Hypothetical Temporal Reasoning with History Encoding
(Extended Version)

Marcelo Arenas and **Leopoldo Bertossi***
Pontificia Universidad Catolica de Chile
Escuela de Ingenieria
Departamento de Ciencia de Computacion
{marenas,bertossi}@ing.puc.cl

January 20, 1998

Abstract

In this paper we integrate a history–encoding based methodology developed by Chomicki for checking dynamic database integrity constraints into a situation-calculus based specification of database updates as proposed by Reiter. By doing this, we are able to: (1) Answer queries about a whole hypothetical evolution of a database, without having to update the entire database and keep the information associated to all the generated states, (2) Stating and proving dynamic integrity constraints as static integrity constraints, (3) Transform non markovian preconditions for updates into markovian preconditions. All this results, although motivated by problems in database theory, can be applied to theories of action and change in general. The methodology presented here takes advantage of translations between first order past temporal logic formulas and state bounded situation calculus queries and constraints.
Keywords: database updates, situation calculus, past temporal logic, temporal queries, integrity constraints.

1 Introduction

We are interested in answering queries about different states in the evolution of a relational database, when the database is virtually updated by the execution of a sequence of primitive actions. For example, we want to consider queries of the form "Has it always been the case that the database has satisfied a certain property?", or "Has there been a state of the database where a certain property has been satisfied?", or "Has the salary of some employee decreased along the database evolution?". This problem was originally proposed by Raymond Reiter in the context of his specifications in the Situation Calculus (SC) [13] of transaction based database updates [16], and addressed, in the same context, in [19, 18, 20]. The main ingredient of Reiter's specifications are the "successor state axioms for tables", that fully specify when a tuple belongs to a table after executing an arbitrary transaction, in terms of the contents of the database at the execution state.

*Current and Correspondence Address: TU Berlin, FB Informatik, CIS - Sekr. EN7, Einsteinufer 17, D-10587 Berlin, Germany. bertossi@cs.tu-berlin.de

We call this kind of queries "hypothetical" because we have an initial, physical, database at a state S_0 and a list T of primitive transactions $A_1, \ldots, A_n$, that virtually update the database, producing new states $S_1, \ldots, S_n$, and we want to answer the query without physically updating the whole database accordingly (and possibly keeping the data for every intermediate state). We are interested in querying this virtual evolution of the database.

We decided to call this kind of queries about the evolution of the database (hypothetical or not), "temporal queries", because of their similarity with dynamic integrity constraints, which are also called "temporal integrity constraints [6], they refer to several database states. In [19] they were called "historical queries", but this name may make think of historical databases, but here we are not interested in valid time. Although we do not have explicit time, the implicit time if closer to transaction time and our sequences of database states are rather rollback databases. All this terminology is clearly explained in [21].

We consider static queries, that is, about one single database state, as a particular case of dynamic queries. In the same spirit, static integrity constraints, that refer to (all) individual database states, are considered a particular case of dynamic integrity constraints [16, 15]. For example, a typical dynamic integrity constraint is "The salary of an individual cannot decrease along the database evolution", whereas "There is a functional dependency between table $T1$ and table $T2$" is a static integrity constraint.

In [6], Chomicki considers the problem of checking temporal constraints stated in First Order Past Temporal Logic (FOPTL). He has a sequence of transactions that are executed, so the updates are not virtual. In order to minimize the cost of checking the truth of a temporal constraint, he progressively updates newly defined relations, or auxiliary views, R_α's, that correspond to the temporal subformulas , α's, that appear in the constraint, and encodes the history evolution up to a certain database state in these auxiliary views. We can say that those historical views update the information that is relevant to give an answer to the query once the final state is reached; they carry until the last state the relevant historical information needed to give the final answer, so a final, non temporal, query is posed to the final state. Chomicki does not have any explicit notion of relevance as in [19, 18, 20].

In this paper we start by considering the problem of answering dynamic queries. There is no big difference with the problem of checking dynamic constraints, except for the fact that we *prove* the ICs from the specification of the database evolution [16, 3], instead of *checking* them. Nevertheless, the fact that we do have and use a specification of the database evolution, but Chomicki doesn't, makes a big difference. He does not consider explicit named transactions at all. In our case, our primitive transactions can be more complex that just deletions and insertions. All we need is a logical specification of their effects. In addition to this, having the specification of the dynamics allows us to reason hypothetically.

Our approach in this paper will be the combination of Chomicki's history encoding with the possibility of reasoning about the database evolution without having to physically update the database. In this sense we will be in position to apply Chomicki's methodology in hypothetical reasoning and this, with more general primitive transactions.

Although Chomicki uses temporal logic and we have specifications in the situation calculus, we will manage to translate Chomicki's formalism and algorithms to the context of Reiter like specifications: Chomicki's views R_α's could be defined as views with respect to the Reiter's specification of the database dynamics. In this case, it will be easy to derive successor state axioms for these views, since their contents at each state depends on their contents in the previous state and the transaction executed to reach the current state. It will turn out that Chomicki's task, in our setting, corresponds to solving the "temporal projection problem" [9] for a query stated in terms of the R_α's at the final state in the transaction list, that is, the problem of querying a virtual state obtained after the (hypothetical) execution of the whole transaction log.

We can say that while Chomicki answers the query by positioning himself at the final physical state of the database, we query a single future virtual state from the initial, physical, state.

We use the methodology developed for answering queries to two other important problems in databases. Firstly, we propose a mechanism for transforming preconditions for the actions execution that depend on the history of the database, into new preconditions that depend on the execution state only. Secondly, we develop a methodology for transforming dynamic integrity constraints into static integrity constraints. By doing this, we are in position to apply automated proofs techniques for proving static integrity constraints to the case of dynamic integrity constraints [4, 17].

This paper is structured as follows. In section 2 we will briefly describe the situation calculus based specifications of database updates. In section 3 we briefly review temporal logic and Chomicki's methodology. In section 4 we compare the situation calculus and temporal logic in the context of our problem. In section 5 we show how to specify the dynamics of the auxiliary, history encoding, views. In section 6 we address the problem of answering the hypothetical queries with the specification of the dynamics of the expanded database. In section 7 we apply our methodology to the problem of transforming non Markovian transactions, i.e. transaction whose preconditions are temporal, into Markovian transactions. In section 8 we use the machinery developed in previous sections to the problem of transforming dynamic integrity constraints into static integrity constraints. In section 9 we discuss other issues around the applicability and relationship of the methodology to other problems in the database domain. In section 10 we compare our work with other approaches in the literature. Finally, in section 11, we draw some conclusions.

2 Situation Calculus based Specifications of Database Updates

Characteristic ingredients for a situation calculus language $\mathcal{L}$, besides the usual symbols of first–order languages for domain descriptions, are: (1) Sorts *action*, *state*, and *object* (this last one for the objects in the domain; this sort could be split into sub–sorts if necessary); (2) Predicate symbols of the sort $(object, \ldots, object, state)$ to denote tables ("fluents" in the AI literature). These are predicates that depend on the state of the world; (3) Operation symbols of the sort $(object, \ldots, object) \rightarrow action$ for denoting primitive actions with objects as parameters (or applied to objects); (4) A constant S_0 to denote the initial state; (5) An operation symbol *do* of sort $(action, state) \rightarrow state$, that executes an action at a given state producing a successor state.

In this first–order language there are variables for individuals of each sort, so it is possible to quantify over objects, actions, and situations. We usually denote variables for actions with a, and variables for states, with s.

The specification of a dynamically changing world, by means of an appropriate language of the situation calculus, consists in stating the laws of evolution of the world. This is typically done by specifying: (1) Fixed, state independent, but domain dependent knowledge about the objects of the world; (2) Knowledge about the state of the world at the initial situation S_0, that is given in terms of formulas that do not mention any state besides S_0; (3) Preconditions for performing the different actions (or making their execution possible). We introduce a predicate *Poss* in $\mathcal{L}$ of sort $(action, state)$ to say that the execution of an action is possible in a state; (4) The immediate (positive or negative) effects of actions in terms of the fluents whose truth values we know are changed by their execution.

In [16], Raymond Reiter used the SC to specify the evolution of a database in terms of the way it changes when the primitive transactions are exectued. This is a nice first–order formalism for specifying this behaviour, with a clear and well studied semantics and with deductive methods that can be automated and applied in different resasoning tasks around the specification [3]. For some reasonig tasks Reiter adds a second–order induction principle on states. We have applied automated induction to prove integrity constraints from specification of transaction based database updates in the SC [4].

In Reiter's formalism, the knowledge contained in items (1) and (2) above is considered the initial database D_{S_0}. The information given in item (3) is formalized by means of a precondition axiom of the form $\forall \bar{x} \forall s(Poss(A(\bar{x}), s) \equiv \pi_A(\bar{x}, s))$, for each action name A.

In the specification there is, in addition, for each primitive table $F(\bar{x}, s)$, a Successor State Axiom (SSA) of the form

$$(\forall \bar{x}, a, s) Poss(a, s) \supset F(\bar{x}, do(a, s)) \equiv \varphi_F(\bar{x}, a, s). \tag{1}$$

Here, φ is a formula that contains s as the only state term, in particular, it does not contain the *do* symbol. We say that the formula is *simple in s*. This axiom completely determines the truth value of the occurrence a tuple in table F after an arbitrary, and legal, primitive transaction has been executed.

The specification also contains Unique Names Axioms (UNA) for states and actions[1].

As said before, we also assume a second–order induction axiom as a part of the foundational axioms of the SC:

$$\forall P\ ((P(S_0) \wedge \forall a, s((P(s) \supset P(do(a, s))) \supset \forall s P(s)). \tag{2}$$

On the basis of this induction principle it is possible to define an *accessibility relation* between states, $s_1 \leq s_2$, in particular, the *accessible states* as those states s that satisfy, $S_0 \leq s$. They are exactly those obtained from the initial state by the performance of a finite sequence of legal transactions: (a) The initial state S_0 is accessible; (b) If a state s is accessible and an action a is possible at state s, then the successor state $do(a, s)$ that results from performing action a at state s is accessible; (c) No other states are accessible.

Notice that our primitive transactions can be more general than the usual deletions and insertions. All we need to consider a transaction as primitive is to introduce a name for it in the language and consider its effects on tables through its occurrence in the SSAs of the them.

We will usually denote by Σ the SC specification containing the elements mentioned above. Notice that we are not considering integrity constraints (ICs) as a part of the specification, but entailments of the specification ([16, 11]).

The *regression operator*, $\mathcal{R}$, [14, 16] applied to a formula evaluated at a successor state returns an equivalent formula (wrt the specfcation) evaluated at the current state. This is done by using the SSAs. More precisely, if fluent F, appearing in a formula ψ, has a SSA like 1, then the operator, $\mathcal{R}$, applied to ψ, replaces each occurrence of an atomic formula of the form $F(\bar{t}, do(a, s))$ by $\varphi_F(\bar{t}, a, s)$.

Finally, a dynamic query (or formula) in the SC is a formula that explicitly refers to several states of the database. Examples of this are dynamic integrity constraints, queries about the evolution of the database, etc.

Example 1. We consider a database of a company, with the following relations and primitive transactions:

[1]For distinct action names A and A', $A(\bar{x}) \neq A'(\bar{y})$. For each action name A, the axiom $A(x_1, \ldots, x_n) = A(y_1, \ldots, y_n) \supset x_1 = y_1 \wedge \cdots \wedge x_n = y_n$. $S_0 \neq do(a, s)$. $do(a, s) = do(a', s') \supset a = a' \wedge s = s'$.

Relations:

$Emp(x,s)$	:	Person x is an employee of the company when the database is in state s.
$Ceo(x,s)$	:	Person x is a Chief Executive Officer of the company, in the state s
$Salary(x,p,s)$	:	The salary of the person x is p in the state s.

Actions:

$hire(x)$	:	Person x is hired by the company.
$fire(x)$	:	Person x is fired by the company.
$quit(x)$	:	Person x quits to his (her) job.
$promote(x)$	:	Person x is promoted to Chief Executive Officer.
$changeSalary(x,p)$	:	The salary of the person x is changed to p dollars.

Initial Database: Our especification have the following formulas in the initial database:

$$\forall x (Emp(x, S_0) \equiv x = John) \tag{3}$$

$$Sue \neq John \tag{4}$$

Successor State Axioms: Our relations have the following SSAs.

$$\forall(a,s) Poss(a,s) \supset \quad \forall x[Emp(x, do(a,s)) \equiv a = hire(x) \vee Emp(x,s) \wedge a \neq fire(x) \wedge a \neq quit(x)]$$

$$\forall(a,s) Poss(a,s) \supset \quad \forall x[Ceo(x, do(a,s)) \equiv a = promote(x) \vee Ceo(x,s) \wedge a \neq fire(x) \wedge a \neq quit(x)]$$

$$\begin{aligned}\forall(a,s) Poss(a,s) \supset \quad & \forall(x,p)[Salary(x,p,do(a,s)) \equiv a = changeSalary(x,p) \vee Salary(x,p',s) \wedge \\ & \neg \exists p'(a = changeSalary(x,p') \wedge p \neq p')]\end{aligned}$$

Action Precondition Axioms: Now, assume that we have the following precondition axioms for the actions in the database.

$$\forall(x,s)[Poss(hire(x),s) \equiv \neg Emp(x,s)].$$

$$\forall(x,s)[Poss(fire(x),s) \equiv Emp(x,s)].$$

$$\forall(x,s)[Poss(quit(x),s) \equiv Emp(x,s)].$$

$$\forall(x,s)[Poss(promote(x),s) \equiv Emp(x,s) \wedge \neg Ceo(x,s)].$$

$$\forall(x,p,s)[Poss(changeSalary(x,p),s) \equiv Emp(x,s) \wedge \forall p'(Salary(x,p',s) \supset p' \leq p)].$$

In this example, is a dynamic query the following sentence:

$$\forall(s', s'')(S_0 \leq s' < s'' \supset \forall(x, p', p'')((salary(x,p',s') \wedge salary(x,p'',s'')) \supset p' \leq p'').$$

3 Past Temporal Logic and Chomicki's History Encoding

First–Order Past Temporal Logic [7, 8] is an extension of first–order logic by means of two new rules for building formulas: if φ, ψ are formulas, so are $\bullet\varphi$ and φ **since** ψ. Although there is no explicit time at the object level in this logic, the intuitive contents of the first formula, evaluated at a certain time, is that φ was true in the previous time. The second one means that formula ψ became true at some time in the past and from that time on, φ has been true.

More formally, formulas are interpreted in finite sequences of structures, D_i, indexed by discrete linear time:

(a) For $0 < i \leq n$, $D_0, \ldots, D_n \models_i \bullet\varphi$ iff $D_0, \ldots, D_n \models_{i-1} \varphi$
(b) For $0 < i \leq n$, $D_0, \ldots, D_n \models_i \varphi$ **since** ψ iff there is $j < i$ such that $D_0, \ldots, D_n \models_j \psi$ and for all $j < k \leq i$, $D_0, \ldots, D_n \models_k \varphi$.

Here, $\models_i$ means that the formula is being evaluated at time i. Non temporal first–order formulas are evaluated as usual.

Other temporal operators can be defined from the two previous ones: $\Diamond\varphi =_{def}$ *True* **since** φ (sometime in the past φ was true), $\Box\varphi =_{def} \neg\Diamond\neg\varphi$ (always in the past φ was true).

Notice that every model of the situation calculus restricted to a finite path in the situation calculus tree gives rise to a sequence of first–order structures, that can be used to interpret FOPTL formulas. In this case, the structures are indexed by states, S. In this case, we may write $\models_S$.

In [6], Chomicki uses FOPTL for formulating temporal integrity constraints. He addresses the problem of their checking; in the following sense. Let us assume that we have a temporal database $D = (D_0, \ldots, D_k)$ and a temporal IC φ. By executing an update we will generate a new database state D_{k+1}. Is it the case that $D = (D_0, \ldots, D_k, D_{k+1})$ satisfies φ?.

To answer this question we might consider all the information stored in the states $D_0, \ldots, D_{k+1}$, but this would be too costly in terms of space and time. Instead, Chomicki develops a methodolgy for storing in the state D_k the information that is necessary to evaluate φ in the state D_{k+1}. More precisely, this procedure generates a new temporal database $(\overline{D}_0, \ldots, \overline{D}_k)$, where each original state has been extended with the information that is needed to evaluate φ in the successor state. It contains, apart form the original base relations, new auxiliary relations generated as follows: for each subformula α, with temporal operators, of the given IC φ, a new table r_α is created. It has as many arguments as the number of free variables in α.

In addition, there will be a new relation r_φ representing the IC. This relation will be evaluated at each new state in order to check the IC, without having to query past database states. Since this table comes from a sentence, it will contain just one of the two possible entries $true$ or $false$. We will explain in section 5 how to update (and define througt this process) all these auxiliary tables. This will be done by providing appropriate SSA for them in the SC.

4 Situation Calculus vs FOPTL

In this section we will show that there exists a sort of equivalence between FOPTL and Situation Calculus for a particular kind of queries.

Consider a sequence of states S_0, ..., S_n originated by the execution of a sequence T of primitive transactions $A_1, \ldots, A_n$. There is a sequence of relational databases D_0, ..., D_n associated to them. We will see that for every dynamic query in the Situation Calculus about these states, there exists a query in FOPTL to be posed at the final state S_n (wrt to the sequence of structures D_0, ..., D_n) that has the same answer. Reciprocally, given a FOPTL formula to be evaluated at state S_n, we will see that there is a SC dynamic formula mentioning the sequence of states $S_0, \ldots, S_n$ that gives the same answer. If a SC formula $\varphi(s)$ and a FOPTL formula ψ satisfy this correspondence, we say that they formulas are *answer–equivalent* in S_n.

Example 2. The formulas $\exists s\ (S_0 \leq s < S_n \wedge \forall x P(x, s))$ and $\Diamond\forall x P(x)$ are answer–equivalent in

S_n.

□

Notice that the SC formulas we are interested in refer only to the states arising from the execution of transaction T. Next, we will characterize syntactically this kind of formulas.

Definition 1. (a) Given T as above, $\mathcal{L}_T$ is the class of SC formulas that satisfy:
1. If $\varphi(s)$ is a simple formula in s, then $\varphi(do(T', S_0))$ is in $\mathcal{L}_T$, for every T' of the form $[A_1, ..., A_k]$ $(0 \leq k \leq n)$ (with $k = 0$ we obtain state S_0).
2. If φ and ψ are in $\mathcal{L}_T$, and x is a variable symbol, not of the state sort, then $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \supset \psi$, $\varphi \equiv \psi$, $\exists x \varphi$ and $\forall x \varphi$ are in $\mathcal{L}_T$.
3. If $\varphi(s)$ is in $\mathcal{L}_T$ and s_L, s_U are term of sort state, then $\exists s(s_L \leq s \leq s_U \wedge \varphi(s))$ and $\forall s(s_L \leq s \leq s_U \supset \varphi(s))$ are in $\mathcal{L}_T$ (the symbol $\leq$ can be replaced by $<$).

(b) A formula φ is T–bounded if it is in $\mathcal{L}_T$ and for every state term s (quantified or not) that appear in this formula, it is possible to deduce that $S_0 \leq s \leq do(T, S_0)$ from φ and the foundational axioms of the SC, in particular, from the axioms that define $\leq$ as a partial order and the unique name axioms for states.

Example 3. Let us consider the sentence in $\mathcal{L}_T$:

$$\exists s(S_0 \leq s < do(T, S_0) \wedge P(s) \wedge \forall s'(S_0 \leq s' < s \supset Q(s) \wedge \forall s''(s' < s'' < s \supset R(s)))). \tag{5}$$

In this formula the state variables s, s' and s'' appear. Using the relations $<$ and $\leq$ in this formula, we obtain $S_0 \leq s' < s'' < s < do(T, S_0)$. Finally, using the axiom that defines $\leq$ as a transitive relation, we conclude that (5) is T–bounded.

□

Example 4. Now, let us consider the formula in $\mathcal{L}_T$:

$$\exists s(S_0 \leq s \leq s' \wedge P(s)) \tag{6}$$

In this case the state variables s and s' appear. Is not possible to conclude that $S_0 \leq s' \leq do(T, S_0)$, because we do not have any upper bound for this state. Therefore, (6) is not T–bounded.

□

By an easy adaptation of an example in [1], we know that not every SC formula can be translated into an equivalent FOPTL formula. Nevertheless, this translation is possible if we restrict ourselves to T–bounded formulas. In fact, we can do this translation using $\bullet$ as the only temporal connective. For doing this, we first need to replace every universal state quantification of the form $\forall s\ (s_L \leq s \leq s_U \supset \varphi(s))$, by a conjunction of the form $\varphi(s_L) \wedge \cdots \wedge \varphi(s_U)$, and every existential state quantification of the form $\exists s\ (s_L \leq s \leq s_U \wedge \varphi(s))$, by a disjunction of the form $\varphi(s_L) \vee \cdots \vee \varphi(s_U)$. Finally, it is necessary to replace every formula of the form $\varphi(do([A_1, ..., A_k], S_0))$ $(k \leq n)$ by a FOPTL term of the form $\bullet^{n-k}\varphi$ [2]. For example, let us consider $T = [A_1, A_2]$ and the sentence:

$$\exists s\ (S_0 \leq s \leq do(T, S_0) \wedge P(x, s) \wedge \forall s'(S_0 \leq s' < s \supset Q(y, s'))).$$

In this case, when we eliminate the quantifiers, we obtain:

[2] $\bullet^k\varphi$ is an abbreviation for $\underbrace{\bullet \cdots \bullet}_{k \text{ times}}\varphi$.

$$P(x, S_0) \ \vee \ P(x, do(A_1, S_0)) \wedge Q(y, S_0) \vee$$
$$P(x, do([A_1, A_2], S_0)) \wedge Q(y, S_0) \wedge Q(y, do(A_1, S_0)),$$

and finally, replacing by the connective $\bullet$, we obtain:

$$\bullet^2 P(x) \ \vee \ \bullet P(x) \wedge \bullet^2 Q(y) \ \vee \ P(x) \wedge \bullet^2 Q(y) \wedge \bullet Q(y). \tag{7}$$

We can see that the temporal formula resulting from this translation does not use all the expressive power of FOPTL. In addition, it may contain too many temporal operators, what can make too expensive the process of answering the new query. For this reason, is not desirable to use formulas of the form (7), but more natural translations, maybe using other temporal operators.

In the rest of this section we will show an operator that translates every formula written in FOPTL, to be evaluated at a given state s, into a SC formula. Next, we will use that operator for justifying the translations of some SC formulas into temporal formulas that contain connectives other from $\bullet$.

The translation of FOPTL formulas into SC formulas is done with the compilation operator $\mathcal{C}(\varphi, s)$ defined below, that receives in its first argument the formula and in its second argument, the state where the temporal formula is evaluated. We are assuming that the linearly ordered states underlying the temporal formula are the database states corresponding to the SC states $S_0, S_1, \ldots, do(T, S_0)$, that are also linearly ordered by the accessibility relation $\leq$.

This is definition of the compilation operator:

$$\begin{array}{rcl}
\mathcal{C}(t_1 * t_2, s) & =_{def} & t_1 * t_2, \ * \in \{<, >, \leq, \geq, =, \neq\}. \\
\mathcal{C}(P(\bar{x}), s) & =_{def} & P(\bar{x}, s), \ \ \textit{if } P \textit{ is a predicate symbol} \\
\mathcal{C}(\neg\varphi, s) & =_{def} & \neg\mathcal{C}(\varphi, s) \\
\mathcal{C}(\varphi \wedge \psi, s) & =_{def} & \mathcal{C}(\varphi, s) \wedge \mathcal{C}(\psi, s) \\
\mathcal{C}(\varphi \vee \psi, s) & =_{def} & \mathcal{C}(\varphi, s) \vee \mathcal{C}(\psi, s) \\
\mathcal{C}(\varphi \supset \psi, s) & =_{def} & \mathcal{C}(\varphi, s) \supset \mathcal{C}(\psi, s) \\
\mathcal{C}(\varphi \equiv \psi, s) & =_{def} & \mathcal{C}(\varphi, s) \equiv \mathcal{C}(\psi, s) \\
\mathcal{C}(\exists x \varphi, s) & =_{def} & \exists x \mathcal{C}(\varphi, s) \\
\mathcal{C}(\forall x \varphi, s) & =_{def} & \forall x \mathcal{C}(\varphi, s) \\
\mathcal{C}(\bullet\varphi, s) & =_{def} & \exists(a, s') \ (s = do(a, s') \wedge \mathcal{C}(\varphi, s')) \\
\mathcal{C}(\varphi \textbf{ since } \psi, s) & =_{def} & \exists s' (S_0 \leq s' < s \wedge \mathcal{C}(\psi, s') \wedge \forall s'' (s' < s'' \leq s \supset \mathcal{C}(\varphi, s''))) \\
\mathcal{C}(\Diamond\varphi, s) & =_{def} & \exists s' (S_0 \leq s' < s \wedge \mathcal{C}(\varphi, s')) \\
\mathcal{C}(\Box\varphi, s) & =_{def} & \forall s' (S_0 \leq s' < s \supset \mathcal{C}(\varphi, s'))
\end{array}$$

The last two cases in the definition can be eliminated, because they can be obtained from the previous cases using the definitions: $\Diamond\varphi \equiv \textit{True} \textbf{ since } \varphi$ and $\Box\varphi \equiv \neg(\textit{True} \textbf{ since } \neg\varphi)$.

It is easy to see that by applying the operator $\mathcal{C}$ to an arbitrary FOPTL formula, with $do(T, S_0)$ in the second argument, we always obtain a T–bounded formula.

Now, consider some examples of translations. The SC sentences are dynamic queries taken from [18].

Example 5. "Sue has always been working in the company, in all states generated by the execution of the sequence of actions T at S_0?". This dynamic query can be expressed by the formula:

$$\forall s \ (S_0 \leq s \leq do(T, S_0) \supset emp(Sue, s)),$$

which is equivalent to the formula:

$$emp(Sue, do(T, S_0)) \land \forall s \ (S_0 \leq s < do(T, S_0) \supset emp(Sue, s)). \tag{8}$$

If we apply the operator $\mathcal{C}$, at the state $do(T, S_0)$, to the sentence:

$$emp(Sue) \land \Box emp(Sue)$$

we obtain (8), that, in particular, turns out to be a T–bounded formula.

Example 6. "Is there a state such that someone is enrolled in the courses c100 and m100?". This dynamic query can be expressed by the formula:

$$\exists s \ (S_0 \leq s \leq do(T, S_0) \land \exists st \ (enrolled(st, c100, s) \land enrolled(st, m100, s))),$$

that is equivalent to the sentence:

$$\begin{gathered} \exists st \ (enrolled(st, c100, do(T, S_0)) \land enrolled(st, m100, do(T, S_0))) \lor \\ \exists s \ (S_0 \leq s < do(T, S_0) \land \exists st \ (enrolled(st, c100, s) \land enrolled(st, m100, s))). \end{gathered}$$

This formula can be obtained by applying the operator $\mathcal{C}$, at the state $do(T, S_0)$, to the formula

$$\exists st \ (enrolled(st, c100) \land enrolled(st, m100)) \lor \Diamond \exists st \ (enrolled(st, c100) \land enrolled(st, m100)).$$

Example 7. "Was Joe hired as an employee of a lower rank before becoming a Chief Executive Officer (CEO)?". This dynamic query can be expressed by the formula:

$$\begin{gathered} \exists s \ (S_0 \leq s \leq do(T, S_0) \land \\ \forall s' \ (S_0 \leq s' < s \supset emp(Joe, s') \land \neg ceo(Joe, s')) \land \\ \forall s'' \ (s \leq s'' \leq do(T, S_0) \supset emp(Joe, s'') \land ceo(Joe, s''))), \end{gathered}$$

which is equivalent to the sentence:

$$\begin{gathered} \forall s' \ (S_0 \leq s' < do(T, S_0) \supset emp(Joe, s') \land \neg ceo(Joe, s')) \land \\ emp(Joe, do(T, S_0)) \land ceo(Joe, do(T, S_0)) \\ \bigvee \\ \exists s \ (S_0 \leq s < do(T, S_0) \land emp(Joe, s) \land ceo(Joe, s) \land \\ \forall s' \ (S_0 \leq s' < s \supset emp(Joe, s') \land \neg ceo(Joe, s')) \land \\ \forall s'' \ (s < s'' \leq do(T, S_0) \supset emp(Joe, s'') \land ceo(Joe, s''))). \end{gathered}$$

This formula can be obtained by applying the operator $\mathcal{C}$, at the state $do(T, S_0)$, to the formula

$$\begin{gathered} \Box(emp(Joe) \land \neg ceo(Joe)) \land emp(Joe) \land ceo(Joe) \\ \bigvee \\ [emp(Joe) \land ceo(Joe)] \ \textbf{since} \ [emp(Joe) \land ceo(Joe) \land \Box(emp(Joe) \land \neg ceo(Joe))] \end{gathered}$$

Example 8. "Is there anyone that has always been working in the company, in all states generated by the execution of the sequence of actions T at S_0?". This dynamic query can be expressed by the formula:

$$\exists x \forall s \ (S_0 \leq s \leq do(T, S_0) \supset emp(x, s)),$$

which is equivalent to the formula:

$$\exists x(emp(x, do(T, S_0)) \wedge \forall s\ (S_0 \leq s < do(T, S_0) \supset emp(x, s)). \tag{9}$$

If we apply the operator $\mathcal{C}$, at the state $do(T, S_0)$, to the sentence $\exists x(emp(x) \wedge \Box emp(x))$, we obtain the previous formula.

We can see from the examples that there are FOPTL formulas corresponding to $\mathcal{L}_T$ formulas that are much nicer and natural than the ones obtained by means of the operators $\bullet^k$. We will not give a more general and natural translation mechanism from SC formulas into FOPTL formulas, because the natural translation can be guessed (and possibly checked with operator $\mathcal{C}$). In section 6, we will show how to use the correspondence between SC dynamic queries and FOPTL queries for answering the former. We may also think of using FOPTL as the original, direct, language for posing the dynamic queries, assuming that the underlying temporal states correspond to the intended SC states.

5 Modifying Specifications from Dynamic Queries

The methodology developed by Chomicki can be used, with adaptations, in the context of situation calculus based specifications of database updates. In fact, we will generate a mechanism that receives as input a SC specification Σ and a FOPTL sentence $\varphi(\bar{x})$, to be evaluated at the final state S resulting from the virtual update of the database at the initial state, S_0 (say, $S = S_n = do(T, S_0)$). Notice that this formula $\varphi(\bar{x})$ implicitly refers to all states between S_0 and S. Our mechanism generates a new SC specification, Σ_H, that extends Σ, and a SC formula $\varphi^h(\bar{x}, S)$, such that the answer to the original query, is the same as the answer obtained by evaluating $\varphi^h(\bar{x}, S)$ wrt Σ_H. The new formula refers only to the state S, and Σ_H contains a specification of the dynamics of new, history encoding, auxiliary relations.

With the new specification we will be in position to reason hypothetically along virtual database trajectories. We will find other applications in sections 7, 8 and 9.

Now we will describe the procedures for generating, first, φ^h and, second, Σ_H. They are based on Chomicki's methodology, in the sense that we include new relations names for each temporal subformula of sentence $\varphi(\bar{x})$, and we specify their dynamics in terms of SSAs taking into account the way the new relations are defined.

Let us consider a FOPTL formula φ. We will define the formula $\varphi^h(s)$ by induction:

1. If φ is an atomic formula of the form $F(\bar{t})$ (i.e. a table name instantiated with terms), then $\varphi^h =_{def} F(\bar{t}, s)$ (here, we add an extra state argument, obtaining a SC atomic formula).
2. If φ is $\neg\psi$, then $\varphi^h =_{def} \neg(\psi^h)$.
3. $(\varphi * \psi)^h =_{def} (\varphi^h * \psi^h)$, where $*$ is any of the usual binary propositional connectives.
4. $(Q\varphi)^h =_{def} Q(\varphi^h)$, where Q is any of the usual first–order quantifiers.
5. If φ is $\bullet\psi(\bar{x})$, with ψ non temporal, then $\varphi^h =_{def} R_\varphi(\bar{x}, s)$, where R_φ is a new table name.
6. If $\varphi(\bar{x})$ is $\psi(\bar{x})$ **since** $\theta(\bar{x})$, where $\psi(\bar{x})$ and $\theta(\bar{x})$ are non temporal formulas, then $\varphi^h =_{def} R_\varphi(\bar{x}, s)$, where R_φ is a new table name.

Notice that by bottom–up transformation of a FOPTL formula φ, we can always obtain φ^h applying the steps above. We can see that every temporal subformula of φ gives rise to a new table name. Notice also that formula φ^h is a SC formula that is simple in s.

Now we will try to specify the dynamics of the new tables by means of suitable SSAs for them and definitions for these new tables in the initial state. We need to consider the last two cases in the previous definition:

(a) Let $\alpha(\bar{x})$ be a temporal formula of the form $\bullet\psi(\bar{x})$, where $\psi(\bar{x})$ is a non temporal formula. This formula is true at a given state if $\psi(\bar{x})$ is true at the previous state. Then, the new table $R_\alpha(\bar{x}, s)$ has the following SSA:

$$\forall(a,s)Poss(a,s) \supset \forall\bar{x}\ (R_\alpha(\bar{x}, do(a,s)) \equiv \psi^h(\bar{x}, s)). \tag{10}$$

Although, in the initial state $\alpha(\bar{x})$ is false for each $\bar{x}$, because there is not a previous state to S_0, so we define $R_\alpha(\bar{x}, S_0)$ as follows:

$$\forall\bar{x}\ \neg R_\alpha(\bar{x}, S_0). \tag{11}$$

(b) Let $\alpha(\bar{x})$ be a formula of the form $\psi(\bar{x})$ **since** $\theta(\bar{x})$, where $\psi(\bar{x})$ and $\theta(\bar{x})$ are non temporal formulas. This formula is true at a state s, with predecessor state s', iff (ψ **since** θ) was true at s' and ψ is still true at s or ψ became true at s and θ became true at s'. This is equivalent to saying that ((ψ **since** θ) $\vee\ \theta$) is true at s' and ψ is true at s. Then, for the new table $R_\alpha(\bar{x}, s)$ it holds:

$$\forall(a,s)Poss(a,s) \supset \forall\bar{x}\ (R_\alpha(\bar{x}, do(a,s)) \equiv \psi^h(\bar{x}, do(a,s)) \wedge (R_\alpha(\bar{x}, s) \vee \theta^h(\bar{x}, s)). \tag{12}$$

This is not a SSA, because there is a $do(a,s)$ term in ψ^h on the RHS, but we can get rid of it applying Reiter's regression operator $\mathcal{R}$, that takes a formula, instantiated at a successor state of the form $do(A,s)$, to a formula instantiated at the previous state, s. For doing this, it uses the SSAs of the tables appearing in the formula $\psi^h(\bar{x}, do(a,s))$ [14, 16]. We obtain:

$$\forall(a,s)Poss(a,s) \supset \forall\bar{x}\ (R_\alpha(\bar{x}, do(a,s)) \equiv \mathcal{R}[\psi^h(\bar{x}, do(a,s))] \wedge (R_\alpha(\bar{x}, s) \vee \theta^h(\bar{x}, s))). \tag{13}$$

Finally, because S_0 has no predecessor state, we define $R_\alpha(\bar{x}, S_0)$ as follows:

$$\forall\bar{x}\ \neg R_\alpha(\bar{x}, S_0). \tag{14}$$

The application of the regression operator leaves the RHS of (13) as a simple formula in s. When α is a temporal sentence, the SC formula $R_\alpha(s)$ becomes a dynamic propositional predicate.

Notice that with these SSAs for the history encoding relations we are in position to reason about the update of the new relations by any primitive transaction of the original specification. In particular, the transactions may be the usual deletions and insertions, but we can have more general primitive transactions. Apart from this generality, in opposition to Chomicki's work, we can reason hypothetically with the new tables.

Now we will illustrate the transformation methodology with some examples.

Example 9. Let $\beta(x)$ the temporal formula $\Diamond P(x)$. We know that $\Diamond P(x) \equiv$ *True* **since** $P(x)$. Then, we obtain the following SSA for the new table $R_\beta(x, s)$:

$$\forall(a,s)Poss(a,s) \supset \forall x\ (R_\beta(x, do(a,s)) \equiv \mathcal{R}[True] \wedge (R_\beta(x,s) \vee P(x,s))).$$

Since, independently of the state at we evaluate *True*, we obtain truth, this SSA is equivalent to:

$$\forall(a,s)Poss(a,s) \supset \forall x\ (R_\beta(x, do(a,s)) \equiv R_\beta(x,s) \vee P(x,s)).$$

By (14), for R_β in the initial state we have $\forall x\ \neg R_\beta(x, S_0)$.

In general, if γ and δ are the formulas $\Diamond\psi(\bar{x})$ and $\Box\psi(\bar{x})$, where $\psi(\bar{x})$ is a non temporal formula, respectively, then the SSAs for the relations R_γ and R_δ, arising from them, are:

$$\forall(a,s)Poss(a,s) \supset \forall\bar{x}\ (R_\gamma(\bar{x}, do(a,s)) \equiv R_\gamma(\bar{x}, s) \vee \psi^h(\bar{x}, s)) \tag{15}$$

$$\forall(a,s)Poss(a,s) \supset \forall\bar{x}\ (R_\delta(\bar{x}, do(a,s)) \equiv R_\delta(\bar{x}, s) \wedge \psi^h(\bar{x}, s)). \tag{16}$$

Now, taking into account that $\Box\psi(\bar{x}) = \neg\Diamond\neg\psi(\bar{x})$, we have the following definitions for the new predicates in the initial state: $\forall\bar{x}\ \neg R_\gamma(\bar{x}, S_0)$, $\forall\bar{x}\ R_\delta(\bar{x}, S_0)$.

Example 10. Let us consider a specification with contains the following SSA for the fluent $P(x,s)$:

$$Poss(a,s) \supset \forall x(P(x, do(a,s)) \equiv a = A(x) \vee P(x,s) \wedge a \neq B(x)). \tag{17}$$

We want to evaluate the query

$$P(x)\ \textbf{since}\ (\Diamond Q(x)). \tag{18}$$

In this case, if we denote with β the temporal subformula $\Diamond Q(x)$,then we introduce a new table R_β, with SSA:

$$\forall(a,s)Poss(a,s) \supset \forall x\ (R_\beta(x, do(a,s)) \equiv R_\beta(x,s) \vee Q(x,s)).$$

Introducing R_β in (18) we obtain the formula $P(x)$ **since** $R_\beta(x)$. If we denote this formula with $\alpha(x)$, for the new table R_α we obtain the SSA:

$$\forall(a,s)Poss(a,s) \supset \forall x\ (R_\alpha(x, do(a,s)) \equiv \mathcal{R}[P(x, do(a,s))] \wedge (R_\alpha(x,s) \vee R_\beta(x,s))).$$

Now, replacing $\mathcal{R}[P(x, do(a,s))]$ by the RHS of (17), we finally obtain the SSA:

$$\forall(a,s)Poss(a,s) \supset \forall x\ (R_\alpha(x, do(a,s)) \equiv (a = A(x) \vee P(x,s) \wedge a \neq B(x)) \wedge (R_\alpha(x,s) \vee R_\beta(x,s))).$$

Finally, by (b) and the previous example, we have the following definition for the new predicates in the initial state: $\forall x \neg R_\alpha(x, S_0)$, $\forall x \neg R_\beta(x, S_0)$.

Example 11. Let us consider the query:

$$(\Diamond P(x))\ \textbf{since}\ (\Box Q(x)).$$

If we denote with β and γ, the temporal subformulas $\Diamond P(x)$, $\Box Q(x)$, resp., then for the new tables R_β and R_γ we have the following SSAs:

$$\forall(a,s)Poss(a,s) \supset \forall x\ (R_\beta(x, do(a,s)) \equiv R_\beta(x,s) \vee P(x,s)) \tag{19}$$

$$\forall(a,s)Poss(a,s) \supset \forall x\ (R_\gamma(x, do(a,s)) \equiv R_\gamma(x,s) \wedge Q(x,s)). \tag{20}$$

Plugging in the tables R_β and R_γ into the original query, we obtain $R_\beta(x)$ **since** $R_\gamma(x)$. We denote this new formula by $\alpha(x)$. The SSA of the new fluent R_α is:

$$\forall(a,s)Poss(a,s) \supset \forall x\ (R_\alpha(x, do(a,s)) \equiv \mathcal{R}[R_\beta(x, do(a,s))] \wedge (R_\alpha(x,s) \vee R_\gamma(x,s))).$$

Replacing $\mathcal{R}[R_\beta(x, do(a,s))]$ by the RHS of (19), we obtain:

$$\forall(a,s)Poss(a,s) \supset \forall x\ (R_\alpha(x, do(a,s)) \equiv (R_\beta(x,s) \vee P(x,s)) \wedge (R_\alpha(x,s) \vee R_\gamma(x,s))).$$

Finally, by (14) and the previous example, we have the following definition for the new predicates in the initial state: $\forall x \neg R_\alpha(x, S_0)$, $\forall x \neg R_\beta(x, S_0)$, and $\forall x R_\gamma(x, S_0)$.

6 Answering Hypothetical Dynamic Queries

Let us assume that we have an specification of the dynamics of a database as in section 2. We also have a sequence T of legal primitive transactions $A_1, \ldots, A_n$, whose execution generates states $S_1, \ldots, S_n$ from S_0, that is, $S_n = do(T, S_0)$. Without necessarily realizing a physical update of the database, we want to ask a T–bounded dynamic query φ, or its version in FOPTL, say ψ to this trajectory of the database. For simplicity, let us think that these are sentences that query the whole evolution of the database from S_0 to S_n. So, we are asking whether:

$$\Sigma \models \varphi \tag{21}$$

or

$$D_0, \ldots, D_n \models_{S_n} \psi. \tag{22}$$

In (21) we have a SC dynamic query posed against the virtually updated database. Notice that we are evaluating against the specification of the dynamics plus the initial database. The newly generated states appear in the query. In (22), $D_0, \ldots, D_n$ is the sequence of (possibly virtually) generated databases. Of course, the FOPTL sentence ψ has no explicit states terms, in particular, in it, the SC dynamic predicates lost their state argument. This sentence is evaluated at state S_n, and this is what the $\models_{S_n}$ indicates.

We may decide to use FOPTL for asking temporal queries. Actually, for simplicity of the presentation, we will assume that this is the case. By the results of the previous section, we have that asking (22) is equivalent to asking

$$\Sigma_H \models \psi^h(do(T, S_0)). \tag{23}$$

Here, Σ_H contains a specification of the auxiliary history encoding relations, and $\psi^h(do(T, S_0))$ is the SC formula $\psi^h(s)$ generated from ψ (as in section 5), but instantiated at the final state $do(T, S_0)$, and this is the only state mentioned in the formula.

We have seen that we have transformed our problem of answering a temporal (dynamic) query wrt to a virtually updated database into the *temporal projection problem*. In artificial intelligence planning, this is the problem of querying a state of the world that is reached by execution of a sequence of legal actions [9].

Example 12. Let us consider the specification Σ given in section 4. Suppose that in this database we want to know if is there anyone that has always been working in the company, in all states generated by the execution of the sequences of actions $T = [hire(Sue), fire(John)]$ at S_0, that is, we want to know if the following is true:

$$\Sigma \models \exists x \forall s (S_0 \leq s \leq do(T, S_0) \supset Emp(x, s)). \tag{24}$$

By example 8 we know that this setence is answer-equivalent in $do(T, S_0)$ with the FOPTL formula

$$\exists x (Emp(x) \wedge \Box Emp(x)).$$

If we apply the methodology, showed in section 5, to $\Box Emp(x)$, we obtain

$$\exists x (Emp(x) \wedge \Box Emp(x))^h = \exists x (Emp(x, s) \wedge R_\alpha(x, s)), \tag{25}$$

where $R_\alpha(x, s)$ is a new table, that replace $\Box Emp(x)$, with the following definition:

$$\forall x \; R_\alpha(x, S_0). \tag{26}$$

$$\forall (a, s) Poss(a, s) \supset \forall x (R_\alpha(x, do(a, s)) \equiv Emp(x, s) \wedge R_\alpha(x, s)). \tag{27}$$

Thus $\Sigma_H = \Sigma \cup \{(26), (27)\}$, and (24) holds if and only if the following is true:

$$\Sigma_H \models \exists x(Emp(x, do(T, S_0)) \wedge R_\alpha(x, do(T, S_0))). \tag{28}$$

For solving this temporal projection problem in the context of situation calculus based specifications of database updates, we may think of applying some established techniques:

1. Apply the regression operator to $\psi^h(do(T, S_0))$, that is, solve the problem of determining whether

$$D_0, UNA \models \mathcal{R}(\psi^h(do(T, S_0))), \tag{29}$$

obtaining evaluation wrt to the initial database (plus unique names axioms for actions, and possibly, individuals). This is the approach presented in [14, 16]. With this technique we do not need to physically update the database.

2. Roll forward the database according to Σ_H from S_0 to S_n, obtaining the new initial state S_0' and the new initial database D_0', and evaluate the query

$$D_0' \models \psi^h(S_0'), \tag{30}$$

as done in [12]. In this case we are physically updating the database, and we are essentially n Chomicki's case.

3. Apply the methodology presented [19, 18, 20] for answering dynamic queries using relevance, that solves the problem by performing a minimal update of the database. This is achieved by determining the primitive transactions and database tuples that are relevant to the query. Only relevant transactions are executed and relevant tuples are updated accordingly. Actually, we do not need the whole methodology, because it was designed for answering general dynamic queries in a canonical query format, whereas here we have a query posed only to the final state. This case can be handled by more specific algorithms presented in [18], which are derived from the general methodology for this particular problem. This approach let us still work hypothetically with a minimal physical update of (possibly auxiliary) tables.

Example 13. Let us consider the specification Σ in example 25. For answering (24) we need to evaluate if the new specification Σ_H, generated in this example, satisfies

$$\Sigma_H \models \exists x(Emp(x, do(T, S_0)) \wedge R_\alpha(x, do(T, S_0))). \tag{31}$$

Applying once the Regression Operator to the RHS, we obtain:

$$\begin{gathered}\exists x(\mathit{fire}(John) = hire(x) \vee Emp(x, do(hire(Sue), S_0)) \wedge \mathit{fire}(John) \neq \mathit{fire}(x) \wedge \mathit{fire}(John) \neq quit(x) \\ \bigwedge \\ Emp(x, do(hire(Sue), S_0)) \wedge R_\alpha(x, do(hire(Sue), S_0)))\end{gathered}$$

Is possible to simplify this sentence using Unique Name Axioms for Actions, obtaining:

$$\exists x(Emp(x, do(hire(Sue), S_0)) \wedge x \neq John \wedge R_\alpha(x, do(hire(Sue), S_0)))$$

Again we apply the Regression Operator, obtaining:

$$\exists x((\mathit{hire}(\mathit{Sue}) = \mathit{hire}(x) \vee \mathit{Emp}(x, S_0) \wedge \mathit{hire}(\mathit{Sue}) \neq \mathit{fire}(x) \wedge \mathit{hire}(\mathit{Sue}) \neq \mathit{quit}(x))$$
$$\bigwedge$$
$$x \neq \mathit{John} \wedge \mathit{Emp}(x, S_0) \wedge R_\alpha(x, S_0))$$

Using Unique Name Axiom for Action, we generate:

$$\exists x((x = \mathit{Sue} \vee \mathit{Emp}(x, S_0)) \wedge x \neq \mathit{John} \wedge \mathit{Emp}(x, S_0) \wedge R_\alpha(x, S_0)) \tag{32}$$

Finally, (31) is true iff $\{(3), (4), (26)\} \models (32)$. Thus, for verifying (24) we need to answer a query about the extended initial database.

7 Non Markovian Transactions

As we have seen in section 2, Reiter's formalism contains action precondition axioms (APAs), which are of the form

$$\forall(\bar{x}, s)\ \mathit{Poss}(A(\bar{x}), s) \equiv \Pi_A(\bar{x}, s), \tag{33}$$

where $\Pi_A(\bar{x}, s)$ is a formula simple in s. That is, the conditions for the execution of a primitive transaction depend on the current state of the database, only. We call these actions "Markovian". Many concepts and algorithms that have arised from his formalism is based on this kind of action precondition axioms and assumption. Nevertheless, there are natural scenarios in which the conditions for executing an action depend on a longer history of the database. These are "non Markovian" transactions. In this section we show that is possible to use the methodology presented in section 5 for transforming history dependent actions into Markovian actions.

Example 14. We consider a voters database with the following relations and primitive transactions:

Relations:

$\mathit{Age}(x, n, s)$	:	The age of person x is n when the database is in state s
$\mathit{Candidate}(x, s)$	:	Person x is a candidate for a public position in the state s
$\mathit{InJail}(x, s)$	:	Person x is in jail in the state s

Actions:

$\mathit{Vote}(x, y)$	:	Person x votes for candidate y
$\mathit{BreakLaw}(x)$	:	Person x breaks the law
$\mathit{Release}(x)$	:	Person x finishes his (her) reclusion in jail

Successor State Axioms Our relations have SSAs, for example, for the table *InJail* we have:

$$\forall(a, s)\mathit{Poss}(a, s) \supset \mathit{InJail}(x, do(a, s)) \equiv a = \mathit{BreakLaw}(x) \vee \mathit{InJail}(x, s) \wedge a \neq \mathit{Release}(x). \tag{34}$$

Now, assume that we have the following preconditon axiom for action *Vote*:

$$\forall(x, y, s)\ (\mathit{Poss}(\mathit{Vote}(x, y), s) \quad \equiv \quad \exists n(\mathit{Age}(x, n, s) \wedge n \geq 18) \wedge \mathit{Candidate}(y, s)$$
$$\bigwedge$$
$$\forall s'(S_0 \leq s' \leq s \supset \neg \mathit{InJail}(x, s'))).$$

We can see that the precondition formula is not simple in s. We will tranform this preconditon axiom into one of the form (33). For motivation, we will do it in an ad hoc manner. Later on, we will apply a general methodology.

We introduce a new relation, *NotInJail*, such that , when instantiated in state s, it becomes true if *InJail* has not been true for all state s' such that $S_0 \leq s' \leq s$. This behaviour can be captured with the following axioms:

$$\forall x \ (NotInJail(x, S_0) \equiv \neg InJail(x, S_0)) \tag{35}$$

$$\forall(a, s) Poss(a, s) \supset \forall x \ (NotInJail(x, do(a, s)) \equiv NotInJail(x, s) \wedge a \neq BreakLaw(x)). \tag{36}$$

With this new fluent, the action precondition axiom for *Vote* can be changed into:

$$\forall(x, y, s) \ (Poss(Vote(x, y), s) \quad \equiv \quad \exists n(Age(x, n, s) \wedge n \geq 18) \wedge Candidate(y, s) \bigwedge NotInJail(x, s)).$$

Now, the precondition formula is simple in s. It will be checked with respect to the original specification together with formulas (35) and (36).

We will generalize the methodology in the previous example. Let A be actions in our database Σ, whose action precondition axiom is:

$$\forall(\bar{x}, s) \ Poss(A(\bar{x}), s) \equiv \varphi(\bar{x}, s) \tag{37}$$

where $\varphi(\bar{x}, s)$ is a formula that contains as subformula the expression:

$$\forall s'(S_0 \leq s' \leq s \supset \psi(\bar{x}, s)). \tag{38}$$

We want to replace (38), in (37), by a simple formula in s. For doing this, we have to assume that there exists a SSA for $\psi(\bar{x}, s)$ of the form [2]:

$$\forall(a, s) Poss(a, s) \supset \forall \bar{x} \ (\psi(\bar{x}, do(a, s)) \equiv \gamma_\psi^+(\bar{x}, a, s) \vee \psi(\bar{x}, s) \wedge \neg\gamma_\psi^-(\bar{x}, a, s)).$$

We define then a new fluents R as follows:

$$\forall \bar{x} \ R(\bar{x}, S_0) \equiv \psi(\bar{x}, S_0). \tag{39}$$

$$\forall(a, s) Poss(a, s) \supset \forall \bar{x} \ (R(\bar{x}, do(a, s)) \equiv R(\bar{x}, s) \wedge \neg\gamma_\psi^-(\bar{x}, a, s)). \tag{40}$$

Replacing (38) by $R(\bar{x}, s)$ in (37), we obtain a new precondition for A, of the form:

$$\forall(\bar{x}, s) \ Poss(A(\bar{x}), s) \equiv \theta(\bar{x}, s). \tag{41}$$

We have that (37) and (41) satisfy the following relation:

$$\Sigma \models Poss(A(\bar{t}), S) \quad \text{iff} \quad [\Sigma - (37)] \cup (39), (40) \models Poss(A(\bar{t}), S),$$

for any ground terms $\bar{t}, S$.

¿From this methodology, it is possible to obtain a general mechanism for replacing formulas of the form $\exists s' \ (S_0 \leq s' \leq s \wedge \psi(\bar{x}, s))$ in the precondition axioms, by formulas simple in s.

Let us now consider the general case. Assume that we have a specification Σ and a primitive transaction A with the precondition axiom:

$$\forall(\bar{x}, s) \ Poss(A(\bar{x}), s) \equiv \varphi(\bar{x}, s), \tag{42}$$

where $\varphi(\bar{x}, s)$ is a s-bounded formula that is not necessarily simple in s and has an answer–equivalent FOPTL formula $\psi(\bar{x})$ in s. Then, A is not necessarily a Markovian transaction.

We generate a new specification Σ'_H, which is the new specification Σ_H generated by considering the temporal formula $\psi(\bar{x})$ as in section 6, but with axiom (42) replaced by the new preconditon axiom

$$\forall(\bar{x}, s)\ Poss(A(\bar{x}), s) \equiv \psi^h(\bar{x}, s).$$

This specification includes SSAs for the new auxiliary tables, generated from $\psi^h(\bar{x}, s)$, and is equivalent to the initial specification in the sense that:

$$\Sigma \models Poss(A(\bar{t}), S) \quad \textit{iff} \quad \Sigma'_H \models Poss(A(\bar{t}), S),$$

for every ground terms $\bar{t}, S$.

Example 15. Let us apply the methodology presented above to the previous example. Considering that $\forall s'(S_0 \leq s' < s \supset \neg InJail(x, s'))$ is answer–equivalent to $\Box\neg Jail(x)$ in s, we obtain an specification Σ'_H that contains a new predicate R_α, with the following definition:

$$\forall x\ R_\alpha(x, S_0), \tag{43}$$

$$\forall(a, s) Poss(a, s) \supset \forall x\ (R_\alpha(x, do(a, s)) \equiv R_\alpha(x, s) \wedge \neg InJail(x, s)). \tag{44}$$

The new precondition axiom for $Vote$ is:

$$\forall(x, y, s)\ (Poss(Vote(x, y), s) \quad \equiv \quad \exists n(Age(x, n, s) \wedge n \geq 18) \wedge Candidate(y, s) \\ \bigwedge \\ \neg InJail(x, s) \wedge R_\alpha(x, s))$$

We can see that this precondition axiom is logically equivalent (wrt to the expanded specification) to the precondition axiom obtained in the previous example. This is because $NotInJail(x, s)$, in that example, says that x was never in jail until state s, inclusively. This is the same the formula $\neg InJail(x, s) \wedge R_\alpha(x, s))$ says in this example, where $R_\alpha(x, s)$ speaks for all states less than s.

Example 16. Suppose that we have an educational database with a relation, $Fails(x, c, s)$, meaning that the student x fails the course c in the state s and an action $Eliminate(x)$, meaning that the student x is eliminated from the university. Additionally, we assume that one of the conditions for eliminating a student from the university is failing three times the same course. The precondition axiom for the action $Eliminate$ is:

$$\forall(x, s)\ Poss(Eliminate(x), s) \quad \equiv \quad \varphi(x, s) \vee \exists(c, s', s'', s''')\ (S_0 \leq s' < s'' < s''' \leq s \wedge Fails(x, c, s') \\ \bigwedge \\ Fails(x, c, s'') \wedge Fails(x, c, s''')),$$

where $\varphi(x, s)$ represents other reasons for eliminating the student x. We will apply the methodology presented in section 5 to the second part of the RHS of the precondition axiom. For doing this we need to consider the following formula:

$$\exists c\ [Fails(x, c, s) \wedge \exists s''(S_0 \leq s'' < s \wedge Fails(x, c, s'') \wedge \exists s'(S_0 \leq s' < s'' \wedge Fails(x, c, s')))] \\ \bigvee \\ \exists(c, s''')\ [S_0 \leq s''' < s \wedge Fails(x, c, s''') \wedge \exists s''(S_0 \leq s'' < s \wedge Fails(x, c, s'') \\ \bigwedge \\ \exists s'(S_0 \leq s' < s'' \wedge Fails(x, c, s')))],$$

that is answer–equivalent in s to the FOPTL sentence:

$$\exists c\ (\mathit{Fails}(x,c) \wedge \diamond(\mathit{Fails}(x,c) \wedge \diamond \mathit{Fails}(x,c))) \\ \bigvee \\ \exists c\ \diamond(\mathit{Fails}(x,c) \wedge \diamond(\mathit{Fails}(x,c) \wedge \diamond \mathit{Fails}(x,c))). \tag{45}$$

In this case, if we denote with α the temporal subformula $\diamond \mathit{Fails}(x,c)$, then we introduce a new table R_α, with definition:

$$\forall(x,c)\ \neg R_\alpha(x,c,S_0). \tag{46}$$

$$\forall(a,s) Poss(a,s) \supset \forall(x,c)\ (R_\alpha(x,c,do(a,s)) \equiv R_\alpha(x,c,s) \vee \mathit{Fails}(x,c,s)). \tag{47}$$

Replacing R_α in (45), we obtain the formula

$$\exists c\ (\mathit{Fails}(x,c) \wedge \diamond(\mathit{Fails}(x,c) \wedge R_\alpha(x,c))) \\ \bigvee \\ \exists c\ \diamond(\mathit{Fails}(x,c) \wedge \diamond(\mathit{Fails}(x,c) \wedge R_\alpha(x,c))). \tag{48}$$

Now, if we denote with β the temporal subformula $\diamond(\mathit{Fails}(x,c) \wedge R_\alpha(x,c))$, then we introduce a new table R_β, with definition:

$$\forall(x,c)\ \neg R_\beta(x,c,S_0). \tag{49}$$

$$\forall(a,s) Poss(a,s) \supset \forall(x,c)\ (R_\beta(x,c,do(a,s)) \equiv R_\beta(x,c,s) \vee (\mathit{Fails}(x,c,s) \wedge R_\alpha(x,c,s))). \tag{50}$$

Replacing R_α in (48), we obtain the formula

$$\exists c\ (\mathit{Fails}(x,c) \wedge R_\beta(x,c)) \bigvee \exists c\ \diamond(\mathit{Fails}(x,c) \wedge R_\beta(x,c)) \tag{51}$$

Finally, if we denote with γ the temporal subformula $\diamond(\mathit{Fails}(x,c) \wedge R_\beta(x,c))$, then we introduce a new table R_γ, with definition:

$$\forall(x,c)\ \neg R_\gamma(x,c,S_0). \tag{52}$$

$$\forall(a,s) Poss(a,s) \supset \forall(x,c)\ (R_\gamma(x,c,do(a,s)) \equiv R_\gamma(x,c,s) \vee (\mathit{Fails}(x,c,s) \wedge R_\beta(x,c,s))). \tag{53}$$

Replacing R_γ in (51) we obtain as precondition axiom for the action *Eliminate*, in the new specification Σ_H that includes (46), (47), (49), (50), (52) and (53), the following sentence:

$$\forall(x,s)\ Poss(\mathit{Eliminate}(x),s) \equiv \varphi(x,s) \vee \exists c\ (\mathit{Fails}(x,c,s) \wedge R_\beta(x,c,s)) \vee \exists c\ R_\gamma(x,c,s)$$

The RHS of this axiom is a simple formula in s if $\varphi(x,s)$ is a simple formula in s. Otherwise, it is necessary to apply the same methodology to $\varphi(x,s)$ for obtaining a simple formula.

8 Dynamic Integrity Constraints are Static

We can use the methodology presented so far to transform a certain class of dynamic integrity constraints into static integrity constraints.

As proposed by Reiter [15, 16], an integrity constraint should be entailed by the specification of the dynamics of the database. There is some methodology developed for automating inductive proofs of static database integrity constraints, e.g. [4], with RRL [10], and [17], with NQThm [5].

With the proposed transformation, we will be in position to take advantage of this methodology, for the new task of proving dynamic integrity constraints, without having to appeal to more complex induction principles proposed to deal with them [15, 16].

It is our reformulation of Chomicki's methodology in terms of the dynamics of the history encoding relations that will allow us to *prove* dynamic integrity constraints, as opposed to *checking* them.

We will illustrate the methodology by means of an example. After that we will sketch a general treatment of dynamic ICs.

Example 17. Let us consider a specification, Σ, of a database that contains a table $salary(x,p,s)$, which indicates that employee x has salary p in the state s. In this specification we would like the integrity constraint "The salary of every employee never decreases along the database evolution" to be satisfied. It can be represented by the sentence

$$\forall(s',s'')(S_0 \leq s' < s'' \supset \forall(x,p',p'')((salary(x,p',s') \wedge salary(x,p'',s'')) \supset p' \leq p''), \tag{54}$$

that should be a logical consequence of Σ. We want to make sure that this integrity constraint holds in every current state s of the database. For doing this task, we consider a new formula $\varphi(s)$:

$$\forall(s',s'')(S_0 \leq s' < s'' \leq s \supset \forall(x,p',p'')((salary(x,p',s') \wedge salary(x,p'',s'')) \supset p' \leq p'')). \tag{55}$$

For this formula, (54) holds in every state if and only if $\forall s\ (S_0 \leq s \supset \varphi(s))$ holds wrt Σ. It possible to apply the methodology described in section 5 for transforming this sentence into a static integrity constraint, that has to be evaluated in a new specification Σ_H, generated from Σ. More precisely, (55) is equivalent to

$$\begin{gathered}\forall(x,p',p'')((\exists s'(S_0 \leq s' < s \wedge salary(x,p',s')) \wedge salary(x,p'',s)) \supset p' \leq p'')\\ \bigwedge \\ \forall s''(S_0 \leq s'' < s \supset \forall(x,p',p'')((\exists s'(S_0 \leq s' < s'' \wedge salary(x,p',s')) \wedge salary(x,p'',s'')) \supset p' \leq p''))\end{gathered}$$

This formula can be obtained applying the operator $\mathcal{C}$, at the state s, to the sentence:

$$\begin{gathered}\forall(x,p',p'')((\Diamond salary(x,p') \wedge salary(x,p'')) \supset p' \leq p'')\\ \bigwedge \\ \Box\forall(x,p',p'')((\Diamond salary(x,p') \wedge salary(x,p'')) \supset p' \leq p'')\end{gathered} \tag{56}$$

In this case we first need to consider the temporal subformulas. If $\alpha(x,p') = \Diamond salary(x,p')$, we generate the new table R_α with defintion:

$$\begin{aligned}&\forall(x,p')\ \neg R_\alpha(x,p',S_0).\\ &\forall(a,s)Poss(a,s) \supset \forall(x,p')\ (R_\alpha(x,p',do(a,s)) \equiv R_\alpha(x,p',s) \vee salary(x,p',s)).\end{aligned}$$

Replacing R_α in (56), we obtain:

$$\begin{gathered}\forall(x,p',p'')((R_\alpha(x,p') \wedge salary(x,p'')) \supset p' \leq p'')\\ \bigwedge \\ \Box\forall(x,p',p'')((R_\alpha(x,p') \wedge salary(x,p'')) \supset p' \leq p'')\end{gathered} \tag{57}$$

Finally, if we denote the temporal subformula:

$$\Box\forall(x,p',p'')((R_\alpha(x,p') \wedge salary(x,p'')) \supset p' \leq p'')$$

with β, then we introduce a new table R_β with definition in the initial state: $R_\beta(S_0)$, and SSA:

$\forall(a,s)Poss(a,s) \supset R_\beta(do(a,s)) \equiv R_\beta(s) \wedge \forall(x,p',p'')((R_\alpha(x,p',s) \wedge salary(x,p'',s)) \supset p' \leq p'')$

Replacing R_β in (57), we obtain:

$$\forall(x,p',p'')((R_\alpha(x,p',s) \wedge salary(x,p'',s)) \supset p' \leq p'') \wedge R_\beta(s)$$

So, we have that (54) holds in every state if and only if the the specification Σ_H, that contains Σ and the definitions the new fluents R_α and R_β, entails the static integrity constraint:

$$\forall s\ (S_0 \leq s \supset [\forall(x,p',p'')((R_\alpha(x,p',s) \wedge salary(x,p'',s)) \supset p' \leq p'') \wedge R_\beta(s)]).$$

In the general case, we will have an IC φ that is a sentence that we want to be true at every state s of the database. In φ there will be quantifications on states. What we can do is to generate a new constraint $\forall S_0 \leq s \supset \varphi'(s)$, where $\varphi'(s)$ is obtained from φ by restricting all the quantifications over states to $\leq s$. For example, $\forall s'$ and $\exists s''$, become $\forall s' \leq s$ and $\exists s'' \leq s$, respectively. Now, if $\varphi'(s)$ has a answer–equivalent FOPTL sentence ψ, then we generate the new static IC $\forall S_0 \leq s \supset \psi^h(s)$ and a new specifcation Σ_H, where ψ^h and Σ_H are constructed as in section 5.

It can be proved that $\Sigma \models \varphi$ iff $\Sigma_H \models \forall s(S_0 \leq s \supset \psi^h(s))$.

9 Other Related Issues

9.1 Conditions in Active Databases

9.2 Connection to Ramification and SSAs for Views

Problem: How to generate "nice" SSAs for the history encoding relations, so that we can read explicitly which are the primitive transactions that can affect them. For this we use the methodology presented in [2] for obtaining "nice" (γ^+/γ^-) SSA for views. This may require the extension of the methodology to the case of definition of recursive views, but considering the defined view in two consecutive states.

10 Related Work

10.1 Answers based on Relevance

In [19, 18] a methodology for answering hypothetical dynamic queries in the presence of a situation calculus based specification of database updates was presented. For doing this, a procedural notion of relevance of primitive transactions and tuples to other tuples in the database was defined introduced. In [20], a semantical counterpart of the notion of relevance was introduced and used as a foundational basis for the application of the procedural notion of relevance. The main idea is to answer an hypothetical dynamic query by considering the actions that are relevant to the query and updating of that part of the database that is relevant to the query. That is, we consider only minimal updates of the database, and those updates correspond only to a minimal number of actions in the given transaction list that virtually updates the original database.

This methodology is based on explicit relevance asks and answers the query from outside, by collecting and processing truth values of formulas that are obtained from the different virtual states of the database along its history. So, several (virtual) database states had to be explicitly considered for answering the query.

With FOPTL we can ask dynamic queries that we cannot pose with the *canonical query format* introduced in [18], actually queries that ask about the existence of individuals with some property along the whole trajectory (as in example 6).

11 Conclusions and Related Work

Acknowledgments: This research has been partially supported by FONDECYT (Grant # 1971304). Part of this work was done while the second author was on sabbatical at the Technical University of Berlin. He is grateful to Ralf Kutsche and all the people in the "Computergestüzte Informationssysteme" (CIS) group for their support and hospitality; and to the Graduierten Kolleg "Verteilte Informationssysteme", the DAAD and the Catholic University of Chile (DIPUC) for their financial support. We are grateful to Javier Pinto for his generous support.

References

[1] S. Abiteboul, L. Herr, and J. Van der Bussche. Temporal versus First–Order Logic to Query Temporal Databases. In *Proc. ACM Symposium on Principles of Database Systems (PODS'96)*, pages 49–57, 1996.

[2] M. Arenas and L. Bertossi. Specifying the Dynamics of Database Views (Extended Version). Can be found as ext-views.ps in http://dcc.ing.puc.cl/~bertossi/, 1997.

[3] L. Bertossi, M. Arenas, and C. Ferretti. SCDBR: An Automated Reasoner for Specifications of Database Updates. *Journal of Intelligent Information Systems*, To appear.

[4] L. Bertossi, J. Pinto, P. Saez, D. Kapur, and M. Subramaniam. Automating Proofs of Integrity Constraints in the Situation Calculus. In *Foundations of Intelligent Systems. Proc. Ninth International Symposium on Methodologies for Intelligent Systems (ISMIS'96), Zakopane, Poland*, pages 212–222. Springer, LNAI 1079, 1996.

[5] R.S. Boyer and J S. Moore. *A Ccomputational Logic Handbook*. Academic Press, 1988.

[6] J. Chomicki. Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. *ACM Transactions on Database Systems*, 20(2):149–186, June 1995.

[7] A. Thayse (ed.). *From Modal Logic to Deductive Databases*. J. Wiley & Sons, 1989.

[8] D. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects*, volume 1. Oxford University Press, 1994.

[9] S. Hanks and D. McDermott. Default Reasoning, Nonmonotonic Logics, and the Frame Problem. In *Proc. National Conference on Artificail Intelligence*, pages 328–333, 1986.

[10] D. Kapur and H. Zhang. An Overview of Rewrite Rule Laboratory (RRL). *J. of Computer and Mathematics with Applications*, 29(2):91–114, 1995.

[11] F. Lin and R. Reiter. State Constraints Revisited. *Journal of Logic and Computation*, 4(5):655–678, 1994. Special Issue on Action and Processes.

[12] F. Lin and R. Reiter. How to Progress a Database. *Artificial Intelligence*, 92(1–2):131–167, 1997.

[13] J. McCarthy and P. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502, Edinburgh, Scotland, 1969. Edinburgh University Press.

[14] R. Reiter. The Frame Problem in the Situation Calculus: a Simple Solution (Sometimes) and a Completeness Result for Goal Regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.

[15] R. Reiter. Proving Properties of States in the Situation Calculus. *Artificial Intelligence*, 64(2):337–351, 1993.

[16] R. Reiter. On Specifying Database Updates. *Journal of Logic Programming*, 25(1):53–91, 1995.

[17] T. Sheard and D. Stemple. Automatic Verification of Database Transaction Safety. *ACM Transactions on Database Systems*, 14(3):322–368, 1989.

[18] B. Siu and L. Bertossi. Answering Historical Queries in Databases using Relevance. Submitted in 1996. http://dcc.ing.puc.cl/ bertossi/papers/hq-ssp.ps.

[19] B. Siu and L. Bertossi. Answering Historical Queries in Databases (Extended Abstract). In *Proc. XVI International Conference of the Chilean Computer Science Society (SCCC'96), M. V. Zelkowitz and P. Straub (eds.)*, pages 56–66, 1996.

[20] B. Siu, L. Bertossi, and M. Arenas. A Semantical Notion of Relevance in Specifications of Database Updates and its Computational Counterpart. Submitted in 1997. http://dcc.ing.puc.cl/ bertossi/papers/jlp.ps.

[21] R. Snodgrass and I. Ahn. Temporal Databases. *IEEE Computer*, September:35–42, 1986.