*.

In order to accomodate such cases we extend futher the situation calculus

- We change the representation of fluents as follows: each fluent F is represented as $F(L)$, where $L = [[t, t'], ...]$ is a list. Each member of the list is a time interval $[t, t']$, which means that the fluent is true at time interval $[t, t']$. At each time moment t'' we reduce for the list the time intervals $[t, t']$ which refer in the past$(t'' > t')$ [20]. Now the rules (dynamic and static) change in order to encapsulate the above change.
- We define the function $FluentHold(S, t)$ which returns the set of all fluents which are true in the time moment t. [21]

The static rules are produced from the same algorithm as before except from step five which changes as follows: At each time moment it which it is necessary to evaluate a static rule, before the evaluation of the rule execute the following algorithm

1. At time moment t, for the static rule $G(t, t_1) \supset f$ do
   (a) let $G = G_1 \vee .... \vee G_n$

[19] The theorems 4.1,4.2,4.3 ensuring that there is a such pair

[20] For example at time point 5 we reduce the time interval $[2, 4]$ because $5 > 4$.

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

(b) For each j from 1 to n do

  - let $G_i = f_1([..]) \wedge .... \wedge f_n([..])$
    i. for each fluent $f_i(L)$ take the first element $[t^{'}, t^{''}]$ of the list L.
    ii. if $t^{'} > t$ then G is false; terminate.
    iii. else $t_i = t^{''} - t$.
  - let $t_{min} = min(t_1, ..., t_n)$
  - replace G_i with $G_i(t, t + t_{min})$

When a static rule $G(t, t^{'}) \supset f(L)$ is evaluated the element $[t, t^{'}]$ is added in the list L and is removed from the L', where $\neg f(L')$ [22]. For the rest of this thesis the notation $G(t, [t, t'])$ is equivalent to $G(t, t')$. The algorithm for adding an element $[t, t^{'}]$ add in the list L and removing it from L' is:

1. Consider the fluent f(L). L is the list at which we add the element $[t, t^{'}]$.
2. if there is an element $[t_{i1}, t_{i2}]$ for which $t_{i1} < t < t_{i2} < t^{'}$ holds then remove it and add $[t_{i1}, t^{'}]$.
3. if there is an element $[t_{i1}, t_{i2}]$ for which $t < t_{i1} < t^{'} < t_{i2}$ holds then remove it and add $[t, t_{i2}]$
4. else add $[t, t^{'}]$.
5. Consider $\neg f(L')$. L' is the list from which we remove the element $[t, t^{'}]$.
6. if there is an element $[t_{i1}, t_{i2}]$ for which $t < t_{i1} < t_{i2} < t^{'}$ holds then remove it.
7. if there is an element $[t_{i1}, t_{i2}]$ for which $t_{i1} < t < t^{'} < t_{i2}$ holds then remove it and add $[t_{i1}, t]$ and $[t^{'}, t_{i2}]$.
8. if there is an element $[t_{i1}, t_{i2}]$ for which $t < t_{i1} < t^{'} < t_{i2}$ holds then remove it and add $[t^{'}, t_{i2}]$.
9. if there is an element $[t_{i1}, t_{i2}]$ for which $t_{i1} < t < t_{i2} < t^{'}$ holds then remove it and add $[t_{i1}, t]$.

The above algorithm can be used for the evaluation of dynamic rules, too. The algorithm for evaluating the dynamic and static rules does not change, except that now there is no need to evaluate the default axioms. Notice that the algorithm which estimates the indirect effects of an action, does that at the time that they start to hold. This means that if some action has direct effects which refer only in the future, the indirect effects are not estimated at the time of

[22] The list L' is the list, which contains the time intervals, in which the fluent $\neg f$ is true

action execution but at the time that the direct effects start to hold. This has the advantage that if in the interval between the execution of the action and the time point at which the effects start to hold some action which cancels the direct effects of the first action occurs, it is not necessary to estimate the indirect effects twice.

Theorem 4.9 *The algorithm for evaluating the dynamic and static rules returns a legal situation when the direct and indirect effects refer only to future situation.*

Proof: The proof is similar with the proof of the theorem 4.8. The proof is similar because the algorithm which estimates the indirect effects of an action, does that at the time that they start to hold.

■

Consider the example with the public worker but with the new assumptions (which we made at the begin of the section) for the actions $good\_grade, bad\_grade$. Assume that the direct effect of the actions $misdemeanor, take\_pardon$ hold for the current moment of the execution of actions. Now the dynamic rules are:

$$\begin{aligned}
&occur(misdemeanor(p), t) \supset illegal(p, [t, t+5m]) \quad (1')\\
&occur(take\_pardon(p), t) \supset \neg illegal(p, [t, \infty]) \quad (2')\\
&occur(bad\_grade(p), t) \supset \neg good\_employee(p, [t+2, \infty]) \quad (3')\\
&occur(good\_grade(p), t) \supset good\_employee(p, [t+2, \infty]) \quad (4')\,.
\end{aligned}$$

Now assume that the initial situation is:

$$\begin{aligned}
&S_0 = \{\neg take\_bonus(p, [[0, \infty]]), take\_salary(p, [[0, \infty]]),\\
&\neg take\_promotion(p, [[0, \infty]]),\ \neg suspended(p, [[0, \infty]]),\\
&\neg good\_employee(p, [[0, \infty]]), \neg illegal(p, [[0, \infty]])\}\,.
\end{aligned}$$

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

$$\begin{aligned}
&occur(good\_grade(p), 2)\\
&occur(misdemeanor(p), 4)\\
&occur(bad\_grade(p), 8)\,.
\end{aligned}$$

At time point 2 the first action will be executed. After the execution the new situation is

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

As we observe at time point 2 no static rule will be evaluated (contrary to what happens in the previous sections example) because the effects of action *good_grade*(p) will holds two time points latter.

At time point 4 the second action will be executed and the effects of the first action start to hold. Now the new situation is

$$\begin{aligned}&S'_2 = \{\neg take\_bonus(p, [[0, \infty]]),\ take\_salary(p, [[0, \infty]]),\\ &\neg take\_promotion(p, [[0, \infty]]),\ \neg suspended(p, [[0, \infty]]),\\ &good\_employee(p, [[4, \infty]]),\ illegal(p, [[4, 9]]),\ \neg illegal(p, [[9, \infty]])\}\,.\end{aligned}$$

As we observe the $\neg good\_employee(p, [[2, 4]])$ does not hold at time point 4 and thus we remove it. [23] The following static rules will be evaluated

$$\begin{aligned}&illegal(p, [4, 9]) \supset suspended(p, [4, 9])\\ &suspended(p, [4, 9]) \supset \neg take\_salary(p, [4, 9])\end{aligned}$$

After the evaluation of the static rules we have the situation

$$\begin{aligned}&S_2 = \{\neg take\_bonus(p, [[0, \infty]]), \neg take\_salary(p, [[4, 9]]),\\ &take\_salary(p, [[9, \infty]]),\ \neg take\_promotion(p, [[0, \infty]]),\\ &suspended(p, [[4, 9]]),\ \neg suspended(p, [[9, \infty]]),\\ &good\_employee(p, [[4, \infty]]),\ illegal(p, [[4, 9]]),\ \neg illegal(p, [[9, \infty]])\}\,.\end{aligned}$$

As we observe the fluents $suspended(p, [[4, 9]])$, $\neg suspended(p, [[9, \infty]])$ and $\neg take\_salary(p, [[4, 9]])$, $take\_salary(p, [[9, \infty]])$ and $illegal(p, [[4, 9]])$, $\neg illegal(p, [[9, \infty]])$ encapsulate the default axioms.

At time point 8 the third action will be executed. After the execution the new situation is

[23]We observe with this representation of fluents we can capture the meaning of default axioms. For example the fluent *illegal*, $\neg$*illegal*.

$$\begin{aligned}S_3 = \{&\neg take\_bonus(p,[[0,\infty]]), \neg take\_salary(p,[[4,9]]),\\ &take\_salary(p,[[9,\infty]]),\ \neg take\_promotion(p,[[0,\infty]]),\\ &suspended(p,[[4,9]]), \neg suspended(p,[[9,\infty]]),\\ &good\_employee(p,[[4,10]]),\ \neg good\_employee(p,[[10,\infty]]),\\ &illegal(p,[[4,9]]),\ \neg illegal(p,[[9,\infty]])\}\,.\end{aligned}$$

At time point 9 no action takes place but the situation changes, because the fluents $illegal(p,[[4,9]])$, $suspended(p,[[4,9]])$ and $\neg take_salary(p,[[4,9]])$ cease to hold.

$$\begin{aligned}S'_4 = \{&\neg take\_bonus(p,[[0,\infty]]), take\_salary(p,[[9,\infty]]),\\ &\neg take\_promotion(p,[[0,\infty]]),\ \neg suspended(p,[[9,\infty]]),\\ &good\_employee(p,[[4,10]]),\ \neg good\_employee(p,[[10,\infty]]),\ \neg illegal(p,[[9,\infty]])\}\,.\end{aligned}$$

Now the static rule

$$\neg suspended(p,[9,\infty]) \wedge good\_employee(p,[9,10]) \supset\ take\_bonus(p,[9,10])$$

must evaluated. Thus the new situation at time point 9 is

$$\begin{aligned}S_4 = \{&take\_bonus(p,[[9,10]]), \neg take\_bonus(p,[[10,\infty]]),\\ &take\_salary(p,[[9,\infty]]),\ \neg take\_promotion(p,[[0,\infty]]),\\ &\neg suspended(p,[[9,\infty]]),\ good\_employee(p,[[4,10]]),\\ &\neg good\_employee(p,[[10,\infty]]), \neg illegal(p,[[9,\infty]])\}\,.\end{aligned}$$

At time point 10 no action will execute but the situation changes as follows:

$$\begin{aligned}S_5 = \{&\neg take\_bonus(p,[[10,\infty]]), take\_salary(p,[[9,\infty]]),\\ &\neg take\_promotion(p,[[0,\infty]]),\ \neg suspended(p,[[9,\infty]]),\\ &\neg good\_employee(p,[[10,\infty]]),\ \neg illegal(p,[[9,\infty]])\}\,.\end{aligned}$$

The transition from the situation S_4 to S_5 happens because $good\_employee(p,\ [[4,10]])$ and $take\_bonus(p,[9,10])$ cease to hold at time point 10.

4.1.5 The ramification problem when actions have duration

In the case that the actions have duration then all effects must be determined with reference to the start, the end and the duration of the actions. If all direct and indirect effects can be described by reference to the start and the end of the action then we can assume that one action with duration is equivalent with two instantaneous actions: one for the start and one for the end. In that case the dynamic rules must be defined for the instantaneous actions. The above algorithms solve the ramification problem without change. But as we show below this is a very strict assumption.

In the case that the effects of an action depend on its duration, the above approach cannot address the ramification problem. Consider the example with the public worker and assume that if the action *good_grade* has a duration of more than two time moments then it has as effect the promotion of the employee.

Usually the duration of an action is unknown before its end. So we cannot describe the direct and indirect effects of an action with reference to the start and to the end. We must change the dynamic and static rules.

The fluents representation does not change. For each action a we define a new functional fluent $f_\alpha(a)$ which returns the duration of the execution of action a until the current moment. If this fluent returns 0 the action does not execute at current moment.

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

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

The above algorithm "adds" at each static rule the effect which depends on the duration of an action. The algorithm of evaluation of the dynamic and static rules does not change.

The main problem with actions with duration is that we cannot assume that each of them is equivalent with two instantaneous actions one at the start and one at the end of the duration. We show that with an example. Consider again the example with the *public_wcorker*. Assume that the action *misdemeanor*(p) has duration and the direct effect is that the *public_wcorker* p is *illegal* for time of the duration of the action *misdemeanor*(p) and also for 5 months after the end of the action. Because we may not know the duration of the action *misdemeanor*(p) from the start we must define the following dynamic rules

$$\begin{aligned}
&occur(start(misdemeanor(p)), t) \supset illegal(p, [t, \infty]) \quad (a)\\
&occur(end(misdemeanor(p)), t) \supset illegal(p, [t, t+5m]) \wedge\\
&\qquad \neg illegal(p, [t+5, \infty]) \qquad (b)
\end{aligned}$$

The justification for the introduction of the two rules comes from the fact that when the execution of action *misdemeanor*(p) starts, the public worker is *illegal* identically, because we do not know when the action ends. After the end of the action and the public worker is considered *illegal* for 5 months. This is right if the fluent *illegal*$(p, [t, \infty])$ does not have as indirect effect to change the truth value of another fluent. For example assume the initial situation is

$$\begin{aligned}
S_0 = \{&\neg take\_bonus(p, [0, \infty]),\ take\_salary(p, [0, \infty]),\\
&\neg take\_promotion(p, [0, \infty]),\ \neg suspended(p, [0, \infty]),\\
&\neg good\_employee(p, [0, \infty]),\ \neg illegal(p, [0, \infty])\}\,.
\end{aligned}$$

Also assume the following execution of actions

$$\begin{aligned}
&occur(start(misdemeanor(p)), 2)\\
&occur(end(misdemeanor(p)), 4)
\end{aligned}$$

At time point 2 the execution of action *misdemeanor*(p) starts and thus the dynamic rule (a) is evaluated. The new situation is

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

After the evaluation of the static rules

$$\begin{aligned}&\mathit{illegal}(p,[2,\infty]) \supset \mathit{suspended}(p,[2,\infty])\\ &\mathit{suspended}(p,[2,\infty]) \supset \neg \mathit{take\_salary}(p,[2,\infty])\end{aligned}$$

we have the situation

$$\begin{aligned}&S_1 = \{\neg \mathit{take\_bonus}(p,[0,\infty]),\ \neg \mathit{take\_salary}(p,[2,\infty]),\\ &\neg \mathit{take\_promotion}(p,[0,\infty]),\ \mathit{suspended}(p,[2,\infty]),\\ &\neg \mathit{good\_employee}(p,[0,\infty]),\ \mathit{illegal}(p,[2,\infty])\}\,.\end{aligned}$$

At time point 4 the execution of action *misdemeanor*(p) ends and the dynamic rule (b) is evaluated. The new situation is

$$\begin{aligned}&S_2 = \{\neg \mathit{take\_bonus}(p,[0,\infty]),\ \neg \mathit{take\_salary}(p,[2,\infty]),\\ &\neg \mathit{take\_promotion}(p,[0,\infty]),\ \mathit{suspended}(p,[2,\infty]),\\ &\neg \mathit{good\_employee}(p,[0,\infty]),\ \mathit{illegal}(p,[4,9]), \neg \mathit{illegal}(p,[9,\infty])\}\,.\end{aligned}$$

As we can observe the fluent *illegal* changes but the fluent $\mathit{suspended}(p,[2,\infty])$ does not change. Thus at time point 9 the fluent $\mathit{suspended}(p,[2,\infty])$ still holds and thus the fluent $\neg \mathit{take\_salary}(p,[2,\infty])$ holds. This is wrong because one expects the fluent *suspended* to cease to hold when illegal ceases to hold. This does not happen. The problem is caused by the dynamic rule which refers to the start of some action. More specifically, the problem is caused by the assumption that each fluent that the specific action makes true, is assumed to be true indefinitely, because we do not known the end of the action. Thus we must change the dynamic rule which refers to the start and similarly with the dynamic rule which refers to the end of the action. Now there is the problem of how to refresh the direct effects of the action as time progress.

In order to solve this problem, we define a natural action *natural*(a) for each action a with duration. This natural action is instantaneous and will be executed periodically. The direct effects of a natural action for action a are exactly the same with the effects of action a. This means that if $\mathit{occur}(a,t) \supset \bigwedge f_i([t,t'_i])$ then $\mathit{occur}(\mathit{natural}(a),t) \supset \bigwedge f_i([t,t'_i])$ The period is equal with the minimal time that some fluent become trues (as direct effect of action a) after the execution of this action. For example,

$$\mathit{occur}(a,t) \supset\ f_1([t,t+3]) \wedge f_2([t,t+4]) \wedge f_3([t,t+7])\,,$$

then the period of execution is 3. The precondition of the action $natural(a)$ is the continuation of the execution of action a. This means

$$Poss(natural(a)) \equiv f_\alpha(a) > 0\,.$$

For each action a with duration, we assume that there are three instantaneous actions $start(a)$, $end(a)$ and $natural(a)$. The dynamic rules must refer to the instantaneous actions so we change the dynamic rules as follows:

1. if the dynamic rule has the form $occur(A,t) \supset \bigwedge f_i([t,t'_i])$ and $t' \neq \infty$ for some f_i then
   (a) replace it with

$$occur(start(A),t) \supset \bigwedge f_i([t,t'_i])$$
$$occur(end(A),t) \supset \bigwedge f_i([t,t'_i])$$

   (b) let $t'' = min\{t'_i : f_i([t,t'_i])\}$
   (c) set natural(A) to execute periodically with period t''.
2. if the dynamic rule has the form $occur(A,t) \supset \bigwedge f_i([t,\infty])$ then replace it with

$$occur(start(A),t) \supset \bigwedge f_i([t,\infty])$$

The algorithm for the evaluation of dynamic and static rules does not change. As we observe if one action has "permanent" direct effects then it is not necessary to define natural action for this action, because there is no need to refresh.

Theorem 4.10 *The algorithm address the ramification problem in case that the effect of an action depend on its duration.*

Proof: In order to be correct the algorithm must always terminate in a consistent situation. This mean sthat all integrity constraint must be satisfied at this situation.

Assume that the integrity constraint Law_j is not satisfied in one situation at time point t. If Law_j does not refer to the effects that depend on the duration then the proof is the same with that of theorem 4.8. If Law_j refers to the effects that depend on the duration then this mean that there is a fluent f which truth value depend on the duration of an action a. Without loss of generality we assume that the fluent f becomes true if the action a executes for more than b time points. Assumed that the action a executes for more than b time points and the fluent f is false. Because the truth value of fluent f depends on the duration of an action a, the static rule which refers to it has the form

$$
\begin{aligned}
&G_f(...) \supset f(...)\\
&where\\
&G_f(....) \equiv (\bigvee(\bigwedge fi(...))) \vee \ (f_\alpha(a) \geq b)
\end{aligned}
$$

This means that the formula $G_f(t, ..)$ is true at time point t because the second part $((f_\alpha(a) \geq b)$ is true. Thus the rule $G_f(t, t') \supset f(t')$ is evaluated and the fluent f becomes true. A contradiction.

■

Consider the example with the public worker with the new assumption. The dynamic rules are

$$
\begin{aligned}
&occur(misdemeanor(p), t) \supset illegal(p, [t, t+5m]) \quad (1)\\
&occur(take\_pardon(p), t) \supset \neg illegal(p, [t, \infty]) \quad (2)\\
&occur(bad\_grade(p), t) \supset \neg good\_employee(p, [t, \infty]) \quad (3)\\
&occur(good\_grade(p), t) \supset good\_employee(p, [t, \infty]) \quad (4)\,,
\end{aligned}
$$

After the execution of the algorithm which produce the dynamic rules we have

$$
\begin{aligned}
&occur(start(misdemeanor(p)), t) \supset illegal(p, [t, t+5m]) \quad (1a)\\
&occur(end(misdemeanor(p)), t) \supset illegal(p, [t, t+5m]) \quad (1b)\\
&occur(natural(misdemeanor(p)), t) \supset illegal(p, [t, t+5m]) \quad (1c)\\
&occur(start(take\_pardon(p)), t) \supset \neg illegal(p, [t, \infty]) \quad (2)\\
&occur(start(bad\_grade(p)), t) \supset \ \neg good\_employee(p, [t, \infty]) \quad (3)\\
&occur(start(good\_grade(p)), t) \supset \ good\_employee(p, [t, \infty]) \quad (4)\,,
\end{aligned}
$$

The period of the execution of action $natural(misdemeanor(p))$ is 5. The static rules as they are produced by the algorithm in section 4.1.2 are:

$$
\begin{aligned}
&R = \{illegal(p, t_1) \supset suspended(p, t_1),\\
&illegal(p, t_1) \vee \neg good\_employee(p, t_2) \supset \ \neg take\_promotion(p, max(t_1, t_2)),
\end{aligned}
$$

$$\begin{aligned}
&suspended(p) \supset \neg take\_salary(p, t_1),\\
&\neg suspended(p, t_1) \wedge good\_employee(p, t_2) \supset\ take\_bonus(p, min(t_1, t_2)),\\
&\neg good\_employee(p, t_1) \supset \neg take\_bonus(p, t_1),\\
&\neg suspended(p, t_1) \supset take\_salary(p, t_1)\\
&False \supset take\_promotion(p, \infty)\}
\end{aligned}$$

Now we must perform the above algorithm at the set R in order to encapsulate the indirect effects which depend on the duration of some action. The only rule which is effected is the last one and changes from $False \supset take\_promotion$ to $f_\alpha(good\_grade) \geq 2) \supset take\_promotion(p, \infty)$

Assume the following sequence of execution

$$\begin{aligned}
&occur(start(good\_grade(p)), 2)\\
&occur(end(good\_grade(p)), 5)\\
&occur(start(misdemeanor(p)), 6)\\
&occur(end(misdemeanor(p)), 13)
\end{aligned}$$

Assume now that we have a public worker p and the initial situation is

$$\begin{aligned}
S_0 = \{&\neg take\_bonus(p, [0, \infty]),\ take\_salary(p, [0, \infty]),\\
&\neg take\_promotion(p, [0, \infty]),\ \neg suspended(p, [0, \infty]),\\
&\neg good\_employee(p, [0, \infty]),\ \neg illegal(p, [0, \infty])\}\,.
\end{aligned}$$

At time point 2 the execution of the action $good\_grade(p)$ starts and the dynamic rule (4) is evaluated. Now the new situation is

$$\begin{aligned}
S'_1 = \{&\neg take\_bonus(p, [0, \infty]),\ take\_salary(p, [0, \infty]),\\
&\neg take\_promotion(p, [0, \infty]),\ \neg suspended(p, [2, \infty]),\\
&good\_employee(p, [2, \infty]),\ \neg illegal(p, [0, \infty])\}\,.
\end{aligned}$$

After the evaluation of the static rule

$$\neg suspended(p, [2, \infty]) \wedge good\_employee(p, [2, \infty])\ \supset take\_bonus(p, [2, \infty])$$

the new situation is

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

At time point 4 the action *good_grade*(p) executes for more than 2 time points thus the static rule $f_\alpha(good\_grade) \geq 2) \supset take\_promotion(p,\infty)$. is evaluated Now the new situation is

$$\begin{aligned}&S_2 = \{take\_bonus(p,[2,\infty]),\ take\_salary(p,[0,\infty]),\\&take\_promotion(p,[4,\infty]),\ \neg suspended(p,[0,\infty]),\\&good\_employee(p,[2,\infty]),\ \neg illegal(p,[0,\infty])\}\,.\end{aligned}$$

At time point 5 the action *good_grade*(p) ends but the situation does not change. At time point 6 the execution of the action *misdemeanor*(p) starts and the dynamic rule (1a) is evaluates. Now the new situation is

$$\begin{aligned}&S_3' = \{take\_bonus(p,[2,\infty]),\ take\_salary(p,[0,\infty]),\\&take\_promotion(p,[4,\infty]),\ \neg suspended(p,[0,\infty]),\\&good\_employee(p,[2,\infty]),\ illegal(p,[6,11]),\ \neg illegal(p,[11,\infty])\}\,.\end{aligned}$$

After the execution of static rules

$$\begin{aligned}&illegal(p,[6,11]) \supset suspended(p,[6,11])\\&suspended(p,[6,11]) \supset \neg take\_salary(p,[6,11])\\&illegal(p,[6,11]) \vee \neg good\_employee(...) \supset\ \neg take\_promotion(p,[6,11])\end{aligned}$$

the situation at time point 6 is:

$$\begin{aligned}&S_3 = \{take\_bonus(p,[2,\infty]),\ \neg take\_salary(p,[6,11]), take\_salary(p,[11,\infty]),\\&\neg take\_promotion(p,[6,11]),\ take\_promotion(p,[11,\infty]),\ suspended(p,[6,11]),\\&\neg suspended(p,[11,\infty]),\ good\_employee(p,[2,\infty]),\\&illegal(p,[6,11]), \neg illegal(p,[11,\infty])\}\,.\end{aligned}$$

The situation changes at point 11 because the fluent $\neg take\_salary(p,[6,11])$, $\neg take\_promotion(p,[6,11])$, $suspended(p,[6,11])$ and $illegal(p,[6,11])$ ceases to hold. At time point 11 the situation is

$$\begin{aligned}&S_4'' = \{take\_bonus(p,[2,\infty]),\ take\_salary(p,[11,\infty]),\\&take\_promotion(p,[11,\infty]),\ \neg suspended(p,[11,\infty]),\\&good\_employee(p,[2,\infty]),\ \neg illegal(p,[11,\infty])\}\,.\end{aligned}$$

At time point 11 the action $natural(misdemeanor(p))$ executes (because $f_\alpha(misdemeanor(p)) > 0$ holds) and refreshes the effect of action $misdemeanor(p)$. The dynamic rule (1c) evaluates. Now the new situation is

$$\begin{aligned}&S_4' = \{take\_bonus(p,[2,\infty]),\ take\_salary(p,[11,\infty]),\\&take\_promotion(p,[11,\infty]),\ \neg suspended(p,[11,\infty]),\\&good\_employee(p,[2,\infty]),\ illegal(p,[11,16]), \neg illegal(p,[16,\infty])\}\,.\end{aligned}$$

After the execution of the static rules

$$\begin{aligned}&illegal(p,[11,16]) \supset suspended(p,[11,16])\\&suspended(p,[11,16]) \supset \neg take\_salary(p,[11,16])\\&illegal(p,[11,16]) \vee \neg good\_employee(...) \supset\ \neg take\_promotion(p,[11,16])\end{aligned}$$

we have

$$\begin{aligned}&S_4 = \{take\_bonus(p,[2,\infty]),\ \neg take\_salary(p,[11,16]), take\_salary(p,[16,\infty]),\\&\neg take\_promotion(p,[11,16])),\ take\_promotion(p,[16,\infty]),\\&suspended(p,[11,16]),\ \neg suspended(p,[16,\infty]),\ good\_employee(p,[2,\infty]),\\&illegal(p,[11,16]), \neg illegal(p,[16,\infty])\}\,.\end{aligned}$$

If we compare the situations S_3 and S_4 we can clearly observe the rules of refreshing of the effects. At time point 13 the action $misdemeanor(p)$ ends and thus the dynamic rule (1b) will be evaluated. Now the new situation is

$$\begin{aligned}&S_5 = \{take\_bonus(p,[2,\infty]),\ \neg take\_salary(p,[11,16]), take\_salary(p,[16,\infty]),\\&\neg take\_promotion(p,[11,16]),\ take\_promotion(p,[16,\infty]),\\&suspended(p,[11,16]),\ \neg suspended(p,[16,\infty]),\ good\_employee(p,[2,\infty]),\\&illegal(p,[13,18]), \neg illegal(p,[18,\infty])\}\,.\end{aligned}$$

At time point 13 no static rule evaluates because the fluents $\neg take\_salary(p, [11,16])$, $suspended(p,[11,16])$ are true at time point 13. But at time point 16 the situation change because the fluent $suspended(p,[11,16])$, $\neg take\_promotion(p,[11,16])$ and $\neg take\_salary(p,[11,16])$ cease to hold. Thus

$$\begin{aligned}
&S_6' = \{take\_bonus(p,[2,\infty]),\ take\_salary(p,[16,\infty]),\\
&take\_promotion(p,[16,\infty]),\ \neg suspended(p,[16,\infty]),\\
&good\_employee(p,[2,\infty]),\ illegal(p,[13,18]), \neg illegal(p,[18,\infty])\}\,.
\end{aligned}$$

Now the following static rules evaluates

$$\begin{aligned}
&illegal(p,[16,18]) \supset suspended(p,[16,18])\\
&suspended(p,[16,18]) \supset \neg take\_salary(p,[16,18])\\
&illegal(p,[16,18]) \vee \neg good\_employee(...) \supset\ \neg take\_promotion(p,[16,18])
\end{aligned}$$

and the new situation is

$$\begin{aligned}
&S_6 = \{take\_bonus(p,[2,\infty]),\ \neg take\_salary(p,[16,18]),\\
&take\_salary(p,[18,\infty]),\ \neg take\_promotion(p,[16,18]),\\
&take\_promotion(p,[18,\infty]),\ suspended(p,[16,18]),\\
&\neg suspended(p,[18,\infty]),\ good\_employee(p,[2,\infty]),\\
&illegal(p,[13,18]), \neg illegal(p,[18,\infty])\}\,.
\end{aligned}$$

Finally the situation changes again at time point 18 because the fluent $illegal(p,[13,18])$, $suspended(p,[16,18])$, $\neg take\_salary(p,[16,18])$ and $\neg take\_pro\,motion(p,[16,18])$ cease to holds. The new situation is

$$\begin{aligned}
&S_7 = \{take\_bonus(p,[2,\infty]),\ take\_salary(p,[18,\infty]),\\
&\neg take\_promotion(p,[6,\infty]),\ \neg suspended(p,[18,\infty]),\\
&good\_employee(p,[2,\infty]),\ \neg illegal(p,[18,\infty])\}\,.
\end{aligned}$$

4.2 Concurrent Execution

4.2.1 Concurrent execution of Instantaneous Actions

In this section, we examine the case that two or more instantaneous actions execute concurrently. The direct and indirect effects of an action do not start necessarily from the next time moment. This means that two or more actions cannot necessarily be executed concurrently even if their preconditions hold. It must be determined that the direct and indirect effects of these actions are consistent not only in the next time moment but in the future, as well. Also, we must ensure that all direct effects of all actions executing concurrently hold. This means that it is not possible for one action to cancel the effects of other actions which execute concurrently.

In our example, assume that the following actions execute concurrently:

$$\begin{gathered} occur(good\_grade(p), 2), \\ occur(bad\_grade(p), 2)\,. \end{gathered}$$

This means that the following must hold:

$$good\_employee(p, [2, \infty]) \wedge \neg good\_employee(p, [2, \infty])\,.$$

This is inconsistent and the inconsistency arises because of the direct effects. Assume now that, apart from the four actions there is another action *grant_promotion* which has as direct effect to grant a promotion to the worker:

$$occur(grant\_promotion(p), t) \supset take\_promotion(p, [t, \infty])\,.$$

Assume now that the following actions execute concurrently:

$$\begin{gathered} occur(grant\_promotion(p), 2), \\ occur(bad\_grade(p), 2)\,. \end{gathered}$$

This means that the following must hold

$$take\_promotion(p, [2, \infty]) \wedge \neg good\_employee(p, [2, \infty])\,.$$

This is consistent but there is the static rule

$$illegal(p, ..) \vee \neg good\_employee(p, ..) \supset \neg take\_promotion(p, ..)\,.$$

Finally,

$$take\_promotion(p, [2, \infty]) \wedge \neg take\_promotion(p, [2, \infty])$$

must hold. Here there is a contradiction between the direct effect of action *grant_promotion* and the indirect effects of action *bad_grade*. Also, there is the case that there is a contradiction between the indirect effects. The following algorithm addresses the ramification problem.

1. Before the concurrent execution of the actions $a_1, ..., a_n$ check the set $E = \{f_i([t, t']) : \exists a_i \ s.t \ occur(a_i, t) \ \supset \ f_i([t, t'])\}$ is satisfiable [24]. If it is not, reject the concurrent execution.
2. After the execution of concurrent actions, evaluate the dynamic rules which refer to those actions.
3. Execute the algorithm of the evaluation of static rules (see below)
4. If the algorithm of the evaluation of static rules returns inconsistency, then reject the last action, else continue.
5. Until some other action executes, use the situations which have been produced by the algorithm of the evaluation of static rules.

The algorithm for the evaluation of static rules appears below. This algorithm cannot change the direct effects of the actions.

1. At time point t if the static rule $G_f \supset f(L_1)$ evaluated then
   (a) if $\neg f(L_2) \in E$ and $L_1 \cap L_2 \neq \{\}$ then return inconsistency [25].
   (b) else set $L_2 = L_2 \setminus (L_1 \cap L_2)$ and evaluate the rule $K_f \supset \neg f(...)$.
2. Repeat step 1 until L_1 and L_2 do not change or until they take previous values. In the latter case, return inconsistency.

[24]The set E contains all direct effects of the actions that must execute concurrently

[25]There is inconsistency between the direct effects one action and the indirect effects of some other action

3. Repeat the step 1 and 2 for all rules.
4. Repeat the step 1,2 and 3 for all time moments at which there are references.

As we may observe, at the time that some actions execute it is necessary to estimate the indirect effects of these actions not only for the current time moments but for all future moments as well. This is necessary because there is the case that the indirect effects of these action become inconsistent at some time point in the future.

Now we show how the above algorithms work for the examples at the start of the section. In the first example we have concurrent execution of the actions

$$\begin{array}{l} occur(good\_grade(p), 2), \\ occur(bad\_grade(p), 2)\,. \end{array}$$

We know that

$$\begin{array}{l} occur(good\_grade(p), 2) \supset \ good\_employee(p, [2, \infty]) \\ occur(bad\_grade(p), 2) \supset \neg good\_employee(p, [2, \infty])\,. \end{array}$$

At the first step of algorithm we have that

$$E = \{good\_employee(p, [2, \infty]), \neg good\_employee(p, [2, \infty])\}\,.$$

The set E is not satisfiable, thus we reject the execution.

For the second example we have

$$\begin{array}{l} occur(grant\_promotion(p), 2), \\ occur(bad\_grade(p), 2)\,. \end{array}$$

This means that

$$take\_promotion(p, [2, \infty]) \land \ \neg good\_employee(p, [2, \infty])$$

must hold. We know that

$$occur(grant\_promotion(p), 2) \supset \ take\_promotion(p, [2, \infty])$$
$$occur(bad\_grade(p), 2) \supset \neg good\_employee(p, [2, \infty])\,.$$

At the first step of algorithm we have that

$$E = \{take\_promotion(p, [2, \infty]),\ \neg good\_employee(p, [2, \infty])\}\,.$$

The set E is satisfiable thus we continue. After we evaluate the static rule

$$illegal(p, ..) \vee \neg good\_employee(p, [2, \infty]) \supset \ \neg take\_promotion(p, [2, \infty])\,.$$

As we observe

$$\neg\neg take\_promotion(p, [2, \infty]) \equiv \ take\_promotion(p, [2, \infty]) \in E\,.$$

Thus we reject the concurrent execution, because there is inconsistency between the direct effect of action $grant\_promotion(p)$ and the indirect effect of action $bad\_grade(p)$.

Now we present a more complex example. Consider the following sequence of execution:

$$occur(misdemeanor(p), 2),$$
$$occur(take\_pardon(p), 4),$$
$$occur(good\_grade(p), 4),$$
$$occur(grant\_promotion(p), 6)$$
$$occur(bad\_grade(p), 6)\,.$$

Assume that the initial situation is

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

At time point 2 the action $misdemeanor$ executes. The new situation is

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

After the evaluation of the static rules

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

we have

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

At time point 4, the actions *take_pardon*, *good_grade* execute concurrently. We have that

$$E = \{\neg illegal(p,[4,\infty]), good\_employee(p,[0,\infty])\}$$

The set E is satisfiable, thus, the new situation is

$$S_2 = \{\neg take\_bonus(p,[0,\infty]), \neg take\_salary(p,[2,7]), take\_salary(p,[7,\infty]),$$
$$\neg take\_promotion(p,[0,\infty]) suspended(p,[2,7]),$$
$$\neg suspended(p,[7,\infty]), good\_employee(p,[4,\infty]),\ \neg illegal(p,[4,\infty])\}$$

At time point 6 we have the concurrent execution of the actions *grant_promotion*, *bad_grade*. We have that

$$E = \{take\_promotion(p,[6,\infty]), \neg good\_employee(p,[6,\infty])\}$$

The set E is satisfiable thus the new situation is

$$S_3'' = \{\neg take\_bonus(p,[0,\infty]), \neg take\_salary(p,[2,7]),$$
$$take\_salary(p,[7,\infty]), take\_promotion(p,[6,\infty]),$$
$$suspended(p,[2,7]), \neg suspended(p,[7,\infty]),$$
$$\neg good\_employee(p,[6,\infty]), \neg illegal(p,[4,\infty])\}$$

Now the static rule

$$illegal(p,..) \vee \neg good\_employee(p,[6,\infty]) \supset \neg take\_promotion(p,[6,\infty])$$

must be evaluated.

But *take_promotion*$(p,[6,\infty]) \in E$. Thus, we must reject the concurrent execution of these two actions and the resulting situation remains S_2. At time point 7 the situation changes because the fluents *suspended*$(p,[2,7])$, $\neg$*take_salary*$(p,[2,7])$ cease to hold. The new situation is

$$S_3' = \{\neg take\_bonus(p,[0,\infty]), take\_salary(p,[7,\infty]),$$
$$\neg take\_promotion(p,[0,\infty]), \neg suspended(p,[7,\infty]),$$
$$good\_employee(p,[4,\infty]), \neg illegal(p,[4,\infty])\}$$

After the evaluation of the static rule

$$\neg suspended(p,[7,\infty]) \vee good\_employee(p,[4,\infty]) \supset take\_bonus(p,[7,\infty]),$$

we have

$$S_3 = \{take\_bonus(p,[7,\infty]), take\_salary(p,[7,\infty]),$$
$$\neg take\_promotion(p,[0,\infty]), \neg suspended(p,[7,\infty]),$$
$$good\_employee(p,[4,\infty]), \neg illegal(p,[4,\infty])\}$$

As we observe, the employee gets a bonus because the action *bad_grade* has been rejected at time point 6 (it cannot be executed concurrently with the action *grant_promotion*). The execution ends at this point.

The following theorem have been proved

Theorem 4.11 *The above algorithm always returns a consistent situation in the case that some instantaneous action execute concurrently.*

Proof: One situation is consistent when all integrity constraints are satisfied. Also in the case of concurrent execution, all direct and indirect effects of the actions which are execute concurrently must be consistent. In the case that some integrity constraints are violated the proof is similar with the theorem 4.8. Now we must prove that all direct and indirect effects of the actions which are executed concurrently are consistent.

Assume that the actions $a_1, ...a_n$ are executed concurrently and the direct effects of these actions are inconsistent. Then the set E will be satisfied and the execution will be rejected. Assume that there is an inconsistency between the direct effects $f([t, t_1])$ of action a_i and the indirect effects of action a_j[26]. Then there must be a sequence of evaluation of static rules

$$G_{f'}(...) \supset f'(...)$$
$$.......$$
$$G_{\neg f}((...) \supset \neg f([t_2, t_3])$$
$$.......$$
$$G_{f''}(...) \supset f''(...)$$

and also $[t, t_1] \cap [t_2, t_3] \neq \emptyset$. In that case the algorithm rejects the concurrent execution of actions $a_1, ...a_n$ [27].Thus there cannot be inconsistency between the direct effect of action a_i and the indirect effects of action a_j.

In the case that there is inconsistency between the indirect effects of actions a_i and the indirect effects of action a_j, then there must be a sequence of evaluation of static rules

$$G_{f'}(...) \supset f'(...)$$
$$.......$$
$$G_{\neg f} \supset \neg f([t_1, t_2])$$
$$......$$
$$G_f(...) \supset f([t_3, t_4])$$
$$....$$
$$G_{\neg f}(...) \supset \neg f([t_1, t_2])$$
$$......$$
$$G_f(...) \supset f([t_3, t_4])$$

[26]In order for an inconsistency to exist, the action a_j must has as indirect effect the $\neg f([t2, t3])$ and $[t, t_1] \cap [t_2, t_3] \neq \emptyset$. If $[t, t_1] \cap [t_2, t_3] = \emptyset$ then there is no inconsistency because the two effects refer to different time intervals.

[27]The algorithm(step 1.b) discovers that some indirect effects are inconsistent with the direct effects which exist in set E.

As we observe the fluents $\neg f$ and f appera twice during the same evaluation of static rules. Thus the algorithm rejects the execution.

■

4.2.2 Concurrent execution of actions with duration

When actions have duration, it is not enough to describe the effects at the start and at the end of their duration. Assume that the action misdemeanor has duration more than five time points. In that case the public worker is assumed illegal for the time that the action misdemeanor executed and for five time points after the end. This means that it is not enough to evaluate dynamic rules at the start and at the end of the action because there is an interval

$$[occur(start(misdemeanor(p))) + 5, occur(end(misdemeanor(p)))]$$

during which the direct effects do not hold. This happens because the dynamic rule renders the fluent *illegal* true for 5 time moments. In order to address this problem, we define a *natural action* $natural(a)$ for each action a with duration. This natural action is instantaneous and will be executed periodically. The direct effects of natural actions are exactly the same with those of action a. This means that if

$$\begin{array}{l} occur(a, t) \supset \bigwedge f_i([t, t'_i]) \\ then \\ occur(natural(a), t) \supset \bigwedge f_i([t, t'_i]) \end{array}$$

The period is equal to the minimal time that some fluent becomes true (as direct effect of action a) after the execution of this action. For example if

$$occur(a, t) \supset f_1([t, t + 3]) \wedge f_2([t, t + 4]) \wedge f_3([t, t + 7])$$

holds, then the period of execution is 3. The precondition of the action $natural(a)$ is the execution of action a. This means

$$Poss(natural(a)) \equiv f_\alpha(a) > 0$$

Furthermore, the actions which execute concurrently do not necessarily start or end at the same time point. Actions $a_1, ...a_n$ execute concurrently at a time point t_1 if all of them execute at t_1. This means that in the next time point perhaps a subset of the $a_1, ...a_n$ execute concurrently (because the rest of them have terminated their execution) together with another set of actions $a_{n+1}, ...a_k$ which start their execution at time point $t_1 + 1$. For instance, assume the following execution of actions

$$\begin{array}{l} start(misdemeanor(p), 1), \\ start(good\_grade(p), 3), \\ end(good\_grade(p), 5), \\ start(bad\_grade(p), 7), \\ end(bad\_grade(p), 9), \\ end(misdemeanor(p), 10)\,. \end{array}$$

At time points $3, 4, 5$ we have the concurrent execution of the actions $misdemeanor(p)$ and $good\_grade(p)$, while at time point $7, 8, 9$ we have the concurrent execution of the actions $misdemeanor(p)$ (the same action) and $bad\_grade(p)$. This means that we must examine at each time point if the actions which execute at this time point may execute concurrently. Consider the following execution of actions

$$\begin{array}{l} start(take\_pardon(p), 1), \\ start(good\_grade(p), 3), \\ start(bad\_grade(p), 7), \\ end(good\_grade(p), 8) \\ end(bad\_grade(p), 9), \\ end(take\_pardon(p), 10)\,. \end{array}$$

At time points $3, 4, 5, 6$ we have the concurrent execution of the actions $take\_pardon(p)$ and $good\_grade(p)$. There is no inconsistency from this concurrent execution. At time point 7 we have concurrent execution of the actions $take\_pardon(p)$, $good\_grade(p)$ and $bad\_grade(p)$. There is an inconsistency here because $good\_grade(p)$ and $bad\_grade(p)$ cannot execute concurrently. In this case the last action ($bad\_grade(p)$) is rejected.

Another problem that arises in the case of concurrent actions with duration, is that of effects that depend on the duration of the actions. It is possible, for instance, that actions may execute concurrently only if their duration is shorter than a time threshold. For example, assume that in the above example, a public worker may receive a promotion if the action $good\_grade$ executed for an interval longer than two months. Assume now that the actions $misdemeanor$ and $good\_grade$ execute concurrently for an interval longer than two months. Apparently, there is an inconsistency. In order to encapsulate the effects of an action that depends on the duration of the execution, we change each static rule $G_f \supset f$ (which refers to fluent f whose truth value changes according to the duration of the execution) as follows:

$$f_\alpha(a) \vee G_f \supset f \, .$$

Now we have a new static rule

$$f_\alpha(good\_grade) > 2 \supset take\_promotion(p, [now, \infty]) \, .$$

Assume that the initial situation is

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

Assume that the following actions execute concurrently:

$$\begin{aligned}
&start(good\_grade(p), 0),\\
&start(misdemeanor(p), 0),\\
&end(good\_grade(p), 3),\\
&end(misdemeanor(p), 3) \, .
\end{aligned}$$

At time point 0 after the start of execution the situation is

$$\begin{aligned}
&S_1 = \{\neg take\_bonus(p, [0, \infty]), \neg take\_salary(p, [0, 5]),\\
&take\_salary(p, [5, \infty]), \neg take\_promotion(p, [0, \infty])\\
&suspended(p, [0, 5]), \neg suspended(p, [5, \infty]),\\
&illegal(p, [0, 5]), \neg illegal(p, [5, \infty]), \neg good\_employee(p, [0, \infty])\}.
\end{aligned}$$

At time point 2 the situation changes because the last static rule will be evaluated.

Now the set E becomes:

$$E = \{\neg take\_promotion(p, [0, \infty]), ...\} \, .$$

But for the new static rule we have $take\_promotion(p, [2, \infty])$. Thus, there is an inconsistency and the concurrent actions must rejected. But if the concurrent execution was

$$\begin{aligned}&start(good\_grade(p), 0),\\&start(misdemeanor(p), 0),\\&end(good\_grade(p), 1),\\&end(misdemeanor(p), 3)\end{aligned}$$

then the last static rule would not be evaluated and thus the concurrent execution would be acceptable. Notice that in that case there is no inconsistency. The algorithm which we presented in the previous section can be modified in order to address the ramification problem in the case that actions have duration. The only decision that must be made concerns the way that we reject the concurrent actions. In case of inconsistency, we choose to reject the action(s) which started their execution more recently. The new algorithm is:

1. At each time point do: If the execution of action a_i starts, then if actions $a_1, ..a_{i-1}, a_{i+1}, ...a_n$ continues to execute or starts its execution check that the set $E = \{f_i([t, t']) : \exists a_i \ s.t \ occur(a_i, t) \ \supset f_i([t, t'])\}$. is satisfiable [28]. If it is not, reject the action a_i.
2. Evaluate the dynamic rules which refer to the actions which start their execution. For each time point/interval included in a list L (such that $f(L)$ holds) execute the algorithm of the evaluation of static rules.
3. If the algorithm of the evaluation of static rules returns an inconsistency, then reject the last action and repeat the execution for the time point that the last action started its execution, else continue.
4. In each time point at which no action executed check the static rules which depend on the action's duration (i.e., those whose body contains the fluent f_a).
   (a) if no static rule evaluates then go to step 1 and wait for the next time point.
   (b) else go to step 4.
5. If an action a_i terminates its execution, then remove from the set E the direct effects which refer to it.

The algorithm for the evaluation of static rules does not change. Consider the following execution:

[28] The set E contains all direct effects of the actions that execute concurrently at the specific time point

$$
\begin{aligned}
&start(misdemeanor(p), 2),\\
&start(bad\_grade(p), 3),\\
&start(good\_grade(p), 4),\\
&end(bad\_grade(p), 5),\\
&end(good\_grade(p), 6),\\
&start(good\_grade(p), 7),\\
&end(misdemeanor(p), 11),\\
&end(good\_grade(p), 12)\,.
\end{aligned}
$$

Assume that the initial situation is

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

At time point 2 the execution of the action $misdemeanor(p)$ starts. The new situation after the evaluation of dynamic and static rules is

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

At time point 3 the execution of the action $bad\_grade(p)$ starts. The situation does not change. At time point 4 the execution of the action $good\_grade(p)$ starts. As we observe the set E now is

$$
E = \{\neg good\_employee(p, [0, \infty]),\ good\_employee(p, [4, \infty]), ..\}
$$

This set is not satisfiable and thus we must reject the execution of action $good\_grade(p)$. Thus we go on with the execution for time point 4 without the execution of action $good\_grade(p)$.

At time point 7 $natural(misdemeanor(p))$ will be executed and it "refreshes" the effect of action $misdemeanor(p)$. Now the new situation is

$$S_2' = \{\neg take\_bonus(p,[0,\infty]), \neg take\_salary(p,[7,12]),$$
$$take\_salary(p,[12,\infty]), \neg take\_promotion(p,[0,\infty]),$$
$$suspended(p,[7,12]), \neg suspended(p,[12,\infty]),$$
$$illegal(p,[7,12]), \neg illegal(p,[12,\infty]), \neg good\_employee(p,[0,\infty])\}$$

Also at time point 7 the execution of action *good_grade*(p) starts. There is no inconsistency at time point 7, thus we accept this action. The new situation is

$$S_2 = \{\neg take\_bonus(p,[0,\infty]), \neg take\_salary(p,[7,12]),$$
$$take\_salary(p,[12,\infty]), \neg take\_promotion(p,[0,\infty]),$$
$$suspended(p,[7,12]), \neg suspended(p,[12,\infty]),$$
$$illegal(p,[7,12]), \neg illegal(p,[12,\infty]), good\_employee(p,[7,\infty])\}\,.$$

At time point 9 the action *good_grade*(p) it executed for more than 2 time points. This means that the static rule

$$f_\alpha(good\_grade) > 2 \supset take\_promotion(p,[now,\infty])$$

must be evaluated. The new situation now is

$$S_3' = \{\neg take\_bonus(p,[0,\infty]), \neg take\_salary(p,[7,12]),$$
$$take\_salary(p,[12,\infty]), take\_promotion(p,[9,\infty]),$$
$$suspended(p,[7,12]), \neg suspended(p,[12,\infty]),$$
$$illegal(p,[7,12]), \neg illegal(p,[12,\infty]), good\_employee(p,[7,\infty])\}$$

Now the static rule

$$illegal(p,L_1) \vee \neg good\_employee(p,L_2) \supset \ \neg take\_promotion(p,L_1 \cup L_2)$$

must be evaluated. Now the new situation is

$$S_3'' = \{\neg take\_bonus(p,[0,\infty]), \neg take\_salary(p,[7,12]),$$
$$take\_salary(p,[12,\infty]), \neg take\_promotion(p,[9,\infty]),$$
$$suspended(p,[7,12]), \neg suspended(p,[12,\infty]),$$
$$illegal(p,[7,12]), \neg illegal(p,[12,\infty]), good\_employee(p,[7,\infty])\}$$

Now we must evaluate again the static rule

$$f_\alpha(good\_grade) > 2 \supset take\_promotion(p, [now, \infty])\,.$$

As we observe we take again the *take_promotion*$(p, [9, \infty])$. Now the algorithm of the evaluation of static rules will return an inconsistency. Thus, we must reject the execution of action *good_grade* and repeat the execution for the time point 7.

As can be seen in this example, rejection may cause backtracking to a previous time point when the effects of some actions depend on their duration. This happens because some action which executed concurrently with other actions may yield a consistent situation if the duration of some of them is shorter than a time threshold. The following result has been established:

Theorem 4.12 *The above algorithm always returns a consistent situation in the case that actions with duration execute concurrently.*

Proof: In order to be correct the algorithm, it must always return a consistent situation. This means that all integrity constraints are satisfied at this situation. In the case that some integrity constraints which do not depend on the duration of the action are violated the proof is similar with the theorem 4.8. Also we must prove that all direct and indirect effects of the actions which are executed concurrent are consistent. In the case that the effects do not depend on the duration of actions then the proof is similar with that of theorem 4.11. Thus we must prove that:

1. All integrity constraints which depend to the duration of actions are satisfied.
2. All indirect effects (of the actions which executed concurrently) which depend on the duration of the actions are consistent among them, and consistent with other effects of the actions.

Proof 1

Assume that integrity constraint Law_j is not satisfied in one situation at time point t. Then there must be a fluent f which must be true, because some action a executed for time more than b time points, but it is not[29]. For the new algorithm of the production of the static rules, we have that the static rule which refers to the fluent f has the form

$$\begin{aligned}&G_f(t, ..) \supset f(...)\\ &where\\ &G_f(t, ..) \equiv (\bigvee(\bigwedge fi)) \vee \ (f_a(t, 1) \wedge (t \geq b))\end{aligned}$$

[29]Now we examine the case that an integrity constraint is not satisfied if some fluent which its truth values depends from the duration to have wrong truth value.

This mean that the formula $G_f(t,t')$ is true at time point t because the second part $((f_a(t,1) \wedge (t \geq b))$ is true. Thus the rule $G_f(t,t' \supset f(...)$ is evaluated. This means that the fluent f becomes true at time point t. Thus Law_j is satisfied.

Proof 2

Assume now that there is inconsistency between two indirect effects. Then there must be a sequence of evaluation of static rules

$$\begin{array}{l} G_{f_1} \supset f_1 \\ .... \\ G_{\neg f} \supset \neg f([t_1,t_2]) \\ ...... \\ G_f \supset f([t_3,t_4]) \\ .... \\ G_{\neg f} \supset \neg f([t_1,t_2]) \\ ...... \\ G_f \supset f([t_3,t_4]) \end{array}$$

In order for inconsistency to exist it must be case that $[t_1,t_2] \cap [t_3,t_4] \neq \emptyset$[30]. As we can observe the fluents $\neg f$ and f take the same value twice in the same evaluation of static rules. Thus the algorithm rejects the execution of the last action.

$\blacksquare$

4.2.3 Concurrent execution of Instantaneous Actions when the effects refer to the future

When the actions are instantaneous, the backtracking rejection is not necessary. The algorithm which has been proposed in section 4.2.1 can solve the problem in that case. This algorithm finds all the situations in the future (assuming that no other action will be executed). If all these situations are consistent then the execution is accepted, otherwise the algorithm rejects the execution. This permits us to avoid the backtracking rejection and we can use all the future situations which the algorithm has already estimated without needing to execute again the static rules.

Consider the example with the public worker, where the direct effects of the actions start to hold after 2 time points from the execution. Thus the dynamic rules are:

[30]If $[t,t_1] \cap [t_2,t_3] = \emptyset$ then there is no inconsistency because the two effects referred in different time intervals.

$$\begin{aligned}
&occur(misdemeanor(p), t) \supset illegal(p, [t+2, t+7]) \quad (1')\\
&occur(take\_pardon(p), t) \supset \neg illegal(p, [t+2, \infty]) \quad (2')\\
&occur(bad\_grade(p), t) \supset \neg good\_employee(p, [t+2, \infty]) \quad (3')\\
&occur(good\_grade(p), t) \supset good\_employee(p, [t+2, \infty]) \quad (4'),
\end{aligned}$$

In addition to the other constraints, assume that if a public worker is in suspension then he is considered to be a bad employee. The static rules are:

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

Consider the following execution:

$$\begin{aligned}
&occur(good\_grade(p), 2)\\
&occur(misdemeanor(p), 2).
\end{aligned}$$

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

$$\begin{aligned}
&S_0 = \{\neg take\_bonus(p, [[0, \infty]]), take\_salary(p, [[0, \infty]]),\ \neg take\_promotion(p, [[0, \infty]]),\\
&\neg suspended(p, [[0, \infty]]),\ \neg good\_employee(p, [[0, \infty]]), \neg illegal(p, [[0, \infty]])\}.
\end{aligned}$$

At time point 2 the actions *good_grade* and *misdemeanor* execute. Now the new situation is

$$\begin{aligned}
&S_1 = \{\neg take\_bonus(p, [[0, \infty]]), take\_salary(p, [[0, \infty]]),\ \neg take\_promotion(p, [[0, \infty]]),\\
&\neg suspended(p, [[0, \infty]]),\ \neg good\_employee(p, [[2, 4]])\ good\_employee(p, [[4, \infty]]),\\
&illegal(p, [[4, 9]]), \neg illegal(p, [[9, \infty]])\}.
\end{aligned}$$

The set E of the direct effects is

$$E = \{illegal(p, [[4, 9]]), good\_employee(p, [[4, \infty]])\}$$

This set is satisfied in the current situation. Thus we continue the execution. Now the algorithm estimates all the indirect effects in the future (not only in the current time moment). This means that the static rules will be executed for all time intervals. The following static rules are evaluated

$$\begin{aligned}
&illegal(p, [4, 9]) \supset suspended(p, [4, 9]) \quad (a)\\
&suspended(p, [4, 9]) \supset \neg good\_employee(p, [4, 9]) \quad (b)\\
&suspended(p, [4, 9]) \supset \neg take\_salary(p, [4, 9]) \quad (c)\,.
\end{aligned}$$

As we observe there is an inconsistency between the indirect effect of the static rule (b) and the effects which are in the set E, because for the time interval $[4, 9]$, $\neg good\_employee \wedge good\_employee$[31] must hold. Thus we reject the execution.

4.2.4 Concurrent execution of Actions with duration when the effects refer to the future

When the actions have duration and their effects refer to the future then the backtracking rejection is necessary because we cannot "predict" the future situations.

Consider the example with the public worker and assume that the direct effect of the actions *good_grade* and *bad_grade* start to hold 2 time points after the time point at which the actions have completed 4 time points of execution. This means that direct effects start to hold 6 time points after the start of execution of the actions if and only if the actions have been executed for more than 4 time points. The direct effects of other actions start to hold after 2 time points after the start of their execution. Thus, the dynamic rules are

$$\begin{aligned}
&occur(start(misdemeanor(p)), t) \supset illegal(p, [t + 2, t + 7]) \quad (1a)\\
&occur(end(misdemeanor(p)), t) \supset illegal(p, [t + 2, t + 7]) \quad (1b)\\
&occur(start(take\_pardon(p)), t) \supset \neg illegal(p, [t + 2, \infty]) \quad (2')\,.
\end{aligned}$$

Assume that if a public worker is in suspension then he is considered a bad employee. The static rules are

[31]Notice that the time interval $[4, 9]$ refer to the future(the execution of the algorithm takes place at time point 2).

$$\begin{aligned}
&R = \{illegal(p, L) \supset suspended(p, L),\\
&illegal(p, L_1) \vee \neg good\_employee(p, L_2) \supset \ \neg take\_promotion(p, L_1 \cup L_2),\\
&suspended(p, L) \supset \neg take\_salary(p, L),\\
&\neg suspended(p, L_1) \wedge good\_employee(p, L_2) \supset take\_bonus(p, L_1 \cap L_2),\\
&\neg good\_employee(p, L) \supset \neg take\_bonus(p, L),\\
&\neg suspended(p, L) \supset take\_salary(p, L),\\
&(f_a(good\_grade(p)) \geq 4) \supset \ good\_employee(p, [now + 2, \infty]),\\
&suspended(p, L) \vee (f_a(bad\_grade(p)) \geq 4) \ \supset \neg good\_employee(p, L)\}
\end{aligned}$$

Consider the following execution

$$\begin{aligned}
&occur(start(misdemeanor(p)), 0)\\
&occur(start(good\_grade(p)), 2)\\
&occur(end(good\_grade(p)), 8)\\
&occur(end(misdemeanor(p)), 8)\,.
\end{aligned}$$

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

$$\begin{aligned}
&S_0 = \{\neg take\_bonus(p, [[0, \infty]]), take\_salary(p, [[0, \infty]]), \ \neg take\_promotion(p, [[0, \infty]]),\\
&\neg suspended(p, [[0, \infty]]), \ \neg good\_employee(p, [[0, \infty]]), \neg illegal(p, [[0, \infty]])\}\,.
\end{aligned}$$

At time point 0 we evaluate the dynamic rule (1a). Afterwards we evaluate the static rules. The new situation is

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

At time point 5 the natural action for the action *misdemeanor* executes and "refreshes" the effects of action *misdemeanor*. The following rule

$$occur(natural(misdemeanor), 5) \supset illegal(p, [7, 12])$$

will be evaluated. After the evaluation of the static rules, the new situation is

$$S_2 = \{\neg take\_bonus(p, [[0, \infty]]),\ \neg take\_salary(p, [[2, 12]]), take\_salary(p, [[12, \infty]]),$$
$$\neg take\_promotion(p, [[0, \infty]]),\ suspended(p, [[2, 12]]), \neg suspended(p, [[12, \infty]]),$$
$$\neg good\_employee(p, [[0, \infty]]),\ illegal(p, [[2, 12]]), \neg illegal(p, [[12, \infty]])\}\,.$$

At time point 6 the action *good_grade* executes for 4 time points, thus the static rule

$$(f_a(good\_grade(p)) \geq 4) \supset good\_employee(p, [8, \infty])$$

must be evaluated. Now the new situation is

$$S_3 = \{\neg take\_bonus(p, [[0, \infty]]),\ \neg take\_salary(p, [[2, 12]]), take\_salary(p, [[12, \infty]]),$$
$$\neg take\_promotion(p, [[0, \infty]]),\ suspended(p, [[2, 12]]), \neg suspended(p, [[12, \infty]]),$$
$$good\_employee(p, [[8, \infty]]),\ illegal(p, [[2, 12]]), \neg illegal(p, [[12, \infty]])\}\,.$$

But at this situation the following static rule

$$suspended(p, [2, 12]) \vee (duration(bad\_grade(p)) \geq 4)\ \supset \neg good\_employee(p, [2, 12])$$

must be evaluated. Now the new situation is

$$S_4 = \{\neg take\_bonus(p, [[0, \infty]]),\ \neg take\_salary(p, [[2, 12]]), take\_salary(p, [[12, \infty]]),$$
$$\neg take\_promotion(p, [[0, \infty]]),\ suspended(p, [[2, 12]]), \neg suspended(p, [[12, \infty]]),$$
$$\neg good\_employee(p, [[2, 12]]), good\_employee(p, [[12, \infty]]),\ illegal(p, [[2, 12]]), \neg illegal(p, [[12, \infty]])\}\,.$$

But at this situation the following static rule

$$(f_a(good\_grade(p)) \geq 4) \supset good\_employee(p, [8, \infty])$$

must be evaluated again. Two rules evaluate one after the other for times. Thus, the algorithm rejects the concurrent execution.

4.3 Extension of the solution for the Sequential Execution

As we have mentioned in section 4.1.5 , in order for the algorithm to always terminate the precondition $G_f \wedge K_f \equiv False$, when $G_f \supset f$ and $K_f \supset \neg f$, must hold for every fluent f. As we can observe, the last algorithm can solve the ramification problem without these preconditions holding. This happens because the algorithm can reject the execution of some action if it has an inconsistent situation as result.

Consider the example with the public worker and assume that there exists an extra integrity constraint: when a public worker is a good employee then s/he takes a promotion. Now the set of static rules is:

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

As we observe, for the pair $(take\_promotion(p), \neg take\_promotion(p)$, the above assumption does not hold because $good\_employee(p, L) \wedge (illegal(p, L_1) \vee \neg good\_employee(p, L_2))$ can be true when $good\_employee(p) \wedge illegal(p)$ holds.

Assume the following execution

$$\begin{aligned}
&occur(misdemeanor(p), 4)\\
&occur(good\_grade(p), 6)\,.
\end{aligned}$$

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

$$\begin{aligned}
&S_0 = \{\neg take\_bonus(p, [0, \infty]),\ take\_salary(p, [0, \infty]), \neg take\_promotion(p, [0, \infty]),\\
&\neg suspend(p, [0, \infty]), \neg good\_employee(p, [0, \infty]),\ \neg illegal(p, [0, \infty])\}\,.
\end{aligned}$$

At time point 4 the action $misdemeanor(p)$ executes. The new situation is

$$\begin{aligned}
&S_1' = \{\neg take\_bonus(p, [0, \infty]),\ take\_salary(p, [0, \infty]), \neg take\_promotion(p, [0, \infty]),\\
&\neg suspended(p, [0, \infty]), \neg good\_employee(p, [0, \infty]),\ illegal(p, [4, 9]), \neg illegal(p, [9, \infty])\}
\end{aligned}$$

After the evaluation of static rules, the situation is

$$S_1 = \{\neg take\_bonus(p,[0,\infty]),\ \neg take\_salary(p,[4,9]), take\_salary(p,[9,\infty]),$$
$$\neg take\_promotion(p,\infty),\ suspended(p,[4,9]), \neg suspended(p,[9,\infty]),$$
$$\neg good\_employee(p,[0,\infty]),\ illegal(p,[4,9]), \neg illegal(p,[9,\infty])\}$$

At time point 6 action $good\_grade(p)$ executes. The new situation is

$$S_2' = \{\neg take\_bonus(p,[0,\infty]),\ \neg take\_salary(p,[4,9]), take\_salary(p,[9,\infty]),$$
$$\neg take\_promotion(p,\infty),\ suspended(p,[4,9]), \neg suspended(p,[9,\infty]),$$
$$good\_employee(p,[6,\infty]),\ illegal(p,[4,9]), \neg illegal(p,[9,\infty])\}$$

Now the static rule

$$good\_employee(p,[6,\infty]) \supset take\_promotion(p,[6,\infty])$$

must be evaluated. Afterwards we must examine if the static rule

$$illegal(p,[4,9]) \vee \neg good\_employee(p,L) \supset\ \neg take\_promotion(p,..)$$

must be evaluated. We observe that we must evaluate this static rule. After that the static rule $good\_employee(p,[6,\infty]) \supset take\_promotion(p,[6,\infty])$ must be evaluated again. Now we produce the same result again $take\_promotion(p,[6,\infty])$ and thus the algorithm returns an inconsistency. Thus it rejects the execution of the action $good\_grade(p)$ and repeats the execution for the time point 6.

4.4 Symmary

The solutions which have been proposed for the ramification problem in conventional databases [21, 34, **?**, 62, 100, 124, 125, 131] 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 persit, i.e., 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 possibly for many future situations as well. In the public worker example, the action *misdemeanor*(p) has the indirect effect that the public worker is in suspension for the next five months. In these five months, the action *good_grade*, could occur, but even if this happens, the employee cannot take 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. Futhermore, five months after the execution of the action *misdemeanor* the situation changes without an action take 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 persit until an action changes their truth value.

We propose a solution for addressed the ramification problem in temporal database when change refer to the future. We extended a previous proposal by McChain and Turner [82]. 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 each fluent f there are two static rules one for its 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 the 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

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

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

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 sybolism 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 evaluated at every time point at which the correspondence fluent is false.

We propose an extension of the above solution for the case of conccurently execution more than one actions.

In the following chapter we study the ramification problem in the case that the effects of an action can change the belief about the past. In this case the problem is very complex with many diferrent aspects. Notice that if an ction change the past then it could cancel the execution some other action which has been executed before it.

5 Changing the belief about the past

5.1 Motivation

Recall the example from the section 4.1. Assume that if a public employee commits a misdemeanor, then for the next five months s/he is considered illegal, except if s/he receives a pardon. When a public employee is illegal, then s/he must be suspended and cannot take promotion for the entire time interval over which s/he is considered illegal. Also when a public employee is suspended cannot take his/her salary until s/he stops being suspended. Each public employee is graded for his/her work which do. If s/he receives a bad grade, then s/he is assumed bad employee and s/he cannot take promotion or increase until s/he receives a good grade. If s/he takes a good grade, then s/he is assumed a good employee and s/he takes bonus if s/he not suspended.

Now we extend the example in order to present the problem in the case that an action could change the belief about the past. Each public employee receives increase and promotion every two and five years, respectively, if s/he is not illegal. If s/he does a misdemeanor, then s/he is assumed illegal and s/he cannot take promotion while being illegal. We have two new function fluents $position(p, l, t, t_1)$, $salary(p, l, t, t_1)$ where the first one means that the public worker p at time point t is in position l for t_1 time points, while the latter one means that the public worker p at time point t takes salary l for t_1 time points. Also we have two new actions *grant_increase* and *grant_promotion* which are being executed every two and five years, respectively. The preconditions of these actions are

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

We extended the predicate *occur* in order to encapsulate the case that an action change the belief about the past.

$$occur(a, t, t_1)$$

means that the action a occurs at time point t and its effects start to hold at the time point t_1 (perhaps $t_1 < t$).

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 the 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

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

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

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

At this time point the public worker takes pardon which means that for the time 21 the public worker ceases to be assumed illegal. This has as 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 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 an increase at time point 22. Thus the next increase in salary must happen 24 time points after the 22 (resp. 60 months for promotion).

Example 2

Execution of an action may have as indirect effects to disqualify [32] an action which has already 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 natural 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 effect the public worker be illegal at time point 20. This mean that the situation must change at time point 20 in the following:

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

[32] 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 60 and 24 time points after the 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 natural 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 the time point 25.

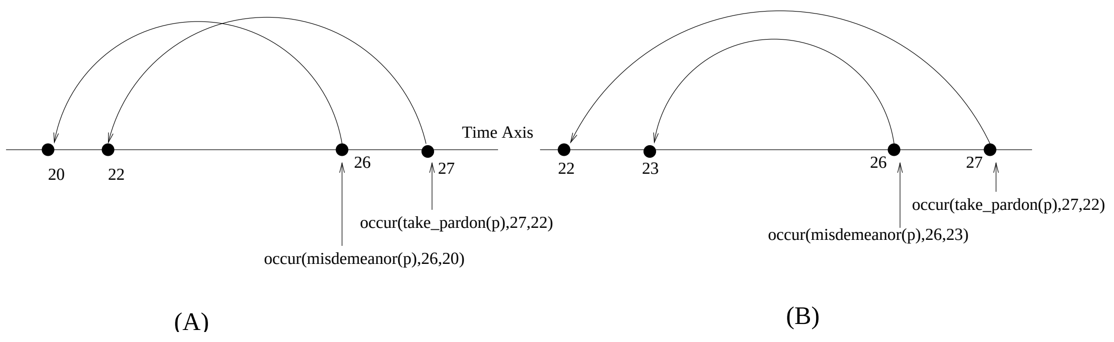
Time Axis
20
22
26
27
occur(take_pardon(p),27,22)
occur(misdemeanor(p),26,20)
(A)
22
23
26
27
occur(take_pardon(p),27,22)
occur(misdemeanor(p),26,23)
(B)

Figure 9: The scenarios of execution

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

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

Thus at time point 24 we must execute the natural actions *grant_promotion* and *take_salary*. This means that we must repeat the execution for the time point 22. But at time 26 we will 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''. The 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 natural 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 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 figure 9. The solution to this problem is to reject the former action whose execution 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

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

Now we have that the natural 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 the time point 23. Notice that at time point 23 the situation S_0 holds(not the S_1 which starts to hold from 24), thus the effects of the *misdemeanor* change the situation S_0. The new situation at time point 23 is

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

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 the time point 22(notice that the effect of the action *take_pardon* change the situation S_0 because the situation S_2'' starts to hold from the time point 23). Now the situation at time point 22 is

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

and we execute the two natural actions at time point 24. At time 26 the action *misdemeanor* is executed again and we have again the situation S_2'' at time 23. Now the two natural 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 effects 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 figure 9.B. Notice that again we break the infinite loop by preferring the action which executes more recently.

In this thesis we propose an algorithm for avoiding the infinite loop by the rejecting the execution of some actions. As we observed from the above examples the correspondences of figure 6 cannot represent the correspondence between situations, actions and time in the case that the effects of the actions refer to the past. This happen because after the execution of an action which changes the past (at time t) we repeat the execution from the 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*:

$$\begin{array}{ll} start(S_0) = 0 & end(S_0) = 24 \\ start(S'_1) = 24 & end(S'_1) = 26 \end{array}$$

After the execution of the action *misdemeanor*

$$\begin{array}{ll} start(S_0) = 0 & end(S_0) = 20 \\ start(S''_1) = 20 & end(S''_1) = 25 \\ start(S''_1) = 25 & end(S_2) = 27 \end{array}$$

At time point 21 we have two different situations S_0 and S''_1. This also happens at time points 22,23,24,25,26. The correspondences of figure 6 ensuring 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 take place. After the execution of the action *misdemeanor* the action *grant_increase* and *grant_promotion* executed at time point 25, while at time 24 no action takes place. The correspondence of figure 6 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.

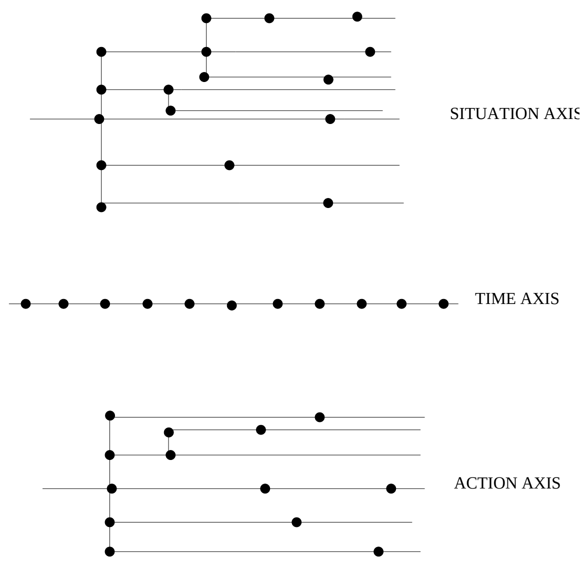

Figure 10: The correspondence between Time-Actions-Situations

We propose to use branching axes for situations and actions, while the time axis remains linear (see figure 10). 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 situations axis is the real evolution of the world. This linear line we call *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.

In the rest of the thesis 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 as actual line, a sequence of situations which is believed to be the evolution of the world up to the current time point, starting at the initial situation.

- We define the fluent $actual(S, t)$ which shows that the 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 [33] 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. [34]

- 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 the situation S in the 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 $T = FluentHold(S, t) \setminus FluentHold(S', t)$ and $T \cap F_P = \emptyset$ [35].

Now we explain informal the predicates $ACCEPTANCE$ and $actual$. Consider figure 11 and assume that the actual line is the top line of the situation axis. Assume that the current

[33] because some fluent ceases to hold
[34] With the execution of the algorithm which we propose in the following sections
[35] this means that there is no change in the fluents of set F_P

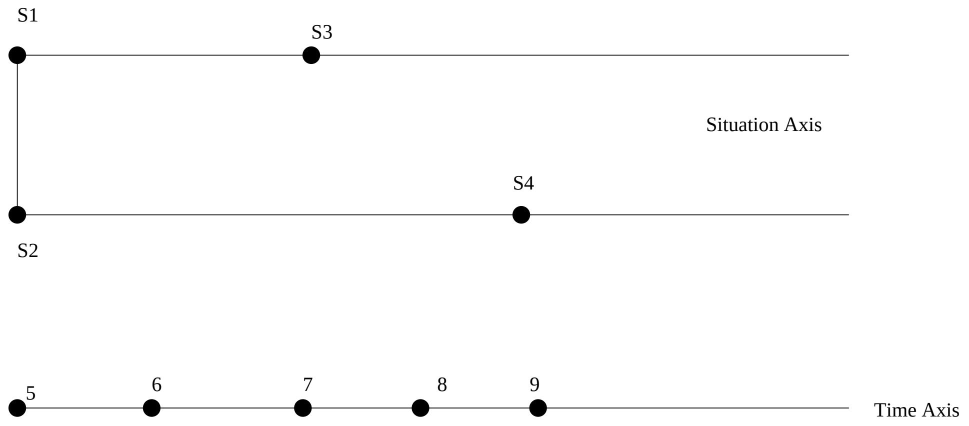

Figure 11: The effects of the past

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 the action a_1 to be acceptable the fluents in F_P must no change their truth values. This mean that at time points 5 and 6 the situation S_2 (in the new actual line) and the 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 the situation S_2 (in the new actual line) and the 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 the situation S_4 (in the new actual line) and the situation S_3 (in the previous actual line) must contain the 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 hold

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

Consider figure 12. Assume that the current time point is 10 and the actual line is the top line (contain the situations S_0, S_{00}, S_1, S_3. As we observe the following hold

$$\begin{array}{ll} start(S_0) = 0 & end(S_0) = 4 \\ start(S_{00}) = 4 & end(S_{00}) = 6 \\ start(S_1) = 6 & end(S_3) = 7 \\ start(S_3) = 7 & end(S_3) = 10 \end{array}$$

Assume the execution $occur(a_1, 10, 5)$. As we observe the action a_1 change the belief about the past (**at time point 5**). Thus we must construct the new actual line. The constructing of the new actual line conclude there step (as seen in the formal definition).

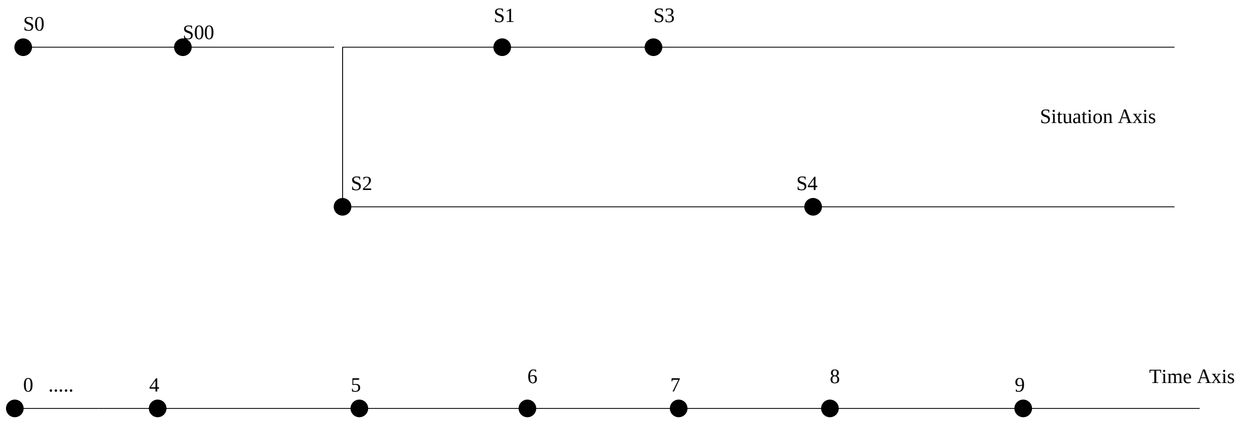

Figure 12: The branching axes

step 1

For the situation S_0 the following holds:

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

This means that the situation S_0 is in the new actual line [36].

step 2

For the situation S_{00} the following holds:

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

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

step 3

For the situations S_1 and S_3 the following hold:

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

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

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

5.3 Fluent Dependencies

This section describe an algorithm that discovers dependencies between fluents in the case that an action change the belief about the past There is necessarily to be distinction between the future and the past. This happened because some fluent cannot change truth value in the past. More specifically a fluent can affect another fluent in the future but not in the past. For changes the future the algorithm is same as presented in the section 4.1.1.

In the past only the fluent which belong in the set F_S can change their truth value. In order to achieve that we modify 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 the 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 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 produced from the set of integrity constraints and from the binary relation I, so the set of static rules is not the same in the two above cases. Thus we construct two binary relation 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 $t \geq now$ 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 which refer to the past and some to the future. Such a combination poses difficult problems which will be addressed in future work.

5.5 Case 1: Change in the past may affect all the fluent

In this case

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

where F is the set of fluent. Because all fluents may change its true value in the past the predicate *Acceptance* is always true [37]. 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 section 4. We propose the following algorithm which returns a consistent situation at each time point(the ramification problem).

Algorithm 1 for constructs a consistent situation

1. At each time point at which some action which changes the truth value of some fluents at the 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 referred.
2. Repeat the execution for the smaller time point t' above. Every time an action is executed, add it to the 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 the set E contains the situations which are produced as effects of the change of past. This helps us to understand when there is infinite loop, which happens when we product the same situation at the same time point. In that case three are two same pair (S, t) in the set E. The set E_1 contains the action which take place after the execution of action which change the past. Thus when there is infinite loop we must reject one action which is in the set E_1. This is achieved by the following algorithm.

The algorithm for the rejecting actions.

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

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 executed more recently in the current time moment.

Example 1 (continued)

We have the execution

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

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

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

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

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

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

$$S'_1 = \{\neg take\_bonus(p, [0, \infty]), \neg take\_salary(p, [20, 25]), take\_salary(p, [[0, 20], [25, \infty]]),$$
$$suspended(p, [20, 25]), \neg suspended(p, [[0, 20], [25, \infty]]),$$
$$\neg good\_employee(p, [0, \infty]),\ illegal(p, [20, 25]), \neg illegal(p, [[0, 20], [25, \infty]]),$$
$$position(p, 1, 24, 62), salary(p, 3, 24, 26)\}\,.$$

Suppose that at time point 24 the public worker takes pardon which means that for the time 21 the public worker stops to be assumed illegal($occur(take\_pardon(p), 24, 21)$). Now we repeat the execution for the 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 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 do not repeat the execution from the time point 21 but it go on at time point 25.

Example 3 (continued)

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

The time start at 0 and the 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 occur the action *misdemeanor* which tell us that the public worker done something illegal at time point 20. This means that the situation must change at time point 20 in the situation S_1''. Now we add (S_1'',20) in to the set E and the action *misdemeanor* in to the set E_1.

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

But at time point 27 executed the action *take_pardon* which tell us that the public worker stop to be illegal at time point 22. Now the new situation is the 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 add the pair (S_2,22) in to the set E and the action *take_pardon* in to the set E_1.

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

We repeat the execution from the time point 22 and at time point 24 must execute the actions *grant_promotion* and *take_salary*. But at time 26 we must execute again the action *misdemeanor*. We known that $occur(misdemeanor(p), 26, 20)) \supset illegal(p, [20, 25])$. But at time point 20 the 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 mean that the action *misdemeanor* does not change the belief about the past thus the algorithm does not evaluate the above dynamic rule.

Example 4 (continued)

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

Consider the same initial situation as the previous example.

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

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

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

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

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

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

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

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

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

Now we add the (*take_pardon*,22) into E and the *take_pardon* into E_1.

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

We repeat the execution from the time point 22 and at time point 24 must execute the actions *grant_promotion* and *take_salary*. But at time 26 we execute again the action

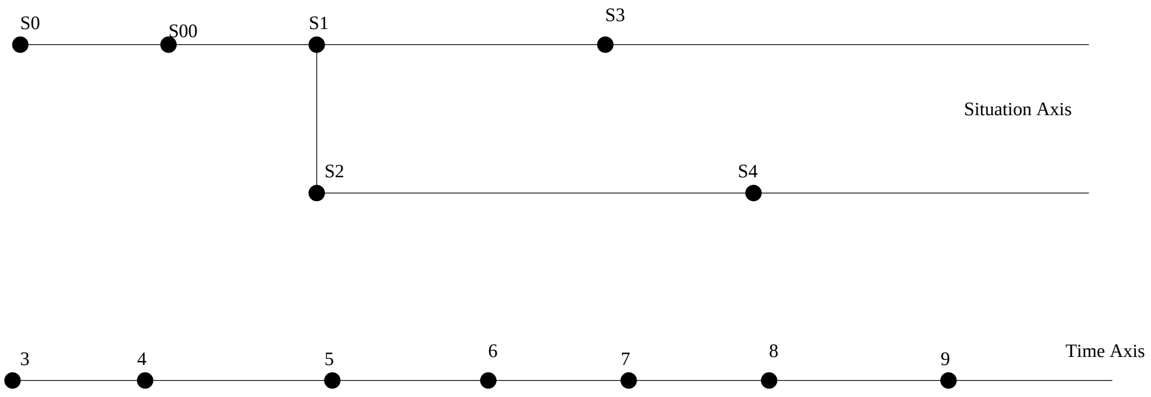

Figure 13: The effects in the past

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

$$occur(misdemeanor(p), 26, 23)) \supset illegal(p, [23, 28])$$
$$occur(take\_pardon(p), 27, 22) \supset \neg illegal(p, [22, \infty])$$

As we observe the $22 < 23 < 26 < 27$. This mean 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.

Now we present two formal results for the above algorithm.

Theorem 5.1 *The above algorithm always returns a consistent situation in the case that some instantaneous 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 the time point t_1 and changes the belief about the past in the 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 the action a will be executed at time point t and its effects start to hold from this time point. [38] Now it is suffices to prove that all situations from the time point t until the current point are consistent.

[38]This means that the predicate $occur(a, t_1, t)$ is "equivalent" to the predicate $occur(a, t, t)$. For example consider in the figure 13 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 the situations S_0, S_{00} until the time point 5. This means that in the time before 5 the old and new actual line is 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 the situation S_{00}, because the actual line will be the same.

As we observe the algorithm at each step (steps 1,2,3) evaluates the static rules until no change occur. 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' ($t < t' \leq t_1$) the algorithm returns a situation S. Assume that integrity constraint Law_j is not satisfied in situation S, and let its CNF be $C_1 \wedge .... \wedge C_n$. Then one of the $C_1, ..., C_n$ is false. Assume that $C_i = f_1 \vee .... \vee f_m$ is false. Then all fluents $f_j, j = 1, ..m$ are false. Assume that f_k and f_p are two of these for which $(f_k, f_p) \in I$ (by the 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: $G_{f_p}(t', L) = G'(..) \vee (\neg \bigwedge f_j(t_j, j = 1, ..m, j \neq p)$. If all fluents $f_j, j = 1, ..m$ are false then $(\neg \bigwedge f_j, j = 1, ..m, j \neq p)$ 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.

■

The proof is similarly with case of sequential execution of action when the action could change only the current and future situation 4.1, if we do the assumption that an action a take place in the time point t_1 and change the belief about the past in the time point $t < t_1$ is equivalent with an action that executed at point t and its effects start to hold for that time point. This assumption does change the effects of the actions. But the algorithm use the knowledge that the action change the belief about the past in order to accept or reject it. The above theorem shown that if the algorithm terminates then return a consistent situation. Now we must prove that the algorithm avoid 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.

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 $S_t^0, S_t^1, ...S_t^k, ...$ [39]

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

In this proof the term "situation" means the truth value of the fluents. Thus the transition from one situation to the next happen only when a fluent changes its truth value. [40]. Notice that because a static rule is evaluated only when the corresponding fluent is false, it is not possible that a static rule $G(t,t') \supset f(t')$ evaluated when the fluent f is true in the 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 loss of generality we assume $S_t^l = S_t^k$, $l < k$.

Thus in the sequence $S_t^l, ..., S_t^k$ 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.

$$
\begin{array}{l}
G(t,t') \supset f'(L') \\
B(t,t'') \supset \neg f'(L'') \\
L' \cap L'' \neq \emptyset
\end{array}
$$

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 the theorem 4.5 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 $(S_t^0, t), (S_t^1, t^1)......(S_t^k, t^k).....$ The 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 change to the new situation S', step 2.a add the pair (S', t') to E. Also, when an action take place, 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 $t^0 < t^1, ...t^n < t_1$. 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

[40]This mean that each $S_t^i = \mathit{FluentHold}(S_m, t)$, for a temporal situation S_m

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 exists twice in the 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

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

$$\begin{aligned} F_P &= \{position, take\_bonus\} \\ F_S &= \{illegal, suspended, good\_employee, \\ &\quad take\_salary, salary\} \end{aligned}$$

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

$$\begin{aligned} &I_{Future} = \{(illegal, suspended),\ (suspended, \neg take\_salary), \\ &(\neg suspended, \neg take\_bonus),\ (good\_employee, \neg take\_bonus), \\ &(\neg good\_employee, take\_salary),\ (\neg suspended, take\_salary)\} \end{aligned}$$

The second referred in the past

$$\begin{aligned} &I_{Past} = \{(illegal, suspended),\ (suspended, \neg take\_salary), \\ &(\neg good\_employee, take\_salary),\ (\neg suspended, take\_salary)\} \end{aligned}$$

Now the algorithm of product static rules return two sets, one for the future effects and one for the past effects [41].

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

[41]The algorithm of producing of static rules is the algorithm which we present in the section 4.1.1 This algorithm depends on the set I. Given different inputs (e.g I_{Past}, I_{Future}), it will be return different sets of static rules.

and

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

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

always hold because all fluents can change their truth value in the future thus all fluents "participate" in the producing of the 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 of the set I_{Past} and R_{past}. Thus for each pair $(f, f') \in I_{Past}$ we have that $(f, f') \in I_{Future}$ while for each pair $(f_1, f_2) \in I_{Future}$ such that f_1 or f_2 cannot change its truth value to the past we have that $(f_1, f_2) \notin I_{Past}$. Steps 3 and 4 of the algorithm for producing the static rules the number of static rules depend from 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 algorithm which 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 belong to the 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 as direct effect to change a fluent which belong in the set F_P then reject the action
      - else add the action to the 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.
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 refer to this action, execute the static rules with belong the set R_{future} until no change occurs.
3. At each time point $t \geq now$ 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)

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

The time start at 0 and the time has the granularity of months. Assume 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 natural actions *grant_promotion* and *grant_increase* executes. Now the new situation is

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

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

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

Now the actions *grant_promotion* and *grant_increase* cannot be executed at time point 24. They will be executed at time 25 which referred in the past. But if the action *grant_promotion* executed at time point 25 despite at 24 the fluent $position \in F_P$ change 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 the situation is the S_1''. Before the execution of action *misdemeanor* at time 24 the situation is S_1' and $S_1'' \backslash S_1' = \{position\} \subseteq F_P$. 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)

Consider now the following execution

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

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

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

At time point 21 the static rules which correspondence 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 execute. At time point 26 the action *take_pardon* execute and change 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 fluent *position* which cannot change 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

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

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 the theorem 4.4 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 in order to find these inconsistent situations. The following algorithm discover the 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 in the set $R_{Future} \setminus R_{Past}$.
   (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 discovery 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 $Law_i = C_1 \wedge ... \wedge C_n$. Then at least one of the $C_1, ..., C_n$ must be false. Assume that C_j is false and $C_j = f_1 \vee ... \vee f_m$. Thus f_l is false for each $l = 1, ..., m$. By the theorems 4.1 and 4.2 we have that there is 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 the steps 3 and 4 of the algorithm of production of static rules we have that: $G_{f_p}(t,..) = G' \vee (\neg \bigwedge f_j(t_j, j = 1,..m, j \neq p)$ and $G_{f_p}(t,..) \supset f_p \in R_{Past}$. If all fluents $f_j, j = 1, ..m$ are false then $(\neg \bigwedge f_j, j = 1, ..m, j \neq p)$ 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 the algorithm 2 is not inconsistent. A contradiction.

Second case is $f_p \in F_p$. Then the rule $(G_{f_p}(t) \supset f_p) \notin R_{Past}$. Then by the steps 3 and 4 of the algorithm of production of static rules we have that: $G_{f_p}(t) = G' \vee (\neg \bigwedge f_j(t_j, j = 1, ..m, j \neq p)$. If all fluents $f_j, j = 1, ..m$ are false then $(\neg \bigwedge f_j, j = 1, ..m, j \neq p)$ is true. Thus $G_{f_p}(t, ..)$ is true (at time point t). So that the rule is executable in the situation S but the algorithm 2 did not evaluate it is not belong to the set R_{Past}. This rule belongs to R_{Future}. Thus $(G_{f_p}(t, ..) \supset f_p(t, ..)) \in (R_{Future} \setminus R_{Past})$. The algorithm discovery the above static rule is executable in S and returns inconsistency. A contradiction [42].

■

Now we must prove that the algorithm which 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 discover all the inconsistent situations. The algorithm in the previous section before return a situation call the algorithm for discovery the inconsistent situations. This mean that cannot return a inconsistent situation. If a situation is inconsistent then it call the algorithm of rejection actions.

■

Theorem 5.5 *The algorithms 2 always returns a consistent situation*

Proof: The above algorithm discover all the inconsistent situations which the algorithm 2 returns. The algorithm 2 before accept a situation call the consistency checking algorithm which discovery all inconsistent situations. This mean that cannot accept an inconsistent situation.

■

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

The proof is similarly with the theorem 5.2.

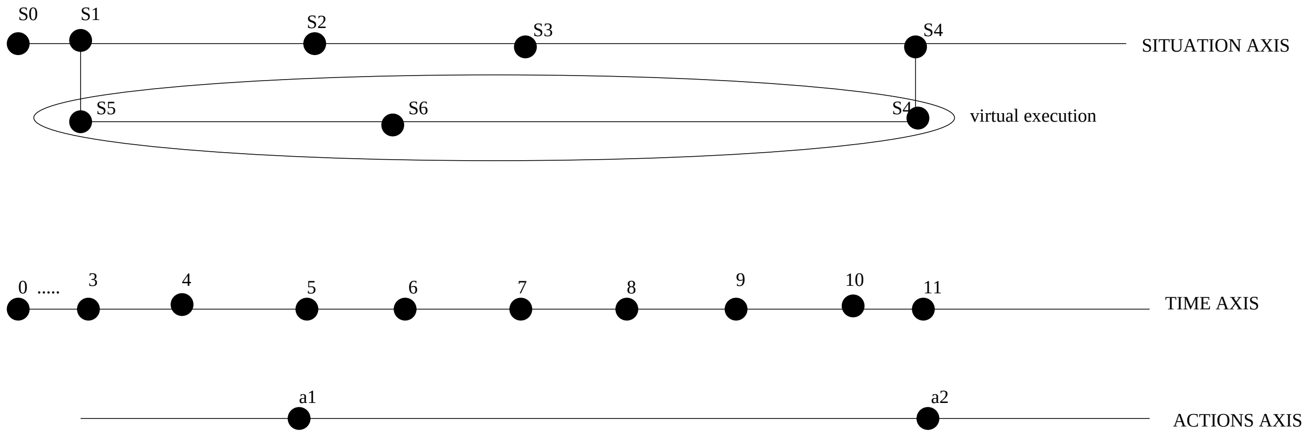

Figure 14: The scenarios of execution

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 the figure 14. Assume that the top line is the evolution of the world and at the time point 11 the action a_2 takes place changes the past at the time 3. The evolution of the world before the execution of the action a_2 is

$$\begin{array}{ll} start(S_0) = 0 & end(S_0) = 3 \\ start(S_1) = 3 & end(S_1) = 5 \\ start(S_2) = 5 & end(S_2) = 7 \\ start(S_3) = 7 & \end{array}$$

Now we evaluate the dynamic rule which corresponds to the action a_2 (at the 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 exists. The evolution of the world after the evolution is

$$\begin{array}{ll} start(S_0) = 0 & end(S_0) = 3 \\ start(S_1) = 3 & end(S_1) = 5 \\ start(S_2) = 5 & end(S_2) = 7 \end{array}$$

[42]We have assume that the algorithm return consistent

$$start(S_3) = 7 \quad end(S_3) = 11$$
$$start(S_4) = 11$$

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 exists).

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 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 thus none fluent change 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 create 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 figure 6 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 the time point 26.

Example 2 (continued)

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

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

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

Now we 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 two above cases is that there we had $start(S_1) = 22$, while now we want to find a situation S_1 s.t if none action 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 return 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 $C_1 \wedge .... \wedge C_n$. Then it must be the case that one of the $C_1, ..., C_n$ is false. Assume that $C_i = f_1 \vee .... \vee f_m$ is false. Then all fluents $f_j, j = 1, ..m$ are false. Then by the 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 the steps 3 and 4 of the algorithm of production of the static rule we have that: $G_{f_p}(t, ..) = G' \vee (\neg \bigwedge f_j(t_j, j = 1, ..m, j \neq p)$. If all fluents $f_j, j = 1, ..m$ are false then $(\neg \bigwedge f_j, j = 1, ..m, j \neq p)$ 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 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 Conclusions and Future Work

6.1 Summary of Contribution

In this thesis we examined three infamous problems the **frame**, the **ramification** and the **qualification** problems in temporal databases. We concentrated on the ramification problem.

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 assumption. This means that nothing changes except if an action takes places. This assumption simplifies the solution of the ramification problem, but it is quite restrictive, because in a temporal reasoning setting the persistence of fluents is unreasonable. This happens because when time considerations are imported, one action could have as effect that the fluent f holds for t time points after the execution of an action. The solutions which are based on the idea of 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. But this has two major disadvantages. First, if some other action cancels the no persistent effect then the corresponding action must be cancelled. Second, the number of actions may greatly increase because we must define one action for each non-persistent effect of each action(e.g. if each action has two non-persistent effects then if A is the number of actions we need $2 \times A$ **extra actions**). Thus the complexity increases significantly.

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

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, possibly, for many future situations as well. This means that the world being modelled may change from one situation to another while the direct and/or indirect effects of actions still hold. Also, in this time span other actions may occur leading to many different situations.

We addressed the ramification problem in temporal database for the following cases:

- Sequential execution of the actions. In this case, we propose a linear correspondence between actions-time-situations as shown in figure 6 of section 3. There are three different assumptions in this case:
  - The effects of the action start to hold for the next time point. We propose an algorithm which solves the ramification problem in this case.

  - The effects of an action start to hold after some time points. In this case, there is the problem that an action may cancel the effects of another action before them starting to hold. We extended the algorithm which we have proposed for the first case.
  - The actions have duration. In this case, the effects must be determined with reference to the start, the end and the duration of the actions. If all direct and indirect effects can be described by reference to the start and the end of the action then we can assume that one action with duration is equivalent to two instantaneous actions: one for the start and one for the end. In this case, the dynamic rules must be defined for the instantaneous actions. Usually the duration of an action is unknown before its end. So we cannot describe the direct and indirect effects of an action with reference to the start and to the end. For each action we propose a natural action which "refreshes" the effects of the action when they cease to hold.

    Additionally to the above, some effects of the action could depend on the duration of the action. This means that if one action could produce some effects after it has complete a time points of continuous execution. In the opposite case, it does not produce these effects. We proposed a new extension of the algorithm.

- Concurrent execution of two or more actions
  - The effects of the action start to hold for the next time point. In this case, instantaneous actions execute concurrently. The direct and indirect effects of an action do not start necessarily from the next time moment. This means that two or more actions cannot necessarily be executed concurrently even if their preconditions hold. It must be determined that the direct and indirect effects of these actions are consistent not only in the next time moment but in the future, as well. Also we must ensure that all direct effects of all actions executing concurrently hold. This means that it is not possible for one action to cancel the effects of other actions which execute concurrently. If there is not consistent situation which encapsulates all direct and indirect effects of the actions, we reject the execution. We propose an algorithm for the solution of the ramification problem in that case.
  - The effects of action start to hold after some time points. In that case we must address all issues as in the case of sequential action execution and additionally we must ensure the consistency between the effects of the actions which executed concurrently. There is also the case that the effects of the actions are inconsistent in a time point in the future. We propose an algorithm for the solution of the ramification problem in that case.
  - The actions have duration. In that case we must ensure all things which we referred to in the case of sequential action execution and all things which we referred to in the previous case. The new algorithm, which we propose for that case, solves the ramification problem for all previous cases.

- An action can change the beliefs about the past. The linear correspondences of figure 6 of section 3 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 we repeat the execution from the time point (in the past) to which the effects of the action referred. Thus, we need branching axes for the situations and actions. The time axis remains linear. We propose the correspondence appearing in figure 10 of section 6. When an action changes the past we start two new linear axes: one for the situations and one for the actions. As we may observe, each time there is a linear line in the branching axis of situation which is the "actual" line. This line contains the situations which are the history up to time point t. Because we repeat the execution of actions after an action changes the beliefs about the past there is the problem of infinite loops. We distinguished three cases and provided a solution in each one of them:

  - When an action could change the values of all the fluents in the past. In that case the only problem is that of infinite loops.
  - When an action could change the value of only some fluents in the past. In that case, we must distinguish between the set F_P of fluents which cannot change their truth value in the past and the set F_S of fluents which could change their truth value in the past.

    In addition to the above problem, we must ensure that the fluents which belong in F_P do not change their value in the past. We solved this case by expanding 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 figure 6 is appropriate. 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 future work we intend to address the ramification problem in the case that two or more action execute concurrently and their effects refer to the past or to the future. Consider the following execution

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

These action have as direct effects

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

respectively. This is consistent but there is the following static rule

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

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 two actions and second to accept

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

In the latter case, there is the problem of determining in which situation executed the dynamic rules and how we construct the actual line between the earlier time point in the past (which some action referred to) and the time point in the future. For example, we executed first the action which referred in the past and afterwards the action which referred in the future, etc.

Also as future work we want to examine the ramification problem in the case when the effects of actions are non-deterministic. For example assume the example with the public worker and assume that there is a new action *grant_accolade* which has as direct effects to take bonus or take a pardon but not both of them. Consider the following execution

$$\begin{array}{l} occur(misdemeanor(p), 8) \\ occur(grant\_accolade(p), 10) \\ occur(grant\_promotion(p), 11) \end{array}$$

If the action *grant_accolade*, which takes place at time point 10, has as direct effect to grant pardon to the public worker p then the action *grant_promotion* executes at time point 11. Otherwise, if it has as direct effect to grant bonus to the public worker p then the action *grant_promotion* cannot execute at time point 11.

A case in which the effects of actions are non-deterministic is in robotics. The movements and the actions of some robot could be non-deterministic. For example assume that a robot move all things from the room to the room B. The robot can move the things with many different ways.

6.3 Conjectural

The ramification problem is of great importance to database systems. Database users and designers may not known exactly all the indirect effects of their transactions. This means that the users/designer can execute a transaction which has as result to violate the integrity constraints. Thus, we need an automatic way which determines the indirect effects of transactions and enables the verification of constraints.

A solution to the ramification problem permits to the designers to realize the effects of their design. For example a transaction can produce an inconsistent situation or can produce a situation which contains undesirable indirect effects. This means that the designers can discover erroneous specifications. This discovery is very difficult or impossible to be done manually. The solution of the ramification problem admir to the designers to understand the effects of their transactions and redesign the database if this is necessary.

The solution to ramification problem is necessary in databases because it enables the design of correct, reliable and consistent databases. All the solutions to the ramification problem in conventional databases are based on the idea of the persistence of fluents. This means that nothing changes until an action takes place. In conventional databases the persistence of fluents hold. Thus these solutions are satisfactory for the non temporal databases. The above solutions cannot produce automatically the non persistent effects. If such effects exists, they have to be described manually. This is very difficult and impractical in the large databases with complex transactions. In temporal databases the predicate and function values can change as time progresses without any action taking place. Thus the above solutions cannot solve the ramification problem in temporal databases.

We suggest a new solution to the ramification problem which based in the situation calculus. In order to acheive that we extented the situation calculus. The extended situation calculus encapsulate the time and provide a way for description of the actions and situations over the time. Also this extension permit us to describe the non-persistence effects. Our solution produce all effects (persistence and non-persistence) of actions automatically. This mean that our solution address the ramification problem in temporal databases.

References

[1] J. B. Amsterdam. Temporal reasoning and narrative conventions. In J. F. Allen, R. Fikes, and E. Sandewall, editors, Proceedings of International Conference on Principles of Khowledgwe Representation and Reasoning(KR), pages 15-21, Cambridge, MA 1991.

[2] Andrew B. Baker. Nonmonotonic reasoning in the frameworke of situation calculus. Artificial Intelliegence, 49:5-23, 1991.

[3] Alessandro Artale and Enrico Franconi.A computational account for a description logic of time and action. 4th International Conference on Knowledge Representation and Reasoning (KR'94), Morgan Kaufmann, San Mateo CA, May 1994.

[4] Alessandro Artale and Enrico Franconi. Hierarchical Plans in a Description Logic of Time and Action. 1995 International Workshop on Description Logics (DL'95), Rome, Italy, June 1995. Also IJCAI'95 workshop on The Next Generation Of Plan Recognition Systems, August 1995.

[5] Alessandro Artale and Enrico Franconi (1998). A Temporal Description Logic for Reasoning about Actions and Plans. Journal of Artificial Intelligence Research (JAIR) Vol. 9, pages 463-506, December 1998.

[6] Alessandro Artale and Enrico Franconi (1999). Representing a Robotic Domain using Temporal Description Logics. Journal of Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM), special Issue on Temporal Logic in Engineering, Vol. 13, No. 2, April 1999.

[7] Andrew B. Baker. Nonmonotonic reasoning in the framework of situation calculus. Artificial Intelligence Journal, 49:5-23,1991.

[8] Chitta Baral and Michael Gelfond. Representing concurrent actions in extended logic programming. In R. Bajcsy, editor, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 866-871, Chambery, France, August 1993. Morgan Kaufmann.

[9] Chitta Baral and Jorge Lobo. Defeasible specifications in actions theories. In M. E. Pollack, editor, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 1441-1446, Nagoya, Japan, August 1997. Morgan Kaufmann.

[10] Chitta Baral and Michael Gelfond and Alessandro Provetti. Representing actions. Laws, observations and hypothesis. Journal of Logic Programming, 31(1-3):201-243, 1997.

[11] Daniel G. Bobrow, editor. Artificial Intelligence 13. Special Issue on NonMonotonic Reasoning. 1980.

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

[13] Gerhard Brewka and Joachim Kertzberg. How to do things with worlds: On formalizing actions and plans. Journal of Logic and Computation, 3(5):517-532, 1993.

[14] Sven-Erik Bornscheuer and Michael Thieslscher. Explicit and implicit inderminism. Reasoning about uncertain and contradictory specifications of dynamic systems. Journal of Logic Programming, 31(1-3): 119-155, 1997.

[15] Sven-Erik Bornscheuer and Michael Thieslscher. Representing concurrent actions and solving conflicts. In B. Nebel and L. Dreschler-Fischer, editors, Proceedings of the German Annual Conference on Artificial Intelligence (KI), Volume 861 of LNAI, pages 16027, Saarbrucken, Germany, September 1994. Springer.

[16] K. Clark. Negation as failure. Logic and Data Bases, 1978.

[17] Marie-Odile Cordier and Pierre Siegel. A temporal revision model reasoning about world change. In B. Nebel, C. Rich, and W. Swartout, editors, Proceedings of the International Conference on Prnciples of Khowledge Representation and Reasoning(KR), pages 732-739, Cambridge, MA, 1992. Morgan Kaufmann

[18] Mark Denecker, Daniele Theseider Durpe, and Kristof Van Belleghem. An inductive definition approach to ramifications. Electronic Transactions on Artificial Intelligence, 1998.

[19] J. P. Denecker and Danny de Schaub. Representing incomplete Khowledge in abductive logic programming. Journal of Logic and Computation, 5(5):553-577, 1995.

[20] Patrick Doherty, Joakim Gustafsson, Lars Karlsson, and Jonas Kvarnstrom. TAL Temporal action Logics language specification and tutorial. Linkoping Electronic Articles in Computer and Information Science, 1998.

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

[22] Joseph J. Finger. Exploiting Constraints in Design Sythesis. Phd thesis, Standford University, CA, 1987.

[23] R. Fikes and N. J. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2:189-208, 1971.

[24] Bertram Fronhofer. The Action-as-Implication Paradigm. CS Press Munchen, 1996.

[25] A. Fusaoka. Situation Calculus on a Dense Flow of Time. Proceedings of the AAAI National Conference on Artificial Intelligence, pages 633-638, 1996

[26] Hector Geffner. Causal theories for nonmonotonic reasoning. In Proceedings of the AAAI National Conference on Artificial Intelligence, pages 524-530, Boston, MA, 1990.

[27] Hector Geffner. Deafult Reasoning: Causal and Conditional Theories. MIT Press, 1992.

[28] Michael Gelfond and Valdimir Lifshitz. Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing, 9:365-385, 1991.

[29] Michael Gelfond and Valdimir Lifshitz. Representing Actions in Extended Logic Programming. In K. Apt, editor, Proceeding of the International Joint Conference and Symposium on Logic Programming (IJCSLP), pages 559-573, Washington, 1992. MIT Press.

[30] Michael Gelfond and Valdimir Lifshitz. Representing actions and change by logic programs. Journal of Logic Programming, 17:301-321, 1993.

[31] Michael Gelfond and Valdimir Lifshitz. Action Languages. Electronic Transactions on Artificial Intelligence, 1998.

[32] Giuseppe De Giacomo, Yves Lesperance, and Hector Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In Proceedings of the Fifteenth International Joint Conference on AI (IJCAI'97), pages 1221-1226, Nagoya, August 1997.

[33] Giuseppe De Giacomo, Yves Lesperance, and Hector Levesque. ConGolog, a concurrent programming language based on the situation calculus. Artificial Intelligence, 121(1-2):109-169, 2000

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

[35] M. Ginsberg and D. Smith. Reasoning about action II: A possible words approach. Artificial Intelligence, 35:311-342, 1988.

[36] Enrico Giunchiglia, G. Neelekantan Kartha, and Vladimir Lifschitz. Actions with indirect effects (extended abstract). In C. Boutiller and Goldszmidt, editors, Extending Theories of Actions: Formal Theory and Practical Applications, Volume SS-95-07 of AAAI Spring Symposia, pages 80-85, Stanford University, March 1995. AAAI Press.

[37] E. Giunchiglia, N. Kartha and V. Lifschitz, Representing action: indeterminacy and ramifications, Artificial Intelligence, Vol. 95, 1997, pp. 409-443.

[38] Enrico Giunchiglia. Determining ramifications in the situation calculus. In L. C. Aiello, J. Doyle, and S. Shapiro, editors, Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR), pages 76-86, Cambridge, MA, November 1996. Morgan Kaufmann.

[39] E. Giunchiglia and V. Lifschitz, Action languages, temporal action logics and the situation calculus, in Working Notes of the IJCAI-99 Workshop on Nonmonotonic Reasoning, Action, and Change, 1999.

[40] E. Giunchiglia and V. Lifschitz, An action language based on causal explanation: preliminary report, in Proc. AAAI-98, 1998, pp. 623-630.

[41] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain and H. Turner, Nonmonotonic causal theories, Artificial Intelligence, to appear.

[42] Moises Goldszmidt and Judea Pearl. Qualitative probabilities for default reasoning, belief revision, and causal modeling. Artificial Intelligence, 24(1-2):57-112, 1996.

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

[44] J. Gustafon. Embrancing occlusion in specifying the indirect effects of actions. In L.C. Aiello, J.Doyle, and S. Shapiro, editors, Proceedings of the International Conference on Principles of Knowledge Representation and Reasonin (KR), pages 87-98, Cambridge, MA, November 1996. Morgan Kaufmann.

[45] A. Haas. The Case for Domain-Specific Frame Axioms. In F. Brown, editor. The frame problem in artificial intelligence. Proceedings of the 1987 workshop, pages 343-348, Los Altos, California.

[46] D. Hanks, S. McDermott. Nonmonotonic logic and temporal projection. Artificial Intelligence, 1987.

[47] Steffen Holldobler and Josef Schneeberger. A new deductive approach to planning. New Generation Computing, 8:225-244, 1990.

[48] Steffen Holldobler and Hans-Peter Storr. Complex plans in the fluent calculus. In S. Holldobler, editor, Intellectics and Computational Logic. Kluwer Academic, 1999.

[49] Steffen Holldobler and Michael Thielscher. Computing change and specificity with equational logic programs. Annals of Mathematics and Artificial Intelligence programs with equality. Journal of Logic Programming, 1(3):211-223, 1984.

[50] Christoph S. Herrman and Michael Thielsher. Reasoning about continuous processes. In B. Clancey and D. Weld, editors, Proceeding of AAAI National Conference on Artificial Intelligence, pages 639-644. Portland, OR, August 1996, MIT Press.

[51] Steffen Holldobler and Josef Schneeberger. A new deductive approach to planing. New Generation Computing, 8:225-244,1990.

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

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

[54] G. Neelakantan Kartha and V. Lifschitz. Actions with indirects effects. In J. Doyle, E. Sandewall, and P. Torasso, editors, Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR), pages 341-350. Bonn, Germany, May 1994. Morgan Kaufmann.

[55] G. Neelakantan Kartha. Soundeness and completeness theorems for three formalizations of actions. In R. Bajscy, editors, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 724-729, Chambery, France, August 1993. Morgan Kaufmann.

[56] G. Neelakantan Kartha. Two counterexamples related to Baker's approach to the frame problem. Artificial Intelligence, 69(1-2):379-391, 1994.

[57] M. Koubarakis, Foundations of Temporal Constraint Databases, Ph.D. thesis, Computer Science Division, Dept. of Electrical and Computer Engineering, National Technical University of Athens, February 1994

[58] M. Koubarakis, Tractable Disjunctions of Linear Constraints: Basic Results and Applications to Temporal Reasoning, Theoretical Computer Science, Vol. 266, pages 311-339, September 2001.

[59] R.A. Kowalski. Database updates in the event calculus. Journal of Logic Programming, 1992.

[60] J. Lee and V. Lifschitz, Describing additive fluents in action language C+, in Proc. IJCAI-03

[61] V. Lifshitz. On the logic of causal explanation. Artificial Intelligence, 451-465, 1997.

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

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

[64] V. Lifshitz. Restricted monotonicity. In Proceedings of the AAAI National Conference on Artifical Intelligence, pages 432-437, Washington DC, July 1993.

[65] V. Lifschitz,Two components of an action language, Annals of Mathematics and Artificial Intelligence, Vol. 21, 1997, pp. 305-320.

[66] V. Lifschitz, Situation calculus and causal logic, in Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning, 1998, pp. 536-646.

[67] V. Lifschitz, Action languages, answer sets and planning, in The Logic Programming Paradigm: a 25-Year Perspective, Springer Verlag, 1999, pp. 357-373.

[68] F. Lin. Embracing causality in specifying in the indirect effects of actions. In Proceedings of International Joint Conferrence of Artificial Intelligence, 1995.

[69] F. Lin. Embracing causality in specifying in the indeterminate effects of action. In Proceedings of American Assocition for Artificial Intelligence, 1996.

[70] F. Lin and R. Reiter. How to progress a database (and why) I: Formal foundations. In In Proc. Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR-94), 1994.

[71] F. Lin and R. Reiter. How to progress a database II: The strips connection. In In Proc. IJCAI-95, 1995.

[72] F. Lin and R. Reiter. Rules as actions: A situation calculus semantics for logic programs. Journal of Logic Programming, Special issue on Reasoning about Action and Change, 31:299-330, 1997.

[73] F. Lin and Y. Shoham. Provably correct theories of action. Journal of ACM, 42(2):293-320, 1995.

[74] F. Lin and Y. Shoham. On non-forgetting and minimal learning. In N. Asher, K. Korta, and J. Ezquerro, editors, To appear in Proc. of the 1993 International Coll. on Cognitive Science. Kluwer Academic Publishers, 1996.

[75] Fangzhen Lin. Applications of the situation calculus to formalizing control and strategic information: The Prolog cut operator. In Proceedings of IJCAI-97, pages 1412-1418, 1997. (IJCAI-97 Distinguished Paper Award).

[76] Fangzhen Lin and Hector J. Levesque. What robots can do: Robot programs and effective achievability. Artificial Intelligence, 101:201-226, 1998.

[77] Fangzhen Lin and Ray Reiter. State constraints revisited. Journal of Logic and Computation, 4(5):655-678, 1994.

[78] Sheila A. McIlraith. A closed-form solution to the ramification problem (sometimes). In In Proceedings of the IJCAI'97. Workshop on Nonmonotonic Reasoning, Action and Change (NRAC-97), 1997.

[79] Sheila A. McIlraith. Explanatory diagnosis: Conjecturing actions to explain obsevations. In In Proceedings of the Eighth International Workshop on Principles of Diagnosis (DX'97), pages 69-78, 1997.

[80] Sheila A. McIlraith. Representing actions and state constraints in model-based diagnosis. In In Proceedings of the National Conference on Artificial Intelligence (AAAI-97), pages 43-49, 1997.

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

[82] N. McCain and Hudson Turner. A causal theory of ramifications and qualifications. In C. S. Mellish, editor, Proceedings of the International Joint Conference on Artifical Intelligence (IJCAI), pages 1978-1984, Montreal, Canada, August 1995.

[83] N. McCain and Hudson Turner. A causal theory of action and change. Proceedings of the AAAI National Conference on Artifical Intelligence, 1997.

[84] Rob Miller and Murray Shanahan. The Event Calculus in Classical Logic - Alternative Axiomatisations. Linkping Electronic Articles in Computer and Information Science, 4(16), 1999.

[85] J. Mylopoulos, V. Chaudhri, D. Plexousakis, A. Shrufi and T. Topaloglou, Building knowledge base management systems, The VLDB Journal, May 1995.

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

[87] Nikos Papadakis and Dimitris Plexousakis. The Ramification and Qualification Problems in Temporal Databases. Proceedings of the 2th hellenic Conference on AI, pp. 18-30 Lecture Notes on Artificial Intelligent vol 2308, 10-11 April 2002, Thessaloniki, Greece.

[88] Nikos Papadakis and Dimitris Plexousakis. Action with Duration and Constraints: The Ramification problem in Temporal Databases. 14th IEEE International Conference on Tools with Artificial Intelligent, pages 83-90, 4-6 November 2002 , Washington D.C.

[89] Nikos Papadakis and Dimitris Plexousakis. Action with Duration and Constraints: The Ramification problem in Temporal Databases. International Journal on Artificial Intelligent, special issue, first issue 2004.

[90] Nikos Papadakis and Dimitris Plexousakis. Addressing the Ramification Problem in Temporal Context: The Case of Concurrent Actions. International Conference on Tools with Artificial Intelligent, pages 545-551. 3-5 November 2003 , Sacramento USA.

[91] E. Pednault. ADL: Exploring the Middle Ground between STRIPS and the Situation Calculus. In R.J. Brachman, H. Levesque, and R. Reiter, editors, Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR' 89), pages 324-332. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1989.

[92] J. Pearl. Embrancing causality in default reasoning. Artifical Intelligence, 1998.

[93] J. Pearl. Causation, action and counterfactuals. Proceedings of the Sixth Conference on Theoritical Aspects of Rationality and Khowledge, 1996.

[94] J. Pearl. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo, CA, 1988.

[95] J. Pearl. A probabilistic calculus of actions. In R. Lopez de Mantaras and D. Poole, editors, Proceedings of the conference on Uncertainty in Artificial Intelligence (UAI), pages 454-462, San Mateo, CA, 1994, Morgan Kaufmann.

[96] Peppas, Pavlos and Wayne Wobcke: On the Use of Epistemic Entrenchment in Reasoning about Action. Presented at: European Conf on Artificial Intelligence, 1992. Venue: Vienna, Austria, 3.8-7.8. Proceedings published by John Wiley and Sons.

[97] Dimitris Plexousakis: Integrity Constraint and Rule Maintenance in Temporal Deductive Knowledge Bases. Proc. of the 19th International Conference on Very Large Data Bases (VLDB), pages 146-157, August 1993.

[98] Dimitris Plexousakis: Compilation and Simplification of Temporal Integrity Constraints. International Workshop on Rules in Database Systems (RIDS), Athens, Greece, September 25 - 27, 1995.

[99] Dimitris Plexousakis. Maintenance of Integrity Constraints in Temporal Deductive Knowledge Bases. Phd Thesis, Dept. of Computer Science, Univ. of Toronto, Jan. 1996.

[100] Dimitris Plexousakis, John Mylopoulos: Accomodating Integrity Constraints During Database Design. Proceedings of EDBT 1996, pages 497-513

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

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

[103] Javier Pinto. Occurrences and narratives as constraints in the branching structure of the situation calculus. Journal of Logic and Computation, 8:777-808, 1998.

[104] R. Reiter A logic for default reasoning. Artificial Intelligence, 13:81-132, 1980.

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

[106] R. Reiter and Y. Zheng. Scheduling in the situation calculus: A case study. Annals of Mathematics and Artificial Intelligence. Special issue on logic programming, nonmonotonic reasoning and action, 1996.

[107] Ray Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, pages 359-380. Academic Press, San Diego, CA, 1991.

[108] Ray Reiter. Proving properties of states in the situation calculus. Artificial Intelligence, 64:337-351, 1993.

[109] Ray Reiter. Sequential, temporal golog. In Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98), pages 547-556, Trento, Italy, 1998.

[110] Ray Reiter. Narratives as programs. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR2000), San Francisco, CA, 2000. Morgan Kaufmann.

[111] Ray Reiter. On knowledge-based programming with sensing in the situation calculus. ACM Transactions on Computational Logic (TOCL), 2(4):433-457, October 2001.

[112] Raymond Reiter. The projection problem in the situation calculus: A soundness and completeness result, with and application to database updates. In Proceedings of the First International Conference on AI Planning Systems, pages 198-203, College Park, Maryland, June 15-17 1992.

[113] Raymond Reiter. Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press, 2001.

[114] E.Sandewall. Systematic assessment of temporal reasoning methods for use in autonomous systems. In B. FronHofer, editor, Workshop on Reasoning about Action and Change at IJCAI, pages 21-36, Chambery, August 1993.

[115] E.Sandewall. The Representation of Knowledge about Dynamic Systems. Oxford University Press 1994.

[116] E.Sandewall. The range of applicability of some non-monotonic logics for stict inertia. Journal of Logic Computation, 4(5):581-615,1994.

[117] E.Sandewall. Reasoning about actions and change with ramification. In Computer Science Today, volume 1000 of LNCS. Springer, 1995.

[118] E.Sandewall. Assesments of ramification methods that use static domain constraints. Proceeding of International Conference on Principles of Knowledge Representation and Reasoning, 1996.

[119] E.Sandewall. Cognitive robotics logic and its metatheory: Features and fluents revised. Linkoping Electronic Articles in Computer and Information Science, 1998.

[120] K. Stergiou and M. Koubarakis, Backtracking Algorithms for Disjunctions of Temporal Constraints, Artificial Intelligence, Vol. 120 (1) (2000) pp. 81-117.

[121] E. Ternovskaia. Automata Theory for reasoning about action. In Proceeding of International Joint Conference on Artificila Intelligent, 1999.

[122] M. Thielsher. Challenges for Action Theories (Book), 2000, Springer Verlag.

[123] M. Thielsher. An analysis of systematic approaches to reasoning about actions and change. In P. Jorrand and V. Sgurev, editors, International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA), pages 195-204, Sofia, Bulgaria, September 1994. World Scientific.

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

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

[126] M. Thielescher. Continuous processes in Fluent Calculus. In Hybruid Systems and AI: Modeling and Control of Discrete + Continuous Systems, Volume SS-99-05 of AAAI Spring Symposia, pages 186-191, Standrord University, March 1999. AAAI Press.

[127] M. Thielescher.Nondeterministic actions in the fluent calculus: Disjunctive state update axioms. In S. Holldobler, editor, Intellectics and Computational Logic. Kluwer Academic, 1999.

[128] M. Thielescher. Causality and the qualification problem. In L. C. Aiello, J. Doyle, and S. Shapiro, editors, Proceedings of the International Conference on Principles of Khowledge Representation and Reasoning (KR), pages 51-62, Cambridge, MA, November 1996. Morgan Kaufmann.

[129] M. Thielscher. Qualified ramifications. In B. Kuipers and B.Wbber, editors, Proceedings of the AAAI National Conference on Artificial Intelligence, pages 466-471, 1997

[130] M. Thielscher. From Situation Calculus to Fluent Calculus: State update axioms as a solution to the inferentila frame problem. Artificial Intelligence, 1999.

[131] M. Winslett. Reasoning about action using a possible models approach. In Proceeding of the AAAI National Conference on Artifical Intelligence, pages 89-93, Saint Paul, MN, August 1988.

[132] Choong-Ho Yi. Towards the assesment of logics for concurrent actions. In D. M. Gabbay, editor, Proceedings of the International Conference on Formal and Applied Practical Reasoning (FAPR), volume 1085 of LNAI, pages 679-690, Bonn, Germany, June 1996. Springer.

[133] Alvaro del Val and Yoan Shoham. Deriving properties of belief update from theories of action(II). In R. Bajscy, editors Proceeding of the International Joint Conference on Artifical Intelligence (IJCAI), pages 732-737, Chambery, France, August 1993. Morgan Kaufmann.

[134] Yan Zang and Norman Y. Foo. Reasoning about persistence: A theory of actions. In R. Bajscy, editor, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 718-723, Chambery, France, August 1993. Morgan Kaufmann.

[135] Yan Zang. Compiling cauasality into action theories. In Proceeding of the Symposium on Logical Formalization of Commonsense Reasoning, pages 26-270. Stanford, CA, January 1996.