Journal of Intelligent Information Systems, 19:2, 231–259, 2002

Hypothetical Temporal Reasoning in Databases*

MARCELO ARENAS† marenas@cs.toronto.edu
Pontificia Universidad Catolica de Chile, Escuela de Ingenieria, Departamento de Ciencia de Computacion, Santiago, Chile

LEOPOLDO BERTOSSI bertossi@scs.carleton.ca
Carleton University, School of Computer Science, Ottawa, Canada

Abstract. In this paper we integrate a history–encoding based methodology for checking dynamic database integrity constraints into a situation-calculus based specification of relational database updates. 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 all the information associated to the generated states, (2) State and prove dynamic integrity constraints as static integrity constraints, (3). Transform history dependent preconditions for updates into local preconditions.

The methodology presented here is based on the introduction of operators of predicate past temporal logic as macros into the specifications, written in the situation calculus, of the dynamics of a database. Temporal subformulas of a query are treated as auxiliary views with the corresponding specification of their dynamics. An implementation of hypothetical temporal query answering is presented.

Keywords: database dynamics, hypothetical reasoning, historical queries, dynamic integrity constraints

1. Introduction

In Reiter (1995), as an application of his solution to the frame problem (Reiter, 1991), Ray Reiter proposed to specify the transaction based updates of a relational database by means of a particular kind of axioms written in the situation calculus (SC) (McCarthy and Hayes, 1969). In Bertossi et al. (1998) the implementation and the functionalities of SCDBR, a computational system for doing automated reasoning from and about those specifications (Bertossi et al., 1998), are reported.

We are motivated by the problem of answering queries about different states[1] in the evolution of a relational database, when the database is virtually updated by the execution of a sequence of primitive transactions, that are indecomposable and domain dependent transactions. For example, we want to consider queries of the form "Has it always been the case that the database has satisfied a given condition C?," or "Has there been a state of the database where a certain condition C has been satisfied?," or "Has the salary of some employee decreased along the database evolution?." Reiter raised this problem in the context of his specifications of transaction based database updates (Reiter, 1995).

*Dedicated to the memory of our dear friend and colleague Javier Pinto (1959–2001), who made important contributions to the logical foundations of the situation calculus.

†Present address: Department of Computer Science, University of Toronto, Toronto, Canada.

Although there is no explicit time in our situation calculus, we call these queries "temporal queries," due to their similarity with dynamic integrity constraints (Reiter, 1995), also called "temporal constraints" (Chomicki, 1995).[2] Furthermore, we call these queries "hypothetical" because we start from an initial, physical database at an initial 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 a query about the generated sequence of states, without physically updating the whole database accordingly (and possibly keeping the data for every intermediate state). We are interested in querying this whole virtual evolution of the database.

The problem of answering this kind of queries was treated in detail in Siu and Bertossi (1996) and a solution was implemented as reported in Bertossi et al. (1998). Nevertheless that solution is based on a kind of minimal progression of the database that depends on a detailed syntactical processing of the axioms of the specification, and the particular syntactical form of them.

In this paper we reconsider this problem and we propose a new solution that relies on processing the query itself, rather than the underlying axioms. This is done on the basis of (1) a formulation of the query in a situation calculus language that contains temporal operators inspired by first order past temporal logic (FOPTL) (Gabbay et al., 1994), (2) a reformulation of Chomicki's history encoding methodology for efficiently checking temporal integrity constraints (Chomicki, 1995), in the context of situation calculus specifications of database updates, and, in particular, (3) a specification in the situation calculus of the evolution of new history encoding auxiliary relations that are generated from the query.

It turns out that the methodology we develop for answering queries can be adapted to give a solution to other reasoning problems. Here we show how to transform dynamic integrity constraints into static integrity constraints, so that any available methodology for handling static integrity constraints can be adapted for the dynamic case. In particular, we can take advantage of our results on automated proving of static integrity constraints (Bertossi et al., 1996) when dealing with the dynamic case.

The other problem we solve consists in transforming preconditions for action executions that depend on the history of the database into preconditions that depend on the local, execution state.

This paper is concerned mainly with the problems of modeling and doing hypothetical reasoning in databases, a subject that has not received enough attention (but see Bonner, 1990; Bonner and Kifer, 1998; Chen, 1997). We think hypothetical reasoning will become more and more important in databases that are used for decision support, where "what if" questions are relevant, as the ones emerging in on-line analytical processing (OLAP) and datawarehousing (Chaudhuri and Dayal, 1997). To have computational tools that can be used to explore different courses of action without having to commit to any of them; in particular, without having to physically update the database, is likely to be very important in many applications of information systems, specially in the presence of rich primitive transactions, like ours, that may affect several tables at the same time.

This paper is structured as follows. In Section 2 we briefly describe the situation calculus based specifications of database updates. In Section 2.1, we introduce temporal queries and constraints in a situation calculus framework. In Section 3.1 we introduce our language

for posing temporal queries. In Section 3.2 we introduce the elements we need to evaluate queries, in particular, the specification of the dynamics of the auxiliary, history encoding, views; and we address the problem of answering the hypothetical temporal queries. In Section 4 we apply the machinery developed in previous sections to the problem of transforming dynamic integrity constraints into static integrity constraints. In Section 5 we apply our methodology to the problem of transforming history dependent transactions, i.e. transactions whose preconditions are temporal, into transactions with local conditions. In Section 6 we sketch some possible extensions of the methodologies introduced in the previous sections. In Section 7 we compare our work with other approaches in the literature, we comment on further work, and draw some conclusions. In Appendix A, we illustrate our implementation of the developed methodology. In Appendix B some proofs are given.

2. Specifying the database dynamics

We will show the main ingredients of a specification in the situation calculus of transaction based database updates, as proposed in Reiter (1995). The SC is a family of languages of many sorted predicate logic used to represent knowledge and reason about dynamic domains that are subject to discrete changes caused by action executions. In its languages, we find domain individuals, states and primitive transactions, i.e. domain specific and indecomposable transactions, that we will also call "actions," and which are at the same first order level. In consequence, first order quantification over all these sorts of individuals is possible. They are usually denoted by $\forall \bar{x}, \forall s, \forall a$ respectively.

Among others we find the following advantages in using the SC as a specification language: (1) It has a clear and well understood semantics. (2) Everything already done in the literature with respect to applications of predicate logic to DBs can be done here, in particular, all static and extensional aspects of databases and query languages are included. (3) Dynamic aspects can be integrated at the same object level, in particular, it is possible to specify how the database evolves as transactions are executed. (4) It is possible to reason in an automated manner from the specification and to extract algorithms for different computational tasks from it. (5) In particular, it is possible to reason explicitly about DB transactions and their effects. (6) In this form, it is possible to extend the functionality of usual commercial DBMSs.

In every SC language we find a constant for an initial state, S_0, a function symbol, *do*, so that $do(a, s)$ denotes the successor state that results from the execution of action a at state s. We also find a predicate, $Poss(a, s)$, with the intended meaning that action a is possible at state s. In a particular SC language we will find function names for parameterized primitive transactions (actions), $A(\bar{x})$, and names for tables, $F(\bar{x}, s)$, that is, predicates with a single state argument[3]. If T is a sequence of action terms $A_1, \ldots, A_n$, to be executed in that order, we abbreviate the situation $do(A_n,\ do(A_{n-1},\ do(\ldots do(A_1, s), \ldots))$ with $do([A_1, \ldots, A_n], s)$ or simply $do(T, s)$.

As in Lin and Reiter (1994), we will assume that the following *foundational axioms of the situation calculus* underlie any database specification[4]: (1) Unique Names Axioms for Actions: $A_i(\bar{x}) = A_j(\bar{y})$, for all different action names A_i, A_j; $\forall(\bar{x}, \bar{y})[A_i(\bar{x}) = A_j(\bar{y}) \supset \bar{x} = \bar{y}]$. (2) Unique Names Axioms for States: $S_0 \neq do(a, s), do(a_1, s_1) = do(a_2, s_2) \supset$

$a_1 = a_2 \wedge s_1 = s_2$. (3) For some reasoning tasks we need an Induction Axiom on States: $\forall P[(P(S_0) \wedge \forall s \forall a(P(s) \supset P(do(a, s))) \supset \forall s\, P(s)]$.

A specification of the transaction based updates on a particular database will contain the following axioms: (4) A set, Σ_0, of SC sentences that do not mention any state term other than S_0. This is knowledge about the initial state, and state independent knowledge. (5) Action Precondition Axioms: For each action name A, a precondition axiom of the form

$$\forall(\bar{x}, s)[Poss(A(\bar{x}), s) \equiv \pi_A(\bar{x}, s)], \tag{1}$$

where $\pi_A(\bar{x}, s)$ is a SC formula that is *simple in s*, that is, it contains no state term other than s, in particular, no *do* symbol, no quantifications on states, and no occurrences of the *Poss* predicate (Reiter, 1995). (6) Successor State Axioms (SSAs): For every table $F(\bar{x}, s)$, an axiom of the form

$$\forall(a, s)Poss(a, s) \supset \forall \bar{x}[F(\bar{x}, do(a, s)) \equiv \Phi_F(\bar{x}, a, s)], \tag{2}$$

where Φ_F is a formula simple in s, in particular, it does not contain the *do* symbol. Provided there is complete knowledge at the initial state, as is usually the case in relational databases, this axiom completely determines the contents of table F at an arbitrary legal database state, i.e. reached from S_0 by a finite sequence of transactions that are possible at their execution states. We are usually interested in reasoning about the states that are accessible in this form from the initial situation. For this purpose, an *accessibility relation* on states, $\leq$, can be defined on the basis of the induction axiom by means of the conditions: $\neg s < S_0, s < do(a, s') \equiv Poss(a, s') \wedge s \leq s'$.

We will denote this specification with Σ. It includes the initial database Σ_0 and the definition of the accessibility relation.

Example 1. Consider a database of a company, with the following 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.

and primitive transactions:

hire(x): Person x is hired by the company.
fire(x): Person x is fired by the company.
promote(x): Person x is promoted to chief executive officer.
changeSalary(x, p): The salary of the person x is changed to p dollars.

The specification of the initial database has the the following formulas:

$$\forall x(Emp(x, S_0) \equiv x = john \vee x = ernest \vee x = page),$$
$$sue \neq john, sue \neq ernest, sue \neq page,$$
$$john \neq ernest, john \neq page, ernest \neq page.$$

That is, John, Ernest and Page are the only employees of the company at the initial database state.

The relations in this specification have the following successor state axioms (see (2)):

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

For example, the first SSA says that x is an employee at an arbitrary legal successor state if he was just hired or he already was an employee and he was not fired in the transition to the successor state.

Now, assume that we have the following precondition axioms for the actions in the database.

$$\begin{aligned}
& \forall(x,s)[Poss(hire(x), s) \equiv \neg Emp(x,s)]. \\
& \forall(x,s)[Poss(fire(x), s) \equiv Emp(x,s)]. \\
& \forall(x,s)[Poss(prompte(x), s) \equiv Emp(x,s) \wedge \neg Ceo(x,s)]. \\
& \forall(x,\, p,\, s)[Poss(changeSalary(x,\, p), s) \equiv Emp(x,s) \wedge \\
& \qquad\qquad \forall p'(Salary(x,\, p',\, s) \supset p' \leq p)].
\end{aligned}$$

□

The *regression operator*, $\mathcal{R}$, (Reiter, 1991, 1995) applied to a formula containing a successor state returns an equivalent formula (with respect to the specification) evaluated at the preceding state. This is done by using the SSAs. More precisely, if predicate F, appearing in a formula ψ, has a SSA like (2), then the operator, $\mathcal{R}$, applied to ψ, replaces each occurrence of an atomic formula of the form $F(\bar{t}, do(a,s))$ in ψ by $\Phi_F(\bar{t}, a, s)$.

The regression operator is implemented in SCDBR, a computational system for reasoning about and from specifications of database dynamics as presented in this section (Bertossi et al., 1998).

2.1. Temporal queries and constraints

In the context of such DB specifications, a temporal query is a SC sentence φ in which all the states involved, including quantified states, lie on a finite state path $S_0 \leq S_1 \leq \cdots \leq S_n$, with $S_i = do(A_i, do(A_{i-1}, \ldots, do(A_1, S_0)\ldots))$, for a sequence of ground actions terms $A_1, \ldots, A_n$, for some n. The query is true if and only if $\Sigma \models \varphi$.

Example 2. In Example 1, the temporal query "*Has Sue been working in the company in all states generated by sequence T at* S_0?" can be expressed in the SC by means of $\forall s(S_0 \leq s \leq S_n \supset Emp(sue, s))$.

The following sentence could be also a temporal query

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

It asks whether the salary has not decreased. □

A static integrity constraint is a formula of the form $\forall s(S_0 \leq s \supset \varphi(s))$, where $\varphi(s)$ is simple in the state variable s, such that $\Sigma \models \forall s(S_0 \leq s \supset \varphi(s))$ is expected to hold (Reiter, 1995; Lin and Reiter, 1994). A dynamic (or temporal) integrity constraint is a SC sentence φ of the form $\forall s_1 \cdots \forall s_n(C(S_0, s_1, \ldots, s_n) \supset \varphi(s_1, \ldots, s_n))$, that should be entailed by Σ. Here, $C(S_0, s_1, \ldots, s_n)$ is a formula that imposes a linear order constraint on the states $S_0, s_1, \ldots, s_n$ in terms of the accessibility predicate $\leq$[5].

Example 3. The sentence $\forall s(S_0 \leq s \supset \forall p\ (Salary\ (sue, p, s) \supset p \geq 4000))$ could be a static integrity constraint, stating that Sue's salary can not be lower than 4000. The sentence

$$\forall(s', s'')(S_0 \leq s' < s'' \supset \\ \forall(x, p', p'')((Salary(x, p', s') \wedge Salary(x, p'', s'')) \supset p' \leq p''). \tag{3}$$

is a dynamic integrity constraint expressing that a salary never decreases. □

In general, we will not have explicit integrity constraints in our specification, Σ, of the database dynamics. We expect them to be logical consequences of Σ (Reiter, 1995; Lin and Reiter, 1994).

In the next section we will introduce temporal operators as found in past temporal logic into the situation calculus. With these operators we will formulate queries and constraints, and their new syntactic form will allow us to process and evaluate them.

3. Answering queries

In Chomicki (1995), the problem of checking temporal constraints stated in FOPTL was considered. These are constraints that talk about, and relate, different states of the database. There we find a sequence of transactions that are physically executed, and in order to minimize the cost of checking, one progressively updates new defined relations, or auxiliary views, r_α, that correspond to the temporal subformulas, α, in the constraint. These views encode part of the database evolution up to a current database state. They are defined and updated in such a way that they store the historical information that is relevant to give an answer to the query about the satisfaction of the integrity constraint once the final (current) state is reached. Then a new, non-temporal, but local and static query can be posed at the final state.

In this paper we will combine our reconstruction of Chomicki's history encoding in the context of specifications of the dynamics of a database with the possibility, opened by those specifications, of reasoning about the database evolution without having to physically update the database. In consequence, we will be in position to do hypothetical temporal reasoning about the database evolution. We can say that while Chomicki answers the query by positioning at the final physical state of the database, we query a single future, virtual state from the initial, physical state. The fact that we are doing virtual updates makes it possible to apply our methodology to any temporal query, whereas, in the presence of physical updates, the queries have to be fixed, predetermined in advance[6].

3.1. A query language

As discussed in Section 2.1, a temporal query is a sentence φ in which all the states involved, including the quantified states, lie on a finite state path $S_0 \leq S_1 \leq \cdots \leq S_n$, with $S_i = do(A_i, do(A_{i-1}, \ldots, do(A_1, S_0)\ldots))$, for a sequence of ground actions terms $A_1, \ldots, A_n$.

In order to answer this kind of queries on an algorithmic basis, we need to define them in a precise sense. Thus, we need to define a query language for asking about the history of a sequence of states. To achieve this, we will introduce in the situation calculus some temporal operators inspired by first order past temporal logic, and a *macro*, `holds`. With these new elements we will be in position to represent an important class of temporal queries. Nevertheless, if desired, the application of `holds` to a formula with temporal operators could be always rewritten into a usual situation calculus formula.

The SC contains predicates with a state as an argument. For example, we use $P(a, s)$ to state that a is an element of table P in the state s. We may eliminate the situation term from predicate P, and use a new meta-predicate, `holds`, and write `holds`$(P(a), s)$ with the same meaning as before. Actually, we would like to extend the application of `holds` to more complex formulas, derived from SC formulas, but keeping the state dependency in the second argument of `holds`.

Definition 1. A formula φ is *state suppressed* (an *ss-formula*) if it is constructed as usual from state independent predicates, state dependent predicates (i.e. database tables or fluents) with the state argument supressed, boolean connectives, and first order quantifications on domain individuals.

For example, the following is an ss-formula: $\forall x \exists p(Ceo(x) \wedge Emp(x, p) \supset p \geq 5000)$.[7] The state arguments have been suppressed from the tables. Predicate `holds` will have an ss-formula in its first argument, and a state in the second argument. This would make `holds` a second order predicate. We can go back to first order expressions by considering `holds` as a macro, as an abbreviation, as something that can be rewritten into an expression of the original situation calculus. Thus, `holds`(φ, s) where φ is an ss-formula, is defined recursively

as follows:

$$
\begin{aligned}
&\texttt{holds}(t_1 * t_2, s) &&:= t_1 * t_2, \text{ for terms for individuals } t_1, t_2, \text{ and} \\
&&&\quad * \in \{<, >, \leq, \geq, =, \neq\}. \\
&\texttt{holds}(P(\bar{x}), s) &&:= P(\bar{x}), \text{ if } P \text{ is a state independent predicate} \\
&\texttt{holds}(F(\bar{x}), s) &&:= F(\bar{x}, s), \text{ if } F \text{ is a predicate for a database table} \\
&\texttt{holds}(\neg\varphi, s) &&:= \neg\texttt{holds}(\varphi, s) \\
&\texttt{holds}(\varphi \wedge \psi, s) &&:= \texttt{holds}(\varphi, s) \wedge \texttt{holds}(\psi, s) \\
&\texttt{holds}(\varphi \vee \psi, s) &&:= \texttt{holds}(\varphi, s) \vee \texttt{holds}(\psi, s) \\
&\texttt{holds}(\varphi \supset \psi, s) &&:= \texttt{holds}(\varphi, s) \supset \texttt{holds}(\psi, s) \\
&\texttt{holds}(\varphi \equiv \psi, s) &&:= \texttt{holds}(\varphi, s) \equiv \texttt{holds}(\psi, s) \\
&\texttt{holds}(\exists x\varphi, s) &&:= \exists x\texttt{holds}(\varphi, s), \text{ if } x \text{ is a variable for domain individuals} \\
&\texttt{holds}(\forall x\varphi, s) &&:= \forall x\texttt{holds}(\varphi, s), \text{ if } x \text{ is a variable for domain individuals.}
\end{aligned}
$$

An advantage of using the macro `holds` is that we can extend the class of formulas φ in $\texttt{holds}(\varphi, s)$ in such a way that they contain new, temporal operators that represent subformulas with some useful, natural and common quantification over states. In addition, we can make the final, evaluation state, s, explicit. For example, we want to represent in a compact and distinguishable way the formula $\forall s'(S_0 \leq s' < s \supset P(a, s'))$ which says that a is an element of table P in every state previous to s, without using an explicit quantification over states. For doing this, we introduce a logical temporal operator, $\Box$, defined by

$$\texttt{holds}(\Box\, P(a), s) := \forall s'(S_0 \leq s' < s \supset \texttt{holds}(P(a), s')).$$

More precisely, for posing temporal queries, we will introduce in the SC the four typical temporal operators of first order past temporal logic, the same operators considered in (Chomicki, 1995). The intended meanings of them are: (a) $\bullet\varphi$ for "φ was true at the previous moment of time". (b) φ **since** ψ for "ψ was true at some time in the past and from that time on, φ has been true". (c) $\diamond\varphi$ for "Sometime in the past φ was true". (d) $\Box\varphi$ for "Always in the past φ was true".[8]

They will be introduced as macros though, via the `holds` predicate. In consequence, the class of ss-formulas (Definition 1) has to be extended by means of the extra rule: If φ, ψ are ss-formulas, then $\bullet\varphi$, φ **since** ψ, $\diamond\varphi$, $\Box\varphi$ are also ss-formulas.

The combinations of `holds` and the temporal operators are defined by the following macros, that can be rewritten as SC formulas as follows:

$$
\begin{aligned}
&\texttt{holds}(\bullet\varphi, s) &&:= \exists(a, s')(s = do(a, s') \wedge \texttt{holds}(\varphi, s')) \\
&\texttt{holds}(\varphi \textbf{ since } \psi, s) &&:= \exists s'(S_0 \leq s' < s \wedge \texttt{holds}(\psi, s') \wedge \\
&&&\qquad \forall s''(s' < s'' \leq s \supset \texttt{holds}(\varphi, s''))) \\
&\texttt{holds}(\diamond\varphi, s) &&:= \exists s'(S_0 \leq s' < s \wedge \texttt{holds}(\varphi, s')) \\
&\texttt{holds}(\Box\varphi, s) &&:= \forall s'(S_0 \leq s' < s \supset \texttt{holds}(\varphi, s')).
\end{aligned}
$$

This is a recursive definition. In it, φ and ψ are formulas that may include connectives $\neg$, $\vee$, $\wedge$, $\supset$ and $\equiv$; quantification over domain individuals; and operators $\bullet$, **since**, $\diamond$ and $\Box$.

From now on, our temporal query language will consist of the formulas we just defined. More precisely, our temporal queries will be of the form

$$\texttt{holds}(\varphi, do(T, S_0))?, \tag{4}$$

where φ is an ss-formula and T is a sequence of ground actions.

Formula φ in (4) will possibly contain temporal operators, that is, it may contain subformulas starting with an application of a temporal operator. In Section 3.2, to each of these subformulas, α, we will associate a new, auxiliary, history encoding view, R_α. Next, for these views we will derive specifications of their dynamics, and use them in the process of query answering. Before doing this, we present some examples of temporal queries expressed in terms of the new operators.

Example 4. We can express the queries shown in Example 2 as follows. "*Has sue been working in the company in all states generated by a sequence of actions T at* S_0?": $\texttt{holds}(Emp(sue) \wedge \Box\, Emp(sue),\ do(T, S_0))$. "*Is is true that the salaries have never decreased along the states generated by action sequence T executed at* S_0?":

$$\texttt{holds}(\forall(x, p', p'')((Salary(x, p'') \wedge \Diamond\, Salary(x, p') \supset p' \leq p'') \wedge \\ \Box\,(Salary(x, p'') \wedge \Diamond\, Salary(x, p') \supset p' \leq p'')), do(T, S_0)).$$

Example 5. The query "*Was Joe hired as an employee of a lower rank before becoming a Chief Executive Officer (CEO) in all states generated by a sequence of actions T at* S_0?" can be expressed by the formula

$$\texttt{holds}(\Box\,(Emp(joe) \wedge \neg Ceo(joe)) \wedge Emp(joe) \wedge Ceo(joe) \vee \\ [Emp(joe) \wedge Ceo(joe)]\textbf{since} \\ [Emp(joe) \wedge Ceo(joe) \wedge \Box\,(Emp(joe) \wedge \neg Ceo(joe))], do(T, S_0))$$

Example 6. The query "*Is there anybody who has always been working in the company* (*along the execution of the sequence of actions T from* S_0)?" can be expressed by the formula

$$\texttt{holds}(\exists x(Emp(x) \wedge \Box\, Emp(x)), do(T, S_0)). \qquad \Box$$

With the temporal operators we can express an interesting and natural class of temporal queries. The introduction of the temporal operators cannot be a substitute for the whole expressive power of the situation calculus (Abiteboul et al., 1996), nevertheless we can express with them the queries we need in most common practical applications.

3.2. Evaluating the query

Our starting point consists of a SC specification Σ as in Section 2, and a query $\texttt{holds}\,(\varphi, S)$, where φ is an ss-sentence, possibly containing temporal operators, to be evaluated at the

final state $S = do(T, S_0)$. As expected, this formula implicitly refers to the states between S_0 and S.

In order to answer the query, we will construct a new SC specification Σ_H that extends Σ, and a new SC sentence, $\mathtt{H}(\varphi, S)$, such that the answer to the original query, $\Sigma \models \mathtt{holds}(\varphi, S)$?, coincides with the answer obtained from the evaluation of $\mathtt{H}(\varphi, S)$ with respect to Σ_H. The new sentence, $\mathtt{H}(\varphi, S)$, refers only to the state S, and Σ_H contains a specification of the dynamics of some new, history encoding, auxiliary relations that correspond to the temporal subformulas in φ. Being in this new scenario, we can use any algorithm for answering queries that refer to a single, future state of the database.

First we will generate $\mathtt{H}(\varphi, S)$, for an ss-formula φ and ground state S. Next, we will show how to generate Σ_H.

1. If φ is of the form $t_1 * t_2$, where t_1 and t_2 are terms (for domain individuals) and $* \in \{<, >, \leq, \geq, =, \neq\}$, then $\mathtt{H}(\varphi, S) := \varphi$.
2. If φ is of the form $P(\bar{t})$, where P is a state independent predicate, then $\mathtt{H}(\varphi, S) := P(\bar{t})$.
3. If φ is of the form $F(\bar{t})$, where F is a predicate for a database table, then $\mathtt{H}(\varphi, S) := F(\bar{t}, S)$.
4. If φ is $\neg\psi$, then $\mathtt{H}(\varphi, S) := \neg\mathtt{H}(\psi, S)$.
5. $\mathtt{H}(\psi * \theta, S) := \mathtt{H}(\psi, S) * \mathtt{H}(\theta, S)$, where $*$ is any of the usual binary propositional connectives.
6. $\mathtt{H}(Qx\,\varphi, S) := Qx\,\mathtt{H}(\varphi, S)$, where Q is any of the usual first order quantifiers.
7. If φ is $\bullet\psi(\bar{x})$, $\diamond\psi(\bar{x})$ or $\Box\psi(\bar{x})$, where ψ does not contain any of the operators $\bullet$, **since**, $\diamond$ and $\Box$, then $\mathtt{H}(\varphi, S) := R_\varphi(\bar{x}, S)$, where R_φ is a new table name.
8. If $\varphi(\bar{x})$ is $\psi(\bar{x})$ **since** $\theta(\bar{x})$, where $\psi(\bar{x})$ and $\theta(\bar{x})$ do not include any of the operators $\bullet$, **since**, $\diamond$ and $\Box$, then $\mathtt{H}(\varphi, S) := R_\varphi(\bar{x}, S)$, where R_φ is a new table name.

By bottom-up transformation of a formula φ that appears in the macros `holds`, we can always obtain such a formula $\mathtt{H}(\varphi, S)$. Notice that this is a SC formula that is simple in the state S, i.e. it talks about an isolated state, S.

Now, we will specify the dynamics of the new tables introduced in the last two cases in the inductive definition above by means of appropriate SSAs:

(a) Let $\alpha(\bar{x})$ be of the form $\bullet\psi(\bar{x})$. This formula is true at a given state iff $\psi(\bar{x})$ is true at the previous state. Then, the new table $R_\alpha(\bar{x}, s)$ has the following SSA: $\forall(a, s)Poss(a, s) \supset \forall\bar{x}(R_\alpha(\bar{x}, do(a, s)) \equiv \mathtt{H}(\psi(\bar{x}), s))$. At the initial state $\alpha(\bar{x})$ is false for each $\bar{x}$, because S_0 has no predecessor state, so we specify $\forall\bar{x}\neg R_\alpha(\bar{x}, S_0)$.

(b) Let $\alpha(\bar{x})$ be of the form $\psi(\bar{x})$ **since** $\theta(\bar{x})$. 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 $R_\alpha(\bar{x}, s)$ it holds:

$$\forall(a, s)Poss(a, s) \supset \forall\bar{x}(R_\alpha(\bar{x}, do(a, s)) \equiv \mathtt{H}(\psi(\bar{x}), do(a, s)) \wedge (R_\alpha(\bar{x}, s) \vee \mathtt{H}(\theta(\bar{x}), s))).$$

This is not a SSA of the form (2), because there is a $do(a, s)$ term in one of the formulas 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 $do(a, s)$, into a formula instantiated at the previous state, s (see Section 2). So, we obtain:

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

Notice that the application of the regression operator leaves the RHS of the equivalence above as a simple formula in s. Also notice that when α is a sentence, then the SC formula $R_\alpha(s)$ becomes a situation dependent propositional predicate. Finally, we also specify $\forall\bar{x}\neg R_\alpha(\bar{x}, S_0)$.

(c) Let $\alpha(\bar{x})$ be of the form $\diamond\psi(\bar{x})$. Given that $\diamond\psi(\bar{x}) := true$ **since** $\psi(\bar{x})$, the new table R_α has the specification:

$$\forall\bar{x}\neg R_\alpha(\bar{x}, S_0),$$
$$\forall(a, s)Poss(a, s) \supset \forall\bar{x}(R_\alpha(\bar{x}, do(a, s)) \equiv \mathtt{H}(\psi(\bar{x}), s) \vee R_\alpha(\bar{x}, s)).$$

Let $\beta(\bar{x})$ be of the form $\Box\psi(\bar{x})$. Since $\Box\psi(\bar{x}) := \neg\diamond\neg\psi(\bar{x})$, R_β has the specification:

$$\forall\bar{x}R_\beta(\bar{x}, S_0)$$
$$\forall(a, s)Poss(a, s) \supset \forall\bar{x}(R_\beta(\bar{x}, do(a, s)) \equiv \mathtt{H}(\psi(\bar{x}), s) \wedge R_\beta(\bar{x}, s)).$$

Example 7. Assume that the original specification Σ contains the following SSA for the table $P(x, s)$:

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

We want to evaluate the query $\exists x(P(x)$ **since** $\diamond Q(x))$ at state $S = do(T, S_0)$. If β is $\diamond Q(x)$, 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 the query, we obtain $\exists x(P(x)$ **since** $R_\beta(x))$. If the formula inside the quantifier is $\alpha(x)$, for the new table R_α we have

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

Replacing $\mathcal{R}[P(x, do(a, s))]$ by the RHS of the SSA for P, we obtain the following SSA for R_α:

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

The new query is $\exists x\, R_\alpha(x, S)$. □

The following proposition shows why we can use H and Σ_H to answer queries about a specification Σ, constructed by using holds.

Proposition 1. *Let Σ be a SC specification and T a legal sequence of ground actions*[9], *then $\Sigma \models \texttt{holds}(\varphi, do(T, S_0))$ if and only if $\Sigma_H \models \texttt{H}(\varphi, do(T, S_0))$.*

Notice that holds $(\varphi, do(T, S_0))$ is instantiated at the final state $do(T, S_0)$, and this is the only state mentioned in the formula. So, we can see that we have transformed our problem of answering a temporal query with respect to a virtually updated database into the *temporal projection problem* of AI (Hanks and McDermott, 1986), that is, the problem of querying a future state obtained by the execution of a sequence of actions. To solve this problem we may apply some existing techniques for Reiter like specifications, e.g. Reiter's query regression (Reiter, 1995), minimal rolling forward of the database based on information that is relevant to the query (Bertossi et al., 1998; Siu and Bertossi, 1996), or even full progression of the database (Lin and Reiter, 1997). All these methodologies are supported by the database reasoner SCDBR (Bertossi et al., 1998).

Example 8. We want to know if there is someone who has always been working in the company, in all states generated by the execution of the sequences of actions $T = [hire(sue), fire(john)]$ from the initial situation. So, we are asking whether

$$\Sigma \models \texttt{holds}(\exists x(Emp(x) \wedge \Box Emp(x)), do(T, S_0)).$$

Applying our methodology we obtain the new SC query

$$\exists x(Emp(x, do(T, S_0)) \wedge R_\alpha(x, do(T, S_0))),$$

and the original specification extended to Σ_H by adding $\forall x\, R_\alpha(x, S_0)$ and

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

Then we ask if $\Sigma_H \models \exists x(Emp(x, do(T, S_0)) \wedge R_\alpha(x, do(T, S_0)))$. Now the query is simple in the final state $do(T, S_0)$.

Running the regression operator twice on the RHS and simplifying the resulting steps by means of the unique names axioms for actions, we obtain the following query to be posed to the initial database D_0:

$$\exists x((x = sue \wedge Emp(x, S_0)) \wedge x \neq john \wedge Emp(x, S_0) \wedge R_\alpha(x, S_0)).$$

Simplifying $R_\alpha(x, S_0)$ to *true*, we obtain the equivalent query

$$Emp(sue, S_0) \vee \exists x(x \neq john \wedge Emp(x, S_0)).$$

SCDBR can answer this query by calling a conventional DBMS on the initial database, or a Prolog program if the initial database is a Prolog database.

The implementation in SCDBR of the methodology we have presented so far is described in Appendix A.

4. Transforming dynamic integrity constraints

In Section 2 we defined a Static Integrity Constraint (SIC) as a formula of the form $\forall s(S_0 \leq s \supset \varphi(s))$, where $\varphi(s)$ was a formula simple in s. By using the macro `holds` we can extend this definition by saying that a static integrity constraint is a formula of the form

$$\forall s(S_0 \leq s \supset \texttt{holds}(\varphi, s)). \tag{5}$$

If φ is a formula that does not include operators $\bullet$, **since**, $\diamond$ and $\Box$, then the previous formula is a static integrity constraint of the form showed in Section 2. If φ includes these operators, then it can represent a more complex kind of integrity constraint. In fact, by means of this operators it is possible to represent several Dynamic Integrity Constraints (DICs). Thus, by using the equivalence between macros `holds` and `H` it is possible to transform a dynamic constraint in one specification into a static constraint in another specification.

Therefore, we can use our methodology to transform DICs into SICs. Actually, the work in Chomicki (1995) has to do with *checking* DICs statically. In our case, with our reformulation of Chomicki's methodology in terms of a specification of the dynamics of the history encoding relations, we can rewrite DICs as SC sentences expressing SICs, which can be *proven* as such from the (extended) specification of the database dynamics. In particular, we can use theorem proving techniques for proving SICs by automated induction, like the ones presented in Bertossi et al. (1996), in order to automatically prove DICs from the specification of the database dynamics. The following proposition formalizes the idea showed above.

Proposition 2. *Given a SC specification* Σ. *If* Σ_H *is constructed from* Σ *as shown in Section* 3.2, *then*

$$\Sigma \models \forall s(S_0 \leq s \supset \texttt{holds}(\varphi, s)) \quad \text{iff } \Sigma_H \models \forall s(S_0 \leq s \supset \texttt{H}(\varphi, s)).$$

Example 9. Let Σ be the specification in Example 1 and ψ be the DIC (3) expressing that an employee's salary cannot decrease, that must hold in every legal current state s of the DB. That is, as a sentence, it must follow from Σ.

We can express this integrity constraint in our extended formalism as follows:

$$\begin{aligned}&\forall s(S_0 \leq s \supset \\ &\quad \texttt{holds}(\forall(e, p', p'')((\diamond \mathit{Salary}(e, p') \wedge \mathit{Salary}(e, p'')) \supset p' \leq p'') \wedge \\ &\qquad \Box((\diamond \mathit{Salary}(e, p') \wedge \mathit{Salary}(e, p'')) \supset p' \leq p''), s))\end{aligned} \tag{6}$$

If $\alpha(e, p')$ is $\diamond \mathit{Salary}(e, p')$, we create a table R_α with specification:

$$\begin{aligned}&\forall(e, p')\neg R_\alpha(e, p', S_0), \\ &\forall(a, s)\mathit{Poss}(a, s) \supset \forall(e, p')(R_\alpha(e, p', \mathit{do}(a, s)) \equiv \\ &\qquad R_\alpha(e, p', s) \vee \mathit{Salary}(e, p', s)).\end{aligned} \tag{7}$$

Introducing R_α in the first argument of `holds` in (6) we obtain:

$$\forall(e, p', p'')(((R_\alpha(e, p') \wedge \mathit{Salary}(e, p'')) \supset p' \leq p'') \wedge \\ \Box((R_\alpha(e, p') \wedge \mathit{Salary}(e, p'')) \supset p' \leq p'')). \tag{8}$$

If $\beta(e, p', p'')$ is the subformula $\Box((R_\alpha(e, p') \wedge \mathit{Salary}(e, p'')) \supset p' \leq p'')$, we create a table R_β with specification

$$\forall(e, p', p'')R_\beta(e, p', p'', S_0),$$
$$\forall(a, s)\mathit{Poss}(a, s) \supset \forall(e, p', p'')(R_\beta(e, p', p'', do(a, s)) \equiv \\ R_\beta(e, p', p'', s) \wedge ((R_\alpha(e, p', s) \wedge \mathit{Salary}(e, p'', s)) \supset p' \leq p'')). \tag{9}$$

Introducing R_β in (8), we obtain

$$\forall(e, p', p'')(((R_\alpha(e, p', s) \wedge \mathit{Salary}(e, p'', s)) \supset p' \leq p'') \wedge R_\beta(e, p, p''s)).$$

Thus, the original DIC holds in every state if and only if the specification Σ_H, consisting of Σ plus (7) and (9), entails the SIC:

$$\forall s(S_0 \leq s \supset \forall(e, p', p'') \\ (((R_\alpha(e, p', s) \wedge \mathit{Salary}(e, p'', s)) \supset p' \leq p'') \wedge R_\beta(e, p', p'', s))). \tag{10}$$

The IC (10) can be split into the two *binary* static integrity constraints $\forall s(S_0 \leq s \supset \forall(e, p', p'')((\mathit{Salary}(e, p'', s) \wedge \neg p' \leq p'') \supset \neg R_\alpha(e, p', s)))$ and $\forall s(S_0 \leq s \supset \forall(e, p', p'')$ $R_\beta(e, p', p'', s))$. As shown in Pinto (1994) and Bertossi et al. (1998), these constraints can be compiled into the specification of the extended database dynamics, in this case, modifying the original SSAs for the new tables R_α and R_β.

5. History dependent transactions

As we saw in Section 2, the formalism for specifying DB updates contains preconditions for action executions that depend on the current state of the database, only. Many concepts and algorithms that have originated from this formalism are based on this kind of local action precondition axioms (APAs). Nevertheless, there are natural scenarios in which the conditions for executing an action should depend on a longer history of the database. For example, in a voters database we might have the following APA for action *vote*

$$\forall(x, y, s)(\mathit{Poss}(\mathit{vote}(x, y), s) \equiv \exists n(\mathit{Age}(x, n, s) \wedge n \geq 18) \wedge \\ \mathit{Candidate}(y, s) \wedge \forall s'(S_0 \leq s' \leq s \supset \neg \mathit{InJail}(x, s'))). \tag{11}$$

That is, x can vote for y as long as y is a candidate, x is not younger than 18, and x has never been in jail. This is a history dependent transaction.

We can use the macros `holds` to represent this kind of actions. In fact, we can extend the definition of action preconditions as follows:

$$\forall(\overline{x}, s)(Poss(A(\overline{x}), s) \equiv \texttt{holds}(\psi(\overline{x}), s)). \tag{12}$$

If ψ includes some of the operators $\bullet$, **since**, $\Diamond$ or $\Box$, we have a history dependent action.

We can use the machinery developed so far for transforming history dependent transactions into local transactions. To do this we only need to construct a new specification Σ'_H from Σ_H, in its turn obtained from $\texttt{H}(\psi(\overline{x}), s)$ as before, but with the original APA (12) replaced by the new APA:

$$\forall(\overline{x}, s)(Poss(A(\overline{x}), s) \equiv \texttt{H}(\psi(\overline{x}), s)),$$

which is of the form (1). As before, the new specification contains SSAs for the auxiliary tables introduced by the construction of $\texttt{H}(\psi(\overline{x}), s)$.

Proposition 3. *Let Σ be a SC specification containing a history dependent APA for action A and let Σ_H be the new SC specification containing SSAs for the auxiliary relations and the old APA replaced by the new, local one. If $Poss_H$ and $\leq_H$ are the possibility predicate and accessibility relation defined on the basis of the new APA, then it holds*:
(a) *For every ground state term S, and ground action term of the form $A(\overline{c})$,*

$$\Sigma \models S_0 \leq S \supset Poss(A(\overline{c}), S) \quad \text{iff } \Sigma_H \models S_0 \leq_H S \supset Poss_H(A(\overline{c}), S).$$

(b) *For every ground state term S,*

$$\Sigma \models S_0 \leq S \quad \text{iff } \Sigma_H \models S_0 \leq_H S. \qquad \Box$$

The proposition says that at every accessible state, action A is possible in the old sense if and only if it is possible in the new sense and that both specifications define the same accessible states.

Example 10. We can apply the methodology to the voters example. In (11), the original APA for $vote(x, y)$ can be expressed by means of the macros `holds` as follows:

$$\begin{aligned}\forall(x, y, s)(Poss(vote(x, y), s) \equiv \texttt{holds}(&\exists n(Age(x, n) \wedge n \geq 18) \wedge \\ &Candidate(y) \wedge \neg InJail(x) \wedge \Box\neg InJail(x), s))\end{aligned} \tag{13}$$

In consequence, we generate a new specification Σ'_H, extending Σ, that includes now:

1. A new table $R_\alpha(x, s)$ that contains, at state s, the people x that have not been in jail before state s, whose specification consists of

$$\forall x\, R_\alpha(x, S_0),$$
$$\forall(a, s)Poss(a, s) \supset \forall x(R_\alpha(x, do(a, s)) \equiv R_\alpha(x, s) \wedge \neg InJail(x, s)).$$

2. The original APA for action A replaced by

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

6. Possible extensions

6.1. *Explicit time*

In Chomicki (1995), explicit time was also included to extend history encoding and deal with real-time issues. In Arenas et al. (1998d) it is shown how to extend our methodology by introducing explicit time in the database. This can be done by considering time as a new parameter for actions (Reiter, 1996). In this way, situations will have their associated times. Here we given only some hints on how explicit time can be accommodated into our framework.

As before, primitive actions will be denoted with function symbols, but now with one extra parameter of a new, temporal sort $\mathcal{T}$. Thus, a term $borrow(book, t_1)$ would denote the instantaneous action of borrowing a book at time t_1. We also include a new function *time* from actions to $\mathcal{T}$, such that, for each action function $A(x, t)$, *time* satisfies the axiom $time(A(x, t)) = t$. Therefore, if $borrow(book, t_1)$ is an action term, then we would have $time(borrow(book, t_1)) = t_1$. We also need a function, *start*, from situations to times, such that $start(s)$ is the starting time for situation s. Since actions are instantaneous, we require that $start(do(a, s)) = time(a)$ (Reiter, 1998). Nevertheless, situations may have a duration (Baier and Pinto, 1998).

The specification of the accessibility relation between situations, $<$, has to be modified by the axiom

$$s_1 < do(a, s_2) \equiv Poss(a, s_2) \wedge s_1 \leq s_2 \wedge start(s_2) \leq time(a).$$

According to this characterization, $s_1 < s_2$ is true if all the actions that lead from s_1 to s_2 are possible in the intermediate situations where they are performed, and their times are in the right order.

We also have to modify the unique names axioms as follows:

$$a(x_1, \ldots, x_n, t) \neq a'(y_1, \ldots, y_m, t').$$
$$a(x_1, \ldots, x_n, t) = a(y_1, \ldots, y_m, t') \supset x_1 = y_1 \wedge \cdots \wedge x_n = y_n \wedge t = t'.$$

In the temporal extension of the situation calculus we will find temporal atomic formulas of the form $t_1 \sim t_2$, where t_1 and t_2 are temporal terms and $\sim \in \{=, \neq, <, >, \leq, \geq\}$. We also have more complex temporal formulas of the form $\diamond_{\sim c}\psi$, where c is a nonnegative integer, stating that formula ψ is true in some previous time, and the previous and present times are subject to the constraint $\sim c$. For example, if the time unit is a day, then $\diamond_{=5}\neg Emp(sue)$ states that five days ago *sue* was not an employee of the company. More precisely, we define:

$$\texttt{holds}(\diamond_{\sim c}\psi, s) := \exists s'(S_0 \leq s' < s \wedge (start(s) - start(s')) \sim c \wedge \texttt{holds}(\psi, s')).$$

For example, since $\texttt{holds}(\mathit{Emp}(\mathit{sue}), s) = \mathit{Emp}(\mathit{sue}, s)$, we have

$$\texttt{holds}(\Diamond_{=5}\neg \mathit{Emp}(\mathit{sue}), s) := \exists s'(S_0 \leq s' < s \wedge (\mathit{start}(s) - \mathit{start}(s')) = 5 \wedge \neg \mathit{Emp}(\mathit{sue}, s')).$$

For a temporal formula $\varphi(\bar{x})$ of the form $\Diamond_{\sim c}\psi(\bar{x})$, where ψ is a non-temporal formula, the formula $\texttt{H}(\varphi(\bar{x}), S)$ (see Section 3.2) is $\exists t(R_\varphi(\bar{x}, t, S) \wedge t \sim c)$, where R_φ is new table. We use the temporal parameter t in R_φ to store the amount of time that has elapsed since $\psi(\bar{x})$ was true. Thus, if $\psi(\bar{x})$ is true at s, then at state $\mathit{do}(a, s)$, in t we store $\mathit{time}(a) - \mathit{start}(s)$. Moreover, if $R_\varphi(\bar{x}, t', s)$ is true, and therefore $\psi(\bar{x})$ was true t' units of time ago at some state previous to s, at state $\mathit{do}(a, s)$, in t we store $t' + \mathit{time}(a) - \mathit{start}(s)$.[10] In consequence, $R_\varphi(\bar{x}, t, s)$ has the following successor state axiom:

$$\forall(a, s)\mathit{Poss}(a, s) \supset \forall(\bar{x}, t)(R_\varphi(\bar{x}, t, \mathit{do}(a, s)) \equiv (\texttt{H}(\psi(\bar{x}), s) \wedge t = (\mathit{time}(a) - \mathit{start}(s))) \vee \exists t'(R_\varphi(\bar{x}, t', s) \wedge t = t' + \mathit{time}(a) - \mathit{start}(s))).$$

At the initial state we define R_φ by $\forall(\bar{x}, t)\neg R_\varphi(\bar{x}, t, S_0)$.

With this extension to explicit time, now it is possible to express a metric temporal precondition for the action *fire* saying that it is possible to fire an employee if he/she has been working at least 30 days in the company:

$$\forall(x, s)(\mathit{Poss}(\mathit{fire}(x), s) \equiv \texttt{holds}(\neg\Diamond_{<30}\neg \mathit{Emp}(x), s)).$$

As before, this precondition can be changed by a new one referring to the execution state only: if $\varphi(x)$ is $\Diamond_{<30}\neg \mathit{Emp}(x)$, then

$$\forall(x, s)(\mathit{Poss}(\mathit{fire}(x), s) \equiv \neg\exists t(R_\varphi(x, t, s) \wedge t < 30)),$$

where the new table R_φ is defined by

$$\forall(a, s)\mathit{Poss}(a, s) \supset \forall(x, t)(R_\varphi(\bar{x}, t, \mathit{do}(a, s)) \equiv (\neg \mathit{Emp}(x), s) \wedge t = (\mathit{time}(a) - \mathit{start}(s))) \vee \exists t'(R_\varphi(x, t', s) \wedge t = t' + \mathit{time}(a) - \mathit{start}(s))).$$

6.2. Open queries

In this paper, we have just considered queries that are sentences, that is without free variables. We think this kind of queries is more likely to occur in hypothetical reasoning, in the sense that they deal with global properties of an hypothetical state of the world. Nevertheless, our methodology can be easily applied to open queries that should return database tuples as answers. Those tuples can be retrieved from the domain elements appearing in the transaction log and from the initial database. This can be done, again, by means of the regression operator that is able to handle free variables. The formulas resulting in the process can be easily simplified taking advantage of the unique names axioms before the final evaluation process (cf. Bertossi et al., 1998 and Appendix A).

6.3. Regular expression queries

In this paper we have considered a wide, but restricted class of historical queries, namely those that can be constructed on the basis of the usual first order past temporal logic operators. In Abiteboul et al. (1996), predicate calculus and extended temporal logic (ETL) were compared in terms of expressive power for temporal databases. ETL is a proper extension of first order past temporal logic by means of formulas that are constructed on the basis of regular expressions. History encoding and our methodology can be extended to include regular expression queries on finite transaction logs.

7. Discussion and conclusions

Among the contributions in this paper we find the following: (1) An embedding and representation of the operators of first order past temporal logic in the situation calculus. (2) An extension of methodology presented in Chomicki (1995) to check dynamic integrity constraints to the case in which there is a specification of the evolution of the database. (3) A methodology for doing hypothetical reasoning along a virtual evolution of the database obtained by the execution of domain specific primitive transactions, whereas (Chomicki, 1995) concentrates on fixed integrity constraints and physical and usual updates of the database. (4) A general solution to the problem of answering temporal queries in the context of Reiter's specifications of database updates, and this solution works both in a progressive as in a regressive way. (5) A general transformation mechanism of dynamic integrity constraints into static integrity constraints, in a context like Reiter's, where both kind of constraints are expected to be logical consequences of the specification. (6) A general mechanism for transforming history dependent preconditions for action executions into preconditions to be evaluated at the execution state. (7) An implementation of all these methodologies. (8) An extension of all the previous results and techniques to the case of explicit time or metric temporal logic.

Preliminary versions of this work can be found in Arenas and Bertossi (1998a, 1998b). In the first case, no temporal operators were considered, and temporal queries were much more complex than here. In the second case, explicit use of first order past temporal logic and translations between it and the SC was made. The current version combines the best of the two approaches.

We think that the methodologies developed here for relational databases could be applied in other scenarios as well, e.g. (1) Hypothetical reasoning in workflows (Bonner, 1999; Davulcu, 1998; Trajcevski et al., 2000), (2) Reasoning from and about policies (Chomicki et al., 2000), and (3) Temporal reasoning in AI in general (Chittaro and Montanari, 2000).

7.1. Complexity issues

As already described, in Chomicki (1995) a procedure for checking temporal integrity constraints is presented. It is based on the idea of history encoding, that we have captured and used in the situation calculus framework. The history encoding methodology presented in Chomicki (1995) turns out to be polynomially bounded in the sense that given an integrity

constraint to be checked, the number of auxiliary relations to be introduced does not depend on the length n of the transaction log, and the amount of historical information stored in the union of the auxiliary relations does not depend on n, but is bounded by a polynomial evaluated on the amount of data in the initial database plus the domain values appearing in the transaction log (the history) and the constraint.

In our SC context, we may have domain defined primitive actions, possibly appearing in transaction logs, that could have the instantaneous effect of inserting a huge amount of data in several database relations at the same time. If we restrict ourselves to most common situation in which those primitive actions are of the form $insert_P(\bar{x})$, $\bar{x}$ into table P and $delete_P(\bar{x})$, that inserts $\bar{x}$ from table P, respectively, then we still have polynomially bounded history encoding in the sense that the number of auxiliary relations depends on the temporal query $\mathrm{holds}\,(\varphi, S_n)$ and not on the length n of the transaction log $A_1, \ldots, A_n$ (producing states $S_1, \ldots, S_n$), and the amount of data stored in the union of them is bounded by a polynomial on the number of domain values appearing in the initial relational database Σ_0 plus $A_1, \ldots, A_n$ and φ.[11]

This situation and analysis is relevant for efficiently answering historical queries by means of a physical progression of the database, in particular, through materialization of the auxiliary views (see Section 3.2). Nevertheless, if we want to answers the query by appealing to a solution of the temporal projection problem via query regression (see Section 3.2), and thus avoiding the materalization of the historical views, we have to analyze things in a different way.

In the regression based approach, the number of auxiliary historical relations does not depend on the length of the transaction log, but on the original query only. The amount of data potentially stored in those relations is no longer relevant since they are not materialized. The problem is that in a general situation, the regression of a given formula may become exponentially long wrt the original query (Reiter, 2001), and thus leading to exponential time to evaluate the regressed query. Nevertheless, there are situations in which this evaluation can be done in polynomial time. This is the case of *context-free* successor state axioms (Reiter, 2001). Those are axioms in which there are no conditions on the database at the execution state for the actions to have their desired *effects* (there might be preconditions for their *executions*, but they do not appear explicitly in the SSAs). This is a common situation in databases, in particular, when only actions of the form $insert_P(\bar{x})$ and $delete_P(\bar{x})$ are considered.

Our running example (Example 1) contains context-free SSAs only. For example, the SSA for *Emp* is

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

Here, action $a = hire(x)$ will have the effect of having x inserted into *Emp* at the successor state, without any condition. Nevertheless, this action does have a precondition to be executed, namely:

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

Notice that the query to be regressed will contain some auxiliary relations, which have derived SSAs that will be used by the regression operator. It is an interesting subject for further investigation to find conditions under which the derived SSAs will be context-free when the SSAs for the original database are context-free. For this purpose, some syntactic techniques introduced in Arenas and Bertossi (1998c) could be useful. In general, the complexity of regression based query answering in knowledge representation and databases requires further research.

7.2. Related work

There has been some work done on hypothetical reasoning in databases. In Bonner (1990), a datalog language, Hypothetical Datalog, that allows both database specification and hypothetical reasoning, is presented. There, special rules for hypothetical reasoning are included in the datalog specification. These rules contain predicates with a sort of annotations indicating that the predicate will be true if certain tuple is deleted or inserted into the database. Then, the kind of reasoning they allow is of the form "Would that predicate be true if these properties are added/included?." The final virtual state is evaluated. It is also possible to specify a sort of hypothetical predicates, whose truth depends on the execution of future add/delete transactions. Hypothetical Datalog also benefits from the possibility of specifying recursive predicates. The queries than can be posed at the hypothetical state are limited by the datalog formalism. In Bonner (1990) other related formalisms for hypothetical reasoning in databases are discussed.

In Chen (1997) a language that can be considered as an extension of Hypothetical Datalog is presented. More complex database updates are integrated into the language, more precisely, as predicates in the rules. In addition, hypothetical reasoning with respect to a particular predicate, L, can be done by evaluating an auxiliary predicate, $\diamond L$. This predicate is evaluated as L, except for the fact that the updates that appear in the rules that have to do with L are not committed. After evaluation, the database goes back to the current physical state.

The more interesting and more powerful logic programming formalism than the two above for specifying and executing updates is Transaction Logic Programming (Bonner and Kifer, 1998). It also allows doing hypothetical reasoning. As discussed in Bertossi et al. (1998), Transaction Logic and the Situation Calculus can complement each other well. In the situation calculus it is possible to specify primitive transactions, in particular, giving an account of the frame problem for this kind of transactions; and transaction logic can be used to specify more complex transactions.

In our case, instead, we restrict ourselves to relational databases (as opposed to deductive databases), but we have arbitrary domain specific transactions which are specified in the SC formalism. The hypothetical situation is created by virtually executing explicit transactions of this kind. Actually, Reiter's formalism is a language for reasoning about action executions (but not for executing actions); this makes it perfect for hypothetical reasoning.

We are in position to query the whole database evolution, with involved relationships between the generated states; for this we take advantage of the existence of explicit states in our formalism and quantifications over them. Our query language, both for trajectories

and particular states is much more expressive. We do not need extra rules or auxiliary predicates for doing hypothetical reasoning; we just use the specification of our database, in particular, we will not get to the situation of having to introduce new rules and predicates in the program which we might have initially forgotten.

With the formalism we have, as it is, we do not have the possibility of introducing recursive predicates. Nevertheless, recursion could be introduced as macros as done in Golog (Levesque et al., 1997), a situation calculus based logic programming language for high level robot programming. Introducing predicates whose truth depends on future states does not seem to be a problem in our formalism, because quantifications on states and the *do* operator should allow doing this.

The problem of answering hypothetical historical queries in the context of the situation calculus specifications of database updates was posed in Reiter (1995), where also some particular cases were treated. Further work on this problem is presented in Reiter (2001), but essentially the case of *context-free* successor state axioms is considered. Our methodology, being much more general, provides the same formulas to be evaluated against the initial database in those cases, but can be applied to a wider range of queries.

Appendix A: Implementation

We have added to the automated reasoner, SCDBR, the functionality of generating the new SC formula and specification, including the application of the regression operator (Bertossi et al., 1998). The reasoner is implemented in PROLOG.

Example 11. Let us consider the formula

$$\texttt{holds}(\Diamond p(v_1)\ \textbf{since}\ \Box q(v_1), s). \tag{14}$$

To evaluate it, we need to consider $\mathrm{H}(\Diamond p(v_1)\ \textbf{since}\ \Box q(v_1), s)$ in an extended specification Σ_H. To do this, we will use some of the procedures of SCDBR.

```
|?- i_p(diamond p(v1) since box q(v1), F),
    tl_initial(F, s, I1, [F2,F3,F4], _, [F5,F6,F7]),
    p_i([F2,F3,F4,F5,F6,F7], [I2,I3,I4,I5,I6,I7]).
I1= r_0(v1,s),
I2= forall(v1):(neg r_0(v1,s0)),
I3= forall(v1):(neg r_0_sl(v1,s0)),
I4= forall(v1):(neg r_0_sr(v1,s0)),
I5= forall(a):(poss(a,s) => (r_0(v1,do(a,s)) <=>
                          (r_0_sl(v1,s) v p(v1,s)) &
                          (r_0(v1,s) v neg r_0_sr(v1,s)))),
I6= forall(a):(poss(a,s) => (r_0_sl(v1,do(a,s)) <=>
                                r_0_sl(v1,s) v p(v1,s))),
I7= forall(a):(poss(a,s) => (r_0_sr(v1,do(a,s)) <=>
                             r_0_sr(v1,s) v neg q(v1,s)))
```

Here, we first introduced (14) by means of the predicate `i_p`. This predicate stores it in `F` as $\diamond p(v_1)$ **since** $\neg\diamond\neg q(v_1)$, that corresponds to its internal representation. Next, the predicate `tl_initial` was applied on `F`, obtaining: (a) `H`$(\diamond p(v_1)$ **since** $\neg\diamond\neg q(v_1), s)$, which is stored in `I1`; (b) the list of the initial state formulas for the generated auxiliary tables, which are stored in `F2`, `F3` and `F4`; (c) the list of the successor state axioms for the generated auxiliary tables, which are stored in `F5`, `F6` and `F7`. Finally, the predicate `p_i` was applied for translating (b) and (c), written in the internal representation, into `I2, I3, I4, I5, I6` and `I7`, written in infix notation.

From the SCDBR's invocation, we can see that $r\_0(v_1, s)$ is equal to `H`$(\diamond p(v_1)$ **since** $\neg\diamond\neg q(v_1), s)$. For producing this formula, the system generated three auxiliary tables r_0, r_0_sl and r_0_sr, with the following initial state definitions:

$$\forall v_1(\neg r\_0(v_1, S_0)), \forall v_1(\neg r\_0\_sl(v_1, S_0)), \forall v_1(\neg r\_0\_sr(v_1, S_0));$$

and with the following SSAs axioms:

$$\begin{aligned}
&Poss(a, s) \supset (r_0(v_1, do(a, s)) \equiv \\
&\qquad (r_0_sl(v_1, s) \vee p(v_1, s)) \wedge (r_0(v_1, s) \vee \neg r_0_sr(v_1, s))),\\
&Poss(a, s) \supset (r_0_sl(v_1, do(a, s)) \equiv r_0_sl(v_1, s) \vee p(v_1, s)),\\
&Poss(a, s) \supset (r_0_sr(v_1, do(a, s)) \equiv r_0_sr(v_1, s) \vee \neg q(v_1, s)).
\end{aligned}$$

If we run the regression procedure in SCDBR to solve the temporal projection problem, we will eventually obtain a query to be posed to the initial database. SCDBR can answer this query by calling a conventional DBMS (actually, ORACLE) or a PROLOG program or a theorem prover (OTTER), depending on the kind of initial, physical database available.

Example 12. Let us consider the specification shown in Example 1. We want to know the list of the employees who have always been working in the company, in all states generated by the execution of the sequence of actions $T = [hire(sue), fire(john)]$ from the initial situation. Thus, we are asking

$$\texttt{holds}(Emp(x) \wedge \Box Emp(x), do(T, S_0)). \tag{15}$$

We can answer this query by means of the following invocation of SCDBR

```
|?- i_p(emp(x) & box emp(x),F),tl_initial(F,do(fire(john),
    do(hire(sue),s0)), F1, [F2], I, [F3]), p_i(F1, F6),
    p_i(F2, F7), p_i(F3, F8), reg_n(F1, 2, F4),
    prune_una(F4, F5), p_i(F5, F9), prolog_initial(F5, I).

Transforming Query into Prolog Goal...Done.

[[x,ernest]]
[[x,page]]

F6 = emp(x,do(fire(john),do(hire(sue),s0))) &
             neg r_0_r(x,do(fire(john),do(hire(sue),s0))),
```

```
F7 = forall(x):(neg r_0_r(x,s0)),
F8 = forall(x):(emp(x,s0) & neg r_0_r(x,do(a,s)) <=>
                                  r_0_r(x,s) v neg emp(x,s))),
F9 = ((sue e
             & neg((r_0_r(x, s0) v neg emp(x, s0)) v
                                  neg (sue e
```

As in Example 11, we use the predicate `tl_initial` for generating H($Emp(x) \wedge \Box Emp(x)$, $do(T, S_0)$). More precisely, the system generates H($Emp(x) \wedge \neg\Diamond\neg Emp(x)$, $do(T, S_0)$) in `F6`:

$$Emp(x, do(\mathit{fire}(\mathit{john}), do(\mathit{hire}(\mathit{sue}), S_0))) \wedge \neg r\_0\_r(x, do(\mathit{fire}(\mathit{john}), do(\mathit{hire}(\mathit{sue}), S_0))), \tag{16}$$

which includes the auxiliary table r_0_r. For this table, the system also generates the following formula describing its initial state

$$\forall x(\neg r\_0\_r(x, S_0)), \tag{17}$$

and the following SSA

$$Poss(a, s) \supset (r\_0\_r(x, do(a, s)) \equiv (r\_0\_r(x, s) \vee \neg Emp(x, s))). \tag{18}$$

In consequence, for answering (15) and (16) needs to be answered with respect to the company database plus the information in (17) and (18).

As we see in the invocation, the procedure `reg_n` is used for applying twice the regression operator on (16). After that, by means of the predicate `prune_una`, the resulting formula is simplified on the basis of the unique name axioms for actions. So, the following formula `F9` is generated to be posed against the initial database state

$$((sue = x \vee Emp(x, S_0)) \wedge \neg john = x) \wedge \neg((r\_0\_r(x, S_0) \vee \neg Emp(x, S_0)) \vee \neg(sue = x \vee Emp(x, S_0))). \tag{19}$$

If the (initial) database of the company is a PROLOG database, we answer to (19) by means of the procedure `prolog_initial`, which uses PROLOG as query language. In this way, we obtain the following tuples as answer to the query:

$$x = ernest \vee x = page.$$

As described in Bertossi et al. (1998), SCDBR can be interfaced with a RDBMS.

Example 13. We have an ORACLE database with information about the employees in the company at the initial state. We execute the list of transactions [*fire*(*john*), *fire*(*ernest*)].

```
q x v emp(x,s0)))
```

Now we want to know all the employees who have worked for the company from the initial state on:

$$\texttt{holds}(Emp(x) \vee \Diamond Emp(x), do(T, S_0))$$

and has to be posed at the state $do(T, S_0)$.

We can ask SCDBR to generate a new query in SQL to be posed against the initial relational database:

```
|?- i_p(emp(x) v diamond emp(x), F), tl_initial(F,
    do(fire(ernest),do(hire(john),s0)), F1, [F2], I,[F3]),
    p_i(F1, F6), p_i(F2, F7), p_i(F3, F8), reg_n(F1,2,F4),
    prune_una(F4,F5),p_i(F5,F9),ora_sql(F5).

[[page],[john],[ernest]]
```

Here `F5` stands for a first order query that the predicate `ora_sql` transforms, via a lower level predicate, into a SQL query to be posed to the ORACLE database. After that, `ora_sql` prints the answer. In this case, we obtain: $x = page \vee x = john \vee x = ernest$.

Appendix B: Proofs and intermediate results

Lemma 1. *Let $\varphi(\bar{x})$ and $\psi(\bar{x})$ be formulas that do not include operators $\bullet$,* **since**, *$\Diamond$ and $\Box$, and Σ_H a* SC *specification constructed from a* SC *specification Σ as was showed in Section* 3.2. *Then*

$$\Sigma_H \models \forall s(S_0 \leq s \supset \forall \bar{x}(\texttt{holds}(\bullet\varphi(\bar{x}), s) \equiv \texttt{H}(\bullet\varphi(\bar{x}), s)))$$
$$\Sigma_H \models \forall s(S_0 \leq s \supset \forall \bar{x}(\texttt{holds}(\Diamond\varphi(\bar{x}), s) \equiv \texttt{H}(\Diamond\varphi(\bar{x}), s)))$$
$$\Sigma_H \models \forall s(S_0 \leq s \supset \forall \bar{x}(\texttt{holds}(\Box\varphi(\bar{x}), s) \equiv \texttt{H}(\Box\varphi(\bar{x}), s)))$$
$$\Sigma_H \models \forall s(S_0 \leq s \supset \forall \bar{x}(\texttt{holds}(\varphi(\bar{x}) \textbf{ since } \psi(\bar{x}), s) \equiv \texttt{H}(\varphi(\bar{x}) \textbf{ since } \psi(\bar{x}), s)))$$

Proof: It is necessary to prove this lemma for operators $\bullet$ and **since.** We are going to do this by induction on states. Firstly, we consider the operator $\bullet$. If $\alpha(\bar{x}) = \bullet\varphi(\bar{x})$, then:

$$\texttt{holds}(\alpha(\bar{x}), s) = \exists(a, s')(s = do(a, s') \wedge \texttt{holds}(\varphi(\bar{x}), s')),$$
$$\texttt{H}(\alpha(\bar{x}), s) = R_\alpha(\bar{x}, s)$$

where R_α is defined as follows:

$$\forall \bar{x} \neg R_\alpha(\bar{x}, S_0)$$
$$\forall(a, s)Poss(a, s) \supset \forall \bar{x}(R_\alpha(\bar{x}, do(a, s)) \equiv \texttt{H}(\varphi(\bar{x}), s))$$

In this case, we have that by unique name for states $\texttt{holds}(\alpha(\overline{x}), S_0)$ is false for every value of $\overline{x}$. Thus, by definition of R_α we conclude that:

$$\Sigma_H \models \forall \overline{x}(\texttt{holds}(\alpha(\overline{x}), S_0) \equiv \texttt{H}(\alpha(\overline{x}), S_0))).$$

Let us suppose that for a given state S, such that $S_0 \leq S$, we have that:

$$\Sigma_H \models \forall \overline{x}(\texttt{holds}(\alpha(\overline{x}), S) \equiv \texttt{H}(\alpha(\overline{x}), S))).$$

Let A a be ground action such that $Poss(A, S)$ is true in S. In this case, we know that $\texttt{holds}(\alpha(\overline{x}), do(A, S)) = \exists(a, s')(do(A, S) = do(a, s') \wedge \texttt{holds}(\varphi(\overline{x}), s'))$ and $\texttt{H}(\alpha(\overline{x}), do(A, S)) = R_\alpha(\overline{x}, do(A, S))$. Thus, by taking into account unique name for states and the SSA of R_α we conclude that:

$$\texttt{holds}(\alpha(\overline{x}), do(A, S)) \equiv \texttt{holds}(\varphi(\overline{x}), S),$$
$$\texttt{H}(\alpha(\overline{x}), do(A, S)) \equiv \texttt{H}(\varphi(\overline{x}), S).$$

Given that $\varphi(\overline{x})$ does not include operators $\bullet$, **since**, $\diamond$ and $\Box$, we have that $\texttt{holds}(\varphi(\overline{x}), s) = \texttt{H}(\varphi(\overline{x}), s)$. Therefore

$$\Sigma_H \models \forall \overline{x}(\texttt{holds}(\alpha(\overline{x}), do(A, S)) \equiv \texttt{H}(\alpha(\overline{x}), do(A, S)))).$$

Secondly, we are going to consider **since**. If $\alpha(\overline{x}) = \varphi(\overline{x})$ **since** $\psi(\overline{x})$, then

$$\begin{aligned} \texttt{holds}(\varphi(\overline{x}) \textbf{ since } \psi(\overline{x}), s) &= \exists s'(S_0 \leq s' < s \wedge \texttt{holds}(\psi(\overline{x}), s') \wedge \\ &\qquad \forall s''(s' < s'' \leq s \supset \texttt{holds}(\varphi(\overline{x}), s''))), \\ \texttt{H}(\varphi(\overline{x}) \textbf{ since } \psi(\overline{x}), s) &= R_\alpha(\overline{x}, s). \end{aligned}$$

where R_α is defined as follows:

$$\forall \overline{x} \neg R_\alpha(\overline{x}, S_0)$$
$$\forall(a, s) Poss(a, s) \supset \forall \overline{x}(R_\alpha(\overline{x}, do(a, s)) \equiv$$
$$\mathcal{R}[\texttt{H}(\varphi(\overline{x}), do(a, s))] \wedge (R_\alpha(\overline{x}, s) \vee \texttt{H}(\psi(\overline{x}), s))$$

Given that there is no state s' such that $s' < S_0$, $\texttt{holds}(\alpha(\overline{x}), S_0)$ is false for every value of $\overline{x}$. Thus, by definition of R_α we conclude that:

$$\Sigma_H \models \forall \overline{x}(\texttt{holds}(\alpha(\overline{x}), S_0) \equiv \texttt{H}(\alpha(\overline{x}), S_0)))$$

Let us suppose that for a given state S, such that $S_0 \leq S$, we have that:

$$\Sigma_H \models \forall \overline{x}(\texttt{holds}(\alpha(\overline{x}), S) \equiv \texttt{H}(\alpha(\overline{x}), S))). \tag{20}$$

Let A a ground action such that $Poss(A, S)$ is true in S. In this case, we know that:

$$\texttt{holds}(\varphi(\bar{x})\ \textbf{since}\ \psi(\bar{x}), do(A, S)) = \exists s'(S_0 \leq s' < do(A, S) \wedge \\ \texttt{holds}(\psi(\bar{x}), s') \wedge \forall s''(s' < s'' \leq do(A, S) \supset \texttt{holds}(\varphi(\bar{x}), s'))).$$

But, this formula is equivalent to

$$\exists s'(S_0 \leq s' < S \wedge \texttt{holds}(\psi(\bar{x}), s') \wedge \\ \forall s''(s' < s'' \leq do(A, S) \supset \texttt{holds}(\varphi(\bar{x}), s''))) \vee \\ (\texttt{holds}(\psi(\bar{x}), S) \wedge \texttt{holds}(\varphi(\bar{x}), do(A, S)))$$

which is equivalent to

$$\exists s'(S_0 \leq s' < S \wedge \texttt{holds}(\psi(\bar{x}), s') \wedge \\ \forall s''(s' < s'' \leq S \supset \texttt{holds}(\varphi(\bar{x}), s''))) \wedge \texttt{holds}(\varphi(\bar{x}), do(A, S)) \vee \\ (\texttt{holds}(\psi(\bar{x}), S) \wedge \texttt{holds}(\varphi(\bar{x}), do(A, S))),$$

Thus, by definition of holds we conclude that $\texttt{holds}(\alpha(\bar{x}), do(A, S))$ is equivalent to

$$(\texttt{holds}(\alpha(\bar{x}), S) \wedge \texttt{holds}(\varphi(\bar{x}), do(A, S))) \vee \\ (\texttt{holds}(\psi(\bar{x}), S) \wedge \texttt{holds}(\varphi(\bar{x}), do(A, S)))$$

But, by (20) we conclude that

$$\Sigma_H \models \forall \bar{x}(\texttt{holds}(\alpha(\bar{x}), do(A, S)) \equiv \\ \texttt{holds}(\varphi(\bar{x}), do(A, S)) \wedge (\texttt{H}(\alpha(\bar{x}), S) \vee \texttt{holds}(\psi(\bar{x}), S)).$$

Given that $\varphi(\bar{x})$ and $\psi(\bar{x})$ do not include operators $\bullet$, **since**, $\Diamond$ and $\Box$, we have that $\texttt{holds}(\varphi(\bar{x}), s) = \texttt{H}(\varphi(\bar{x}), s)$ and $\texttt{holds}(\psi(\bar{x}), s) = \texttt{H}(\psi(\bar{x}), s)$. Therefore

$$\Sigma_H \models \forall \bar{x}(\texttt{holds}(\alpha(\bar{x}), do(A, S)) \equiv \\ \texttt{H}(\varphi(\bar{x}), do(A, S)) \wedge (\texttt{H}(\alpha(\bar{x}), S) \vee \texttt{H}(\psi(\bar{x}), S)).$$

Finally, give that $Poss(A, S)$ is true in S, we have that

$$\Sigma_H \models \forall \bar{x}(\texttt{H}(\varphi(\bar{x})\ \textbf{since}\ \psi(\bar{x}), do(A, S)) \equiv \\ \mathcal{R}[\texttt{H}(\varphi(\bar{x}), do(A, S))] \wedge (R_\alpha(\bar{x}, S) \vee \texttt{H}(\psi(\bar{x}), S))),$$

and

$$\Sigma_H \models \forall \bar{x}(\texttt{H}(\varphi(\bar{x}), do(A, S)) \equiv \mathcal{R}[\texttt{H}(\varphi(\bar{x}), do(A, S))]).$$

Therefore

$$\Sigma_H \models \forall \bar{x}(\texttt{holds}(\alpha(\bar{x}), do(A, S)) \equiv \texttt{H}(\alpha(\bar{x}), do(A, S))).$$

□

Proof of Proposition 1: By Lemma (1) and taking into account that $\Sigma \models \texttt{holds}(\varphi, do(T, S_0))$ if and only if $\Sigma_H \models \texttt{holds}(\varphi, do(T, S_0))$. □

Acknowledgments

This research has been partially supported by FONDECYT Grants (# 1980945, #1990089, #1000593). Part of this work has been done during the second author's sabbatical at the TU Berlin. He is grateful to Ralf Kutsche and the CIS group for their support and hospitality; and to the GK "Distributed Information Systems," the DAAD and the DIPUC for their financial support.

Notes

1. In this paper we do not make any distinction between states and situations.
2. Sometimes, in the literature, they are called "historical queries," but this name may cause confusions with work done by the temporal databases community that calls "historical" the queries about valid time, rather than about transaction time (Snodgrass and Ahn, 1986), which can be better associated to the situations of the situation calculus.
3. They can be seen as usual database tables whose entries have an additional state stamp that is not stored in the extensional databases. In the knowledge representation literature they are usually called "fluents." States are used for distinguishing the successive snapshots of the database.
4. In this paper, a "database specification" always means a specification of the dynamics of a database.
5. Dynamic integrity constraints are usually of this form, but the results we will present in this paper still hold if we admit more involved quantifications on several states, related by the accessibility relation.
6. This is not a limitation if, as in Chomicki (1995), one is interested in some fixed dynamic integrity constraints only, but not in arbitrary temporal queries, as we are.
7. We should use new predicates, say *Emp* (., .) instead of *Emp* (., ., .), but there should be no confusion.
8. The last two operators can be defined in terms of the first two by $\Diamond\varphi := True$ **since** φ, and $\Box\varphi := \neg\Diamond\neg\varphi$.
9. We say that a sequence of actions $T = [A_1, \ldots, A_n]$ is legal if $Poss(A_1, S_0)$, $Poss(A_2, do(A_1, S_0)), \ldots, Poss(A_n, do([A_1, \ldots, A_{n-1}], S_0))$.
10. Notice that, for a fixed value of $\bar{x}$, $R_\varphi(\bar{x}, t, do(a, s))$ could be true for many values of t, if $\psi(\bar{x})$ was true many times in the past.
11. Our methodology allows us to have as Σ_0 something more general than a relational database, e.g. an initial database containing more complex first order formulas. Nevertheless, as shown in Lin et al. (1997) progressing the database of this case, in particular, the historical auxiliary views, might become a very complex process.

References

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

Arenas, M. and Bertossi, L. (1998). Hypothetical Temporal Reasoning with History Encoding. In G. De Giacomo and D. Nardi (Eds.), *Proc. ESSLLI-98 Workshop on Reasoning about Actions: Foundations and Applications* (pp. 1–15). Saarbruecken.

Arenas, M. and Bertossi, L. (1998). Hypothetical Temporal Queries in Databases. In A. Borgida, V. Chaudhuri, and M. Staudt (Eds.), *Proc. 5th International Workshop on Knowledge Representation meets Databases (KRDB'98): Innovative Application Programming and Query Interfaces* (pp. 4.1–4.8). http://sunsite.informatik.rwthaachen.de/Publications/CEUR-WS/Vol-10/. ACM SIGMOD/PODS 98, Seattle.

Arenas, M. and Bertossi, L. (1998). The Dynamics of Database Views. In H. Decker, B. Freitag, M. Kifer, and A. Voronkov (Eds.), *Transactions and Change in Logic Databases* (pp. 197–226). Springer, LNCS 1472.

Arenas, M., Bertossi, L., and Pinto, J. (1998). Representation of Temporal Knowledge in the Situation Calculus. Available at http://www.scs.carleton.ca/∼bertossi/papers/mt1.ps.

Baier, J. and Pinto, J. (1998). Non-Instantaneous Actions and Concurrency in the Situation Calculus (Extended Abstract). In G. De Giacomo and D. Nardi (Ed.), *Proc. ESSLLI-98 Workshop on Reasoning about Actions: Foundations and Applications*, Saarbruecken.

Bertossi, L., Arenas, M., and Ferretti, C. SCDBR. (1998). An Automated Reasoner for Specifications of Database Updates. *Journal of Intelligent Information Systems*, 10(3), 253–280.

Bertossi, L., Pinto, J., Saez, P., Kapur, D., and Subramaniam, M. (1996). Automating Proofs of Integrity Constraints in the Situation Calculus. In Z.W. Ras and M. Michalewiccz (Eds.), *Foundations of Intelligent Systems* (pp. 212–222). Lecture Notes Artificial Intelligence 1079, Springer.

Bonner, A. (1990). Hypothetical Datalog: Complexity and Expressivity. *Theoretical Computer Science*, 76(1), 3–51.

Bonner, A. (1999). Workflow, Transactions, and Datalog. In *Proc. ACM Symposium on Principles of Database Systems (PODS'99)* (pp. 294–305).

Bonner, A. and Kifer, M. (1998). Transaction Logic Programming. In J. Chomicki and G. Saake (Eds.), *Logics for Databases and Information Systems*, Dordrecht: Kluwer Academic Publishers.

Chaudhuri, S. and Dayal, U. (1997). An Overview of Datawarehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1), 65–74.

Chen, W. (1997). Programming with Logical Queries, Bulk Updates, and Hypothetical Reasoning. *IEEE Transactions on Data and Knowledge Engineering*, 9(4), 587–599.

Chittaro, L. and Montanari, A. (2000). Temporal Representation and Reasoning in Artificial Intelligence: Issues and Approaches. *Annals of Mathematics and Artificial Intelligence*, 28(1–4), 47–106.

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

Chomicki, J., Lobo, J., and Naqvi, S.A. (2000). A Logic Programming Approach to Conflict Resolution in Policy Management. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR'2000)*, Morgan Kaufmann (pp. 121–132).

Davulcu, H., Kifer, M., Ramakrishnan, C.R., and Ramakrishnan, I.V. (1998). Logic Based Modeling and Analysis of Workflows. In *Proc. ACM Symposium on Principles of Database Systems (PODS'98)* (pp. 25–33).

Gabbay, D., Hodkinson, I., and Reynolds, M. (1994). *Temporal Logic: Mathematical Foundations and Computational Aspects*. Vol. 1, Oxford University Press.

Hanks, S. and McDermott, D. (1986). Default Reasoning, Nonmontonic Logics, and the Frame Problem. In *Proc. National Conference on Artifical Intelligence* (pp. 328–333).

Levesque, H., Reiter, R., Lesperance, Y., Lin, F., and Scherl, S. (1997). GOLOG. A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming, Special Issue on Reasoning about Action and Change*, 31(1–3), 59–84.

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

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

McCarthy, J. and Hayes, P. (1969). Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie (Eds.), *Machine Intelligence*, 4, 463–502. Edinburgh University Press, Edinburgh, Scotland.

Pinto, J. (1894). Temporal Reasoning in the Situational Calculus. PhD Thesis, Department of Computer Science, University of Toronto, Canada.

Reiter, R. (1984). Towards a Logical Reconstruction of Relational Database Theory. In J. Mylopoulos and J. Schmidt (Eds.), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages* (pp. 191–233). Berlin: Springer-Verlag.

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

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

Reiter, R. (1996). Natural Actions, Concurrency and Continuous Time in the Situation Calculus. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96)*, San Mateo, CA: Morgan Kaufmann.

Reiter, R. (1998). Sequential, Temporal GOLOG. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)* (pp. 547–556). San Mateo, CA: Morgan Kaufmann.

Reiter, R. (2001). Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. Cambridge, MA: MIT Press.

Siu, B. and Bertossi, L. (1996). Answering Historical Queries in Databases (extended abstract). In M.V. Zelkowitz and P. Straub (Eds.), *Proc. XVI International Conference of the Chilean Computer Science Society (SCCC'96)* (pp. 56–66). Valdivia.

Snodgrass, R. and Ahn, I. (1986). Temporal Databases. *IEEE Computer*, 35–42.

Trajcevski, G., Baral, Ch., and Lobo, J. (2000). Formalizing (and Reasoning About) the Specifications of Workflows. In *Proc. CoopIS*, pp. 1–17.