A Semantical Notion of Relevance in Specifications of Database Updates and its Computational Counterpart

Bernardo Siu, Leopoldo Bertossi* and Marcelo Arenas
Pontificia Universidad Catolica de Chile
Escuela de Ingenieria
Depto. de Ciencia de Computacion
Casilla 306, Santiago 22, Chile.
E-mail: {bsiu,bertossi,marenas}@ing.puc.cl

Abstract

Using the situation calculus as underlying formalism for specifying database updates as proposed by Raymond Reiter, a semantical notion of relevance of entries in tables to other entries is introduced. Also a semantical notion of relevant action is provided. Some properties of these semantical notions are investigated. In particular, they are compared with syntactical definitions of relevance that can be used for answering historical queries in databases. Other uses of the semantical concepts of relevance and extensions are discussed.

1 Introduction

In this paper we present semantical notion of relevance in specifications of database updates written in the situation calculus [12] as proposed by Reiter in [15, 17]. We also recall a computational counterpart presented in [18]; and we compare the two approaches.

In this work we were motivated by the problem of answering historical queries in relational databases [18]: Given a initial (physical) relational database D_0 and a list of legal atomic transactions $\bar{T} = [a_1, \ldots, a_n]$, to be executed in sequence on D_0, and then producing the virtually updated databases $D_1, \ldots, D_n$, we want to check, for example, if a given formula holds in all (or some of) the states along $\bar{T}$. We want to make this checking without having to physically update the whole database after executing each transaction, keeping all versions, and inspecting (possibly) all of them. Rather, we would like to do this by a a combination of a kind of list processing on $\bar{T}$ and minimal update of the information in the database, which is represented in the *fluents* (think of the tables in the database).

In order to know which of the transactions to execute and the minimal information to be updated, we introduce the notions of relevant actions and relevant (instances of) fluents. An instance of a fluent (or an action) will be relevant to a query if the answer to the query depends potentially on that fluent (or that action). In [18], we presented algorithms that compute the fluents and actions that are relevant to a query and allow to optimize the process of answering it.

Notice that once the tuples and transactions that are relevant to a historical query have been determined, only the relevant actions appearing in the given transaction list $\bar{T}$ have to be executed

*Send official correspondence to this coauthor

and the database is updated accordingly, considering only the relevant tuples: To check the truth value of a given tuple $F(\bar{c}, \bar{x})$ along a given legal transaction list $\bar{T}$ only the (possibly few) actions in $\bar{T}$ that are relevant to $F(\bar{c}, \bar{x})$ must be executed, and only the instances of fluents that are relevant to $F(\bar{c}, \bar{x})$ must be updated.[1] The formulas appearing in the query can be evaluated against the arising states. In [18], a general format for posing historical queries was defined and the algorithms based on a *computational notion* of relevance were used for answering them. In this paper we justify the application of those algorithms, more precisely, we justify the (computational) notion of relevance, by comparing it with a *semantical notion* of relevance.

The algorithms for computing relevance have been implemented in SCDBR [3, 2], an automated system for reasoning from and about database specifications written in Reiter's formalism.

The rest of the paper is organized as follows: In section 2 we recall Reiter's approach to specifications of database updates. We also introduce some notation and conventions we will use in this paper. In section 3 we present semantical notions of relevance. In section 4 we present an algorithm that determines a (small) set of fluents and actions that are relevant to the query. In section 5 we discuss how these semantical and computational notions relate. In section 6, we discuss possible extensions of our work. Finally, in section 7 we draw some conclusions. The proofs of the results stated in the text appear in the appendix.

2 Reiter's Specification of Database Updates

The situation calculus models a dynamic world as evolving from state to state, starting from an initial state S_0. If s is a state term and a is an action term, then the term $do(a, s)$ denotes the state resulting from executing a on s, and is called a successor state of s. The world changes its state only by executing an action. The world is described in terms of dynamic properties, called *fluents* (think of the tables in a database). Formally, fluents are predicates with an extra state argument. Their evolution can be specified by means of *successor state axioms* (SSAs), which characterize when each fluent becomes true at a state of the form $do(a, s)$ in terms of information about state s [14].

In addition to the SSAs, the specification contains *action precondition axioms*, which are formulas of the form $Poss(A, s) \equiv \Pi_A[s]$, where $Poss(A, s)$ stands for action A being possible at state s. A finite sequence of transactions is *legal* iff each transaction in the list satisfies its precondition at its corresponding execution state. Reiter uses this idea to define the accessibility relation $<$ between states [15, 16]. Basically, $S_i < S_k$ holds iff state S_k can be reached from S_i by executing a legal transaction list.

Usually, for each fluent F there is a SSA of the form

$$Poss(a, s) \supset F(\bar{x})[do(a, s)] \equiv \gamma_F^+(\bar{x}, a)[s] \vee (F(\bar{x})[s] \wedge \neg\, \gamma_F^-(x, a)[s]), \tag{1}$$

where the formulas γ_F^+, γ_F^- are the assumed to be the only conditions under which $F(\bar{x})$ can change its truth value, becoming true or false, respectively, in the state $do(a, s)$. They are obtained from positive effect axioms and negative effect axioms for fluent F, respectively [14]. Formula (1) can be read as: "*The fluent $F(\bar{x})$ holds in state $do(a, s)$ iff the conditions that make it true were satisfied in state s or $F(\bar{x})$ held in state s and the conditions that make it false were not satisfied*". Notice

[1] In the situation calculus, fluents are situation dependent predicates, as such, they have exactly one situation argument, e.g. $F(\bar{x}, s)$. We will usually write this in the form $F(\bar{x})[s]$. This will allow us, for technical reasons, to suppress the situation argument, by simply writing $F(\bar{x})$, what makes the fluent situation independent. *Tuples* are situation independent instances of fluents. They can be seen as the entries in the tables, without the corresponding situation being represented in the table.

that if neither γ_F^+ nor γ_F^- hold, then the truth value of $F(\bar{x})$ is the same in s and $do(a, s)$. When the conditions γ_F^+, γ_F^- are completely known, Reiter [14] proposes to generate this kind of SSAs. In this way he solves the *frame problem*, that is, the problem of specifying in a compact, succinct, way when a fluent does not change when an action is performed [12].

The formulas on the right–hand side of the SSAs are *simple* in s [14], that is, s appears free in them and no other state term is mentioned (but a tacit universal quantification on a and s is assumed in (1)) .

According to [15, 17], a database is specified by a theory of the form $\Sigma = \Sigma_{bd} \cup \Sigma_{ap} \cup \Sigma_{una} \cup \Sigma_{ssa} \cup \Sigma_{S_0}$, where:

1. Σ_{bd} contains (a) state independent axioms for the domain of discourse and (b) the foundational axioms [10, 13] of situation calculus, which describe the ontology of the situations and include an induction axiom over states.
2. Σ_{ap} contains, for each named action A, a *precondition axiom* of the form $(\forall\, s) Poss(A, s) \equiv \Pi_A[s]$, stating when action A is possible. The formula $\Pi_A[s]$ is simple in state s and specifies the conditions that make A possible in state s.
3. Σ_{una} contains unique name axioms for actions and states.
4. Σ_{ssa} contains the SSAs: For each primitive table (fluent) F, there is one SSA of the form

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

where Φ is a formula simple in s with its free variables among $\bar{x}, a, s$. One important and usual kind of SSAs is (1) (see section 5.3).

Sometimes we will refer to formulas $\Phi_{F(\bar{c})}$, meaning that all variables in formula Φ_F must be renamed or instantiated accordingly. For example, if Φ_F is $F(x, y)[s] \vee (\exists\, x')R(x')$, then the formula $\Phi_{F(a,b)}$ will be $F(a, b)[s] \vee (\exists\, x')R(x')$.

5. Σ_{S_0} is the initial database, a theory that describes the world in the initial state S_0. The initial database can be relational and the *closed world assumption* can be made. With this assumption and the SSAs, we will have complete knowledge in future, legal, states of the database.

3 Semantical Notions of Relevance

In this section we present semantical notions of relevance for instances of fluents and actions, defined in terms of models of an action theory. They are interesting in their own, since they attempt to formalize the intuitive concepts of relevance. They also allow us to analyze the computational version of relevance introduced in [18] (see also section 4) for the purpose of answering historical queries.

It will be the case (section 5) that, in general, the syntactical counterpart will sanction as relevant more tuples than the semantical notion. This "unsoundness" of our algorithm for computing relevance might cause only some extra computational work, but no semantically relevant tuple will be missed in the computational process. We will also obtain, for specifications derived from effect axioms, like (1) (see section 5.3), that the set of computationally relevant actions subsumes the set of semantically relevant actions.

We will see that a perfect general computational characterization of the sets of relevant tuples and actions is not possible.

3.1 Semantical relevance of tuples

We will start with an action theory Σ as described in section 2, but we will drop the initial database, Σ_{S_0}, to have the freedom for changing the truth value of some tuples in S_0. We will also change the action preconditions, making every named action possible at every state, because there might exist tuples that appear only in the action preconditions, but not in the SSAs of any tuple. Since we are considering legal transaction lists, we do not need to update the tuples appearing in them, so we will not consider those tuples as relevant. In section 6 we suggest how to change the notions of relevance in order to include action preconditions.

Given an action theory $\Sigma = \Sigma_{bd} \cup \Sigma_{ap} \cup \Sigma_{una} \cup \Sigma_{ssa} \cup \Sigma_{S_0}$, we will define the auxiliary theory Σ^- by:

$$\Sigma^- = (\Sigma - (\Sigma_{S_0} \cup -\Sigma_{ap})) \cup \Sigma^+_{ap},$$

where theory Σ^+_{ap} is the result of replacing each axiom of the form $Poss(A, s) \equiv \Pi_A[s]$ by $Poss(A, s) \equiv true$.

Now, we define the semantical notion of relevance for ground tuples, i.e. ground instances of fluents.

Definition 1 (semantical relevance for ground tuples) *A set L of ground tuples is relevant to a ground tuple $F(\bar{c})$ iff there are models M and M' of Σ^- such that*

1. *They differ at S_0 only in L, that is, for all $F'(\bar{c}') \in L$, it holds*

$$M \models F'(\bar{c}')[S_0] \text{ iff } M' \models \neg\, F'(\bar{c}')[S_0].$$

Any other tuple not in L is interpreted in the same way at S_0 by both models.

2. *There is a list of ground basic action names $[a_1, \ldots, a_n]$, such that the tuple $F(\bar{c})$ is interpreted differently in state $do([a_1, \ldots, a_n], S_0)$, i.e.*

$$M \models F(\bar{c})[do([a_1, \ldots, a_n], S_0)] \text{ iff } M' \models \neg\, F(\bar{c})[do([a_1, \ldots, a_n], S_0)].$$

Tuple $F(\bar{c})$ has the same truth value in intermediate states: If $1 \leq k < n$, then it holds

$$M \models F(\bar{c})[do([a_1, \ldots, a_k], S_0)] \text{ iff } M' \models F(\bar{c})[do([a_1, \ldots, a_k], S_0)].$$

Other tuples can be interpreted arbitrarily (but according to Σ^-) by M and M' in those intermediate states.

3. *L is minimal, in the sense that there is no $L' \subsetneq L$ satisfying the above conditions.*

Figure 1 illustrates the concept of the relevance to a ground tuple $F(\bar{c})$ of a set L of ground tuples. The notation $L, \neg\, L$ means that any tuple in L is interpreted differently by the models M and M' at the corresponding state, though none of them is forced to interpret as permanently true or false all of the tuples in L. Notation $[\neg\,]F(\bar{c})$ means that the tuple $F(\bar{c})$ can either hold or not hold at the corresponding state, but satisfying the conditions 1.–3. Notice that, since we modified the original action preconditions, the transaction list $[a_1, \ldots, a_n]$ is always legal wrt to both M and M'.

We will say that a ground tuple $F'(\bar{c}')$ is relevant to a ground tuple $F(\bar{c})$ if there is a set L such that $F'(\bar{c}') \in L$ and L is relevant to $F(\bar{c})$.

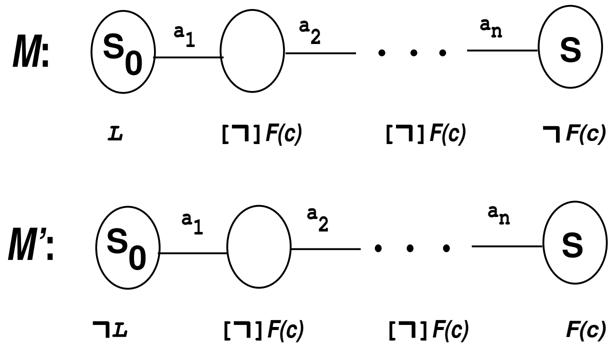

Figure 1: The Semantical Notion of Relevance of Tuples

Notice that in the definition the transaction list $[a_1 \ldots, a_n]$ may be empty (with $n = 0$). In this case, we can take $L = \{F(\bar{c})\}$, obtaining that any tuple is relevant to itself. With the restriction of minimality on a set L of tuples that are relevant to $F(\bar{c})$, we have that $F(\bar{c}) \in L$ implies that $L = \{F(\bar{c})\}$. Notice that without this restriction of minimality we could trivially make any tuple relevant to $F(\bar{c})$ by including it in L.

Since the models M and M' interpret everything in the same way in S_0, except for L, and we are executing the same transaction list, L must be the (minimal) cause for M and M' interpreting $F(\bar{c})[do([a_1, \ldots, a_n], S_0)]$ differently. Notice that if $F'(\bar{c}')$ is not relevant to $F(\bar{c})$ and we want to check the truth value of $F(\bar{c})$, then we do not need to know the truth value of $F'(\bar{c}')$, since it cannot change the truth value of $F(\bar{c})$.

Example 1 Consider the following specification:

$$\begin{aligned}
&Poss(a, s) \supset emp(x, d)[do(a, s)] \equiv \; a = hire(x, d) \lor emp(x, d)[s] \land \\
&\qquad\qquad \neg \exists y(a = fire(y, x) \land emp(y, d) \land boss(y)[s]). \\
&Poss(a, s) \supset boss(x)[do(a, s)] \equiv a = promote(x) \lor boss(x)[s] \land \neg\, a = quit(x). \\
&Poss(a, s) \supset salary(x, \$)[do(a, s)] \equiv \; a = change_salary(x, \$) \lor salary(x, \$)[s] \land \\
&\qquad\qquad \neg \exists \$'(a = change_salary(x, \$') \land \$' \neq \$).
\end{aligned}$$

Action $fire(y, x)$ is understood as y firing x. It would make x unemployed iff y is employed in the same department of x and is a boss.

The tuple $emp(john, 13)$ is relevant to $emp(sue, 13)$: Consider two models M and M' that interpret $boss(john)[S_0]$ and $emp(sue, 13)[S_0]$ as true and differ in S_0 only in $\{emp(john, 13)\}$: Say, $M \models \neg\, emp(john, 13)[S_0]$ and $M' \models emp(john, 13)[S_0]$ (it could be the other way around), interpreting any other tuple in S_0 in identical way. Clearly, the set $\{emp(john, 13)\}$ is minimal. We have that

$$\begin{aligned}
&M \models emp(sue, 13)[do(fire(john, sue), S_0)], \\
&M' \models \neg\, emp(sue, 13)[do(fire(john, sue), S_0)].
\end{aligned}$$

This construction can be done in another way: Consider two models M and M' that interpret $boss(john)[S_0]$ as false, $emp(sue, 13)[S_0]$ as true, and differ in S_0 only in $\{emp(john, 13)\}$: $M \models emp(john, 13)[S_0]$ and $M' \models \neg emp(john, 13)[S_0]$, interpreting any other tuple in S_0 in identical way. We have that

$$\begin{aligned}
&M \models \neg\, emp(sue, 13)[do([promote(john), fire(john, sue)], S_0)], \\
&M' \models emp(sue, 13)[do([promote(john), fire(john, sue)], S_0)].
\end{aligned}$$

Notice that tuple $emp(sue, 13)$ is interpreted in the same way by both M and M' in the intermediate state $do(promote(john), S_0)$.

The tuple $boss(john)$ is also relevant to $emp(sue, 13)$: Consider two models M and M' that interpret both $emp(john, 13)[S_0]$ and $emp(sue, 13)[S_0]$ as true and differ in S_0 only in $\{boss(john)\}$: Say, $M \models \neg\ boss(john)[S_0]$ and $M' \models boss(john)[S_0]$, interpreting any other tuple in S_0 in identical way. Clearly, $\{boss(john)\}$ is minimal. We have that

$$M \models emp(sue, 13)[do(fire(john, sue), S_0)],$$
$$M' \models \neg\ emp(sue, 13)[do(fire(john, sue), S_0)].$$

Notice that the constant $john$ can be replaced by any other constant. This would show that the tuples $emp(x, 13)$ and $boss(x)$ are relevant to $emp(sue, 13)$ for any value of x, an expected result: Anyone employed in department d can be promoted to a boss and any boss can fire sue. ■

We will denote the fact that set L is relevant to $F(\bar{c})$ by $L \leadsto F(\bar{c})$. Sometimes we will need to show explicitly the transaction list $[a_1, \ldots, a_n]$ in the definition of relevance, or at least, its length. We use the following notation:

- If we need to specify that there is a transaction list is of length n, we will use the notation $\stackrel{n}{\leadsto}$. If $L \stackrel{n}{\leadsto} F(\bar{c})$, we say that the set L is relevant in nth degree to $F(\bar{c})$. If $L \stackrel{1}{\leadsto} F(\bar{c})$ holds, we say that L is directly relevant to $F(\bar{c})$. A set can be relevant to a tuple in several degrees: In Example 1, the sets $\{emp(john, 13)\}$ and $\{boss(john)\}$ are directly relevant to $emp(sue, 13)$. The set $\{emp(john, 13)\}$ is also relevant to $emp(sue, 13)$ in second degree.
- If the actions involved in the definition of relevance are $A_1, \ldots, A_n$, we can state this explicitly with the notation $\leadsto_{A_1,\ldots,A_n}$.

Definition 2 *If L, L' are sets of ground tuples, we say that L' is relevant to L, also denoted by $L' \leadsto L$, if for each tuple $F_i(\bar{c}_i) \in L$ there is a subset L_i of L' such that $L_i \leadsto F_i(\bar{c}_i)$ and $\bigcup L_i = L'$.*

Later on we will see that the computational notion of relevance can be applied to non–ground tuples, so it is also interesting to define semantical relevance for this case:

Definition 3 (relevance for non ground tuples) *The set $\{F_1(\bar{c}_1, \bar{x}_1), \ldots, F_n(\bar{c}_n, \bar{x}_n)\}$ is relevant to the tuple $F(\bar{c}, \bar{x})$ iff for every variable substitution (of variables by terms) σ such that $F(\bar{c}, \bar{x})\sigma$ and $F_k(\bar{c}_k, \bar{x}_k)\sigma$ are ground, for $1 \leq k \leq n$, it holds*

$$\{F_1(\bar{c}_1, \bar{x}_1)\sigma, \ldots, F_n(\bar{c}_n, \bar{x}_n)\sigma\} \leadsto F(\bar{c}, \bar{x})\sigma.$$

We keep using the notation $\leadsto$ for relevance of non ground tuples. The semantical notion of relevance of tuples has the following properties

Theorem 1 *The set of triples (Σ, L, F), where Σ is an action theory, F is a tuple and L is a set of tuples relevant to F is not recursively enumerable.*

Corollary 1 *The problem of determining if a set of tuples L is relevant to a tuple F wrt an action theory Σ is undecidable.*

Therefore, there is no algorithm that, given an arbitrary action theory Σ and a tuple $F(\bar{c}, \bar{x})$, generates all the sets of tuples L such that $L \leadsto F(\bar{c}, \bar{x})$. However, in 4, we will we able to compute a small superset of the tuples that are semantically relevant to a given tuple.

3.2 Relevance of actions

Now we present the semantical definition of relevance of actions. An action $A(\bar{c}', \bar{x}')$ is relevant to tuple $F(\bar{c}, \bar{x})$ if it can change, from S_0 to $do(A(\bar{c}', \bar{x}'), S_0)$, the truth value of a tuple that is relevant to $F(\bar{c}, \bar{x})$. More precisely,

Definition 4 (semantical relevance of actions) *The action $A(\bar{c}', \bar{x}')$ is semantically relevant to the tuple $F(\bar{c}, \bar{x})$ wrt an action theory Σ iff there is and a tuple $F''(\bar{c}'', \bar{x}'')$ that is relevant to $F(\bar{c}, \bar{x})$, such that for every variable substitution σ such that $F''(\bar{c}'', \bar{x}'')[S_0]\sigma$ and $A(\bar{c}', \bar{x}')\sigma$ are ground, there is a model M of theory Σ^- (see the beginning of section 3.1) such that:*

$$M \models (F''(\bar{c}'', \bar{x}'')[S_0] \not\equiv F''(\bar{c}'', \bar{x}'')[do(A(\bar{c}', \bar{x}'), S_0)])\sigma$$

Notice that if an action is not relevant to a tuple $F(\bar{c}, \bar{x})$, or to a set of tuples, we are interested in, then we do not have to compute its effects, because it does not affect information that is relevant to the given tuple.

Example 2 Consider a specification with two action names, A and B, and a single fluent F with the SSA

$$Poss(a, s) \supset F(x)[do(a, s)] \equiv a \neq A(x) \vee (F(x)[s] \wedge a \neq A(x)). \tag{2}$$

- Action $A(x)$ is relevant to $F(x)$: Consider a model M such that $M \models F(x)[S_0]$. It holds that $M \models \neg\, F(x)[do(A(x), S_0)]$, for every value of x.
- Action $B(x)$ is also relevant to $F(x)$: Consider a model M such that $M \models \neg\, F(x)[S_0]$. It holds that $M \models F(x)[do(B(x), S_0)]$, for every value of x.

■

Clearly, there is no general algorithm for computing all the actions that are relevant to a tuple, because the notion is defined in terms of the non computable semantical notion of relevance of tuples.

4 Computing Relevance

In this section we present algorithms that, given a tuple $F(\bar{c}, \bar{x})$, allow us to obtain a set of tuples and a set of actions that are "computationally relevant" to $F(\bar{c}, \bar{x})$. This algorithm was used in [18] for answering historical queries using relevance. In section 5 we will relate these procedural notions of relevance with the semantical notions introduced before.

We will motivate and illustrate the algorithms for obtaining the relevant tuples and actions by means of an example:

Example 3 Suppose that we have the same specification of Example 1. We will compute the tuples and actions that are relevant to a generic tuple $emp(x, d)$ (the names of the variables used are not important), then we can obtain the relevant tuples and actions for a specific tuple by an appropriate variable instantiation.

As shown before, the tuples that are semantically relevant to $emp(x, d)$ are $emp(y, d)$ and $boss(y)$, for any value of variable y. On the other hand, in the computational approach, we obtain these *relevant tuples* by recursively looking at the formulas $\Phi_{F(\bar{c},\bar{x})}$ appearing in the right-hand sides of the SSAs[2]. We do not need to look at the tuples appearing in the (omitted) action precondition axioms, because we will check the query against a transaction list $\bar{T}$, known to be legal: Since the action preconditions are satisfied, we do not need to keep the tuples appearing in those formulas.

If we want to keep updated the tuple $emp(x,d)$, we start by adding it to the list of relevant tuples. We will regard x and d as constants, since they stand for a fixed but generic entry in table *emp*. Next, we look at the tuples appearing in formula $\Phi_{emp(x,d)}$ (see section 2 for a description of the formulas $\Phi_{F(\bar{c},\bar{x})}$), that is in

$$a = hire(x,d) \quad \vee \quad (emp(x,d)[s] \wedge \; \neg \; \exists \, y (a = fire(y,x) \wedge emp(y,d) \wedge boss(y)[s])).$$

This gives $emp(y,d)$ and $boss(y)$ (notice that the fluents do not have the state argument). We will replace the variable y by an *anonymous variable*, _. (Later in this section we discuss this renaming.) We delete $emp(x,d)$ from the set of relevant tuples and add $emp(\_,d)$ to it, since $emp(x,d)$ represents a single entry in *emp*, while $emp(\_,d)$ represents several entries, which include $emp(x,d)$. We also add $boss(\_)$. Now, we look at the formula $\Phi_{boss(\_)}$, that is at

$$a = promote(\_) \quad \vee \quad (boss(\_)[s] \wedge \neg \; a = quit(\_)),$$

obtaining $boss(\_)$. We already have it in the set of relevant tuples, so we do not add it again. Now we look at the SSA of $emp(\_,d)$, that is at

$$a = hire(\_,d) \quad \vee \quad (emp(\_,d)[s] \wedge \; \neg \; \exists \, y (a = fire(y,\_) \wedge emp(y,d) \wedge boss(y)[s])).$$

The tuples appearing there are $emp(\_,d)$, $emp(y,d)$ and $boss(y)$. We rename y to _, finding out that all of these tuples are already in the set of relevant tuples, so we do not add them. Since there are no more SSAs, we stop, obtaining that the set of tuples that are relevant to $emp(x,d)$ is $\{emp(\_,d), boss(\_)\}$, which is the expected result.

The set of tuples that are relevant to a specific entry in table *emp* is obtained by variable instantiation. For example, the set of tuples that are relevant to $emp(Sue,13)$ is obtained by the substitution $\sigma = \{x/Sue, d/13\}$. This gives us the set of relevant tuples $\{emp(\_,13), boss(\_)\}$. In general, an auxiliary table may be necessary, e.g. partially instantiated.

The actions that are relevant to an arbitrary tuple are obtained by a similar method: Since $emp(\_,d)$ and $boss(\_)$ are the only tuples we have to update, we look at the actions appearing in the SSAs of $emp(\_,d)$ and $boss(\_)$. This gives us the set $\{fire(\_,\_), hire(\_,d), quit(\_), promote(\_)\}$. Therefore, in order to check the truth value of $emp(Sue,13)$ along a transaction log T, we have to compute the effects of the actions $\{fire(\_,\_),\ hire(\_,13),\ quit(\_),\ promote(\_)\}$, if they appear in $\bar{T}$.

For example, if $\bar{T} = [hire(sue,13),\ promote(john),\ fire(john,sue),\ change\_salary(john,15.000),\ hire(peter,9)]$, the only relevant actions are $hire(sue,13)$, $promote(john)$ and $fire(john,sue)$, since they are the ones that are *particular cases* of elements of the set $\{fire(\_,\_),\ hire(Sue,13),\ quit(\_),\ promote(\_)\}$. The effects of the other actions can be discarded. This concept of particular case will be made precise below. ■

Notice that there can exist circularity in the SSAs: The SSA of $F(\bar{x})$ can mention $F'(\bar{x}')$ and vice-versa. In fact, if we use Reiter's specifications, with formulas of the form (1), then $\Phi_{F(\bar{x})}$ always mentions $F(\bar{x})$. This could make the recursive search go into an infinite loop. However, our algorithm will always stop, because it keeps only one version of the "most general" tuples,

[2]We use the notation $F(\bar{c},\bar{x})$ to refer to an entry in table F with some arguments that are instantiated (the $\bar{c}$ part) and some not that are not (the $\bar{x}$ part). They are not necessarily in any order and any part can omitted.

without state arguments. In order to define precisely this idea of generality, we need the notion of *subsumption*:

Definition 5 *1. An expression E' is subsumed by another expression E if there is a variable substitution σ such that $E\sigma = E'$, i.e. E is* more general *than E'.*

2. The above definition readily extends to sets of expressions: an expression E' is subsumed by a set of expression L if there is an expression E in L and a variable substitution σ, such that $E\sigma = E'$.

3. A set of expressions L' is subsumed by a set of expressions L if each element of L' is subsumed by L.

Notice that an infinite set of expressions can be subsumed by a finite set of expressions. Indeed, the infinite set $L' = \{f(1), f(2), f(3), \ldots\}$ is subsumed by the set $L = \{f(x)\}$, with x being a variable, since each element of L' is a particular case of $f(x)$. Notice also that an empty set never subsumes any nonempty set.

For example, $promote(john)$ is subsumed by $promote(y)$, since there is a variable substitution, namely $\sigma = \{y/john\}$, such that $promote(john) = promote(y)\sigma$. Therefore, the set of relevant actions for $emp(sue, 13)$ subsumes $promote(john)$, because it contains $promote(y)$.

In the algorithm for computing relevance of tuples, we will consider a special variable substitution, σ^*, that changes every variable by a new variable, $\_$. The substitution σ^* makes variables loose their identity. For example, if we apply σ^* to$\{E(x), E'(x)\}$, we obtain $\{E(\_1), E'(\_2)\}$, which does not convey the information that the variables in both expressions are the same. However, this does not matter for determining relevant tuples and actions. Furthermore, the _s can take different values in the same expression. This way, the expression $E(x, y)$ is subsumed by the expression $E(\_,\_)$. This renaming will guarantee that the algorithm for computing relevance stops.

Prior to the application of the algorithm, all of the variables appearing in $F(\bar{c}, \bar{x})$ must be replaced by (or treated as) new constants, so that they are not affected by the variable substitution σ^*. After the algorithm has stopped, the new constants introduced are replaced by the original variables. The reason for doing this is that, since we are considering an arbitrary tuple, those variables stand for specific values, so they will be instantiated sometime later by a variable substitution σ.

For instance, in example 3, if x and d were not replaced by new constants, they would be affected by σ^*, so the relevant tuples to $emp(x, d)$ would be $emp(\_,\_)$ and $boss(\_)$. If we instantiated x, d to Sue, 13 with the variable substitution $\sigma = \{x/Sue, d/13\}$, then the relevant tuples would still be $emp(\_,\_)$ and $boss(\_)$, which subsumes any tuple of the table emp, but gives too much freedom to the tuples in that table. We let σ^* change other variables, since they will not be affected by the instantiation σ anyway.

Now, we present in precise terms the general algorithm for computing CRT, the set of computationally relevant tuples. We define it by means of an operator T, such that, given a set of tuples S, $T(S)$ is obtained in a two–step process: First, the auxiliary set $AuxT(S)$ is generated by collecting in it all the tuples that are mentioned in the SSAs of the tuples in S that are not subsumed by S.[3] Then, $T(S)$ is generated by collecting in it all the tuples in $AuxT(S)$ and all the tuples in S that are not subsumed by $AuxT(S)$.

[3]The fluents can be collected linearly: If the SSA is $F(x) \wedge x = a$, we collect $F(x)$, not $F(a)$.

Definition 6 (computationally relevant tuples operator) *Given a set of tuples S, the operator T is defined by:*

$$
\begin{aligned}
AuxT(S) = \ & \{F'(\bar{c}', \bar{x}')\sigma^* \mid \textit{ there is } F''(\bar{c}'', \bar{x}'') \in S \textit{ and } F'(\bar{c}', \bar{x}') \textit{ is mentioned in } \Phi_{F''(\bar{c}'', \bar{x}'')} \\
& \quad \textit{and } F'(\bar{c}', \bar{x}') \textit{ is not subsumed by } S\}. \\
T(S) = AuxT(S) \cup \ & \{F'(\bar{c}', \bar{x}') \mid F'(\bar{c}', \bar{x}') \in S \textit{ and} \\
& \quad F'(\bar{c}', \bar{x}') \textit{ is not subsumed by } AuxT(S)\}.
\end{aligned}
$$

Example 4 Consider the specification in Example 1. Suppose that $S = \{emp(sue, 13)\}$. The tuples appearing in the SSA of $emp(sue, 13)$ are $emp(y, 13)$ and $boss(y)$. Notice that the tuples do not contain the state argument. Since y is a variable, it is changed to _. Therefore,

$$AuxT(\{emp(sue, 13)\}) = \{emp(\_, 13), boss(\_)\}.$$

Both tuples are collected, since they are not subsumed by $emp(sue, 13)$. Now, we have that

$$T(\{emp(sue, 13)\}) = \{emp(\_, 13), boss(\_)\}.$$

Tuple $emp(sue, 13)$ has been removed, because it is subsumed by $emp(\_, 13)$. ∎

The example shows that $T(S)$ always subsumes S. This is clear, since we are keeping the most general versions of the tuples.

We will need to apply the operator T several times to a given set of tuples. We will denote by $T^i(S)$ the result of applying the operator i times to the set of tuples S. In particular, we will compute $T^i(\{F(\bar{c}, \bar{x})\})$, for a tuple $F(\bar{c}, \bar{x})$, until all the tuples are collected, that is, until the operator O reaches a fix–point. An important property of the operator T is that it does reach a fix–point. This is guaranteed by the following theorem

Theorem 2 *Given a specification without function symbols on non–state variables appearing in the tuples in the SSAs, for any tuple $F(\bar{c}, \bar{x})$, there is an integer number P such that $T^P(\{F(\bar{c}, \bar{x})\}) = T^{P-1}(\{F(\bar{c}, \bar{x})\})$.*

Definition 7 *If $T^P(\{F(\bar{c}, \bar{x})\})$ is the fix–point of the operator T applied to $\{F(\bar{c}, \bar{x})\}$, then the set of relevant tuples to $F(\bar{c}, \bar{x})$, $CRT(F(\bar{c}, \bar{x}))$, is defined by:*

$$CRT(F(\bar{c}, \bar{x})) = T^P(\{F(\bar{c}, \bar{x})\}).$$

This set can be computed with the following algorithm:

Algorithm for Computing Relevant Tuples: The following algorithm computes the set of tuples that are relevant to a tuple $F(\bar{c}, \bar{x})$:

```
Finished := false;
Old := {F(c̄, x̄)};
REPEAT
  /* Store in New the output of T.
   This implies doing the above two-step process */
  New := T(Old);
  IF (Old = New) THEN Finished := true;
  Old := New;
```

```
UNTIL Finished;
RETURN Old;
```

An important property of the algorithm for computing relevance is that it terminates. This is guaranteed by the existence of the fix–point of the operator T.

Example 5 We will compute the fix–point of $T(\{emp(sue, 13)\})$. In Example 4, we saw that the first iteration yields $\{emp(\_, 13), boss(\_)\}$. For the second iteration, the tuples appearing on the right–hand sides of the SSA for $emp(\_, 13)$ are $emp(\_, 13)$, $emp(y, 13)$ and $boss(y)$. Since y is a variable, it is changed to $\_$. The only tuple appearing in the SSA of $boss(\_)$ is $boss(\_)$. All these tuples are subsumed by the result obtained in first iteration. Thus, we have:

$$AuxT(\{emp(\_, 13), boss(\_)\}) = \varnothing,$$
$$T(\{emp(\_, 13), boss(\_)\}) = \{emp(\_, 13), boss(\_)\}.$$

Since we obtained the same as in the first iteration, the fix–point of $T(\{emp(sue, 13)\})$ is the set $\{emp(\_, 13), boss(\_)\}$. ■

As said before, we can compute CRT in a finite number of steps. However, this is not necessarily true if we allow function symbols in the tuples.

Example 6 Suppose we have the following SSA, in which there is a function symbol in a fluent:

$$Poss(a, s) \supset F(n)[do(a, s)] \equiv F(n + 2)[s] \wedge a = A \vee F(n)[s]. \tag{3}$$

If we we want to compute the tuples that are relevant to $F(0)$. We obtain

$$\begin{aligned}
&T^0(\{F(0)\}) = \{F(0)\},\\
&T^1(\{F(0)\}) = \{F(0), F(2)\},\\
&\ldots\\
&T^n(\{F(0)\}) = \{F(0), F(2), \ldots, F(2 * n)\},\\
&\ldots
\end{aligned}$$

The operator does not reach a finite fix–point. The problem comes from the combination of two things: First, the specification is circular, in the sense that an instance of F is relevant to another instance of F. Second, the operator always contains ground tuples and new constants are introduced at each step, which are not subsumed by the result of the previous step. This makes the algorithm go into an infinite loop.

This problem can be satisfactorily solved with a slight modification of (3), consisting in replacing the function symbols in the fluents by introducing new variables: $F(f(x))$ is equivalent to $(\exists x')F(x') \wedge x' = f(x)$. With the following SSA, that is equivalent to (3),

$$Poss(a, s) \supset F(n)[do(a, s)] \equiv a = A \wedge (\exists n')(F(n')[s] \wedge n' = n + 2) \vee F(n)[s], \tag{4}$$

the algorithm yields the fix–point $\{F(\_)\}$:

$$\begin{aligned}
&T^0(\{F(0)\}) = \{F(0)\},\\
&T^1(\{F(0)\}) = \{F(_)\},\\
&T^2(\{F(0)\}) = \{F(_)\} = T^1(\{F(0)\}).
\end{aligned}$$

Notice that (4) does satisfy the hypothesis of Theorem 2, of not allowing function symbols over non–state terms in the fluents. ■

We have to pay a price for the guaranteed existence of the finite fix–point of T: We do not always obtain a precise characterization of the intuitively relevant tuples. In the previous example, the only intuitively relevant tuples to $F(0)$ are of the form $F(2*k)$, with k positive integer, but the fix–point $\{F(\_)\}$ subsumes any entry in table F, in particular, the "irrelevant" tuples $F(1)$, $F(3)$, $F(5)$, $F(7)$, etc. Nevertheless, in section 5 we will see that it is always the case that the set of semantically relevant tuples is subsumed by the set of computationally relevant tuples.

Having defined the computational notion of relevance of tuples, let us define the set of computationally *relevant actions* for a tuple $F(\bar{c}, \bar{x})$. Basically, having obtained $CRT(F(\bar{c}, \bar{x}))$, we directly collect all the actions appearing in the SSAs of the tuples in $CRT(F(\bar{c}, \bar{x}))$ (see Example 3). So, no recursion is necessary.

Definition 8 (set of computationally relevant actions) *The set of actions that are computationally relevant to tuple $F(\bar{c}, \bar{x})$, $CRA(F(\bar{c}, \bar{x}))$, is defined by:*

$$\begin{aligned} CRA(F(\bar{c}, \bar{x})) = \ & \{A(\bar{c}', \bar{x}') \mid \textit{ there is } F''(\bar{c}'', \bar{x}'') \in CRT(F(\bar{c}, \bar{x})) \textit{ and} \\ & A(\bar{c}', \bar{x}') \textit{ is mentioned in } \Phi_{F''(\bar{c}'', \bar{x}'')} \textit{ and} \\ & A(\bar{c}', \bar{x}') \textit{ is not an action variable}\}. \end{aligned}$$

It is not necessarily the case that the actions that are semantically relevant to $F(\bar{c}, \bar{x})$ are subsumed by $CRA(F(\bar{c}, \bar{x}))$. In section 5.2 we present an example of SSAs that illustrates this situation. However, as we will see in section 5.3, there is at least one important syntactical class of SSAs for which $CRA(F(\bar{c}, \bar{x}))$ does subsume all the semantically relevant actions, namely all specifications derived from effect axioms.

5 Semantical vs. Computational Notions of Relevance

5.1 Comparing the notions of relevance of tuples

As shown in section 3, a perfect computational characterization of the semantically relevant tuples is not possible. So, any algorithm that computes a schema for generating tuples that subsumes all the semantically relevant tuples is interesting, and can be used, for example, for answering historical queries [18]. We can say that the algorithm computes a (hopefully small) *superset* of the semantically relevant tuples. We will see that the computational notion of relevance of tuples presented in 4 has this property. In fact, the following example shows that the algorithm may compute a proper superset.

Example 7 Let us consider the specification given in example 6. We can see that $\{F(2*k)\}$ is semantically relevant to $F(0)$ in kth degree: Simply make M interpret as false all the entries of F in S_0, except for $F(2*k)$, and make M' interpret as false all the entries of F in S_0. Consider a transaction list $\bar{T} = [A, \ldots, A]$ of length k. We have that $M \models F(0)[do(\bar{T}, S_0)]$ and $M' \models \neg F(0)[do(\bar{T}, S_0)]$.

Since the truth value of $F(n)$ depends only on the truth values of $F(n+2)$, $F(n+4)$, $F(n+6)$, ... in the previous states, the truth values of the tuples $F(2*k+1)$ are not semantically relevant to $F(0)$. However, they are subsumed by $CRT(F(0))$. ■

Now we state the theorem that ensures that the computational notion of relevance does compute a superset of the relevant tuples:

Theorem 3 *Given tuples $F(\bar{c},\bar{x})$ and $F'(\bar{c}',\bar{x}')$, if $F'(\bar{c}',\bar{x}') \rightsquigarrow F(\bar{c},\bar{x})$, then for every ground substitution σ, $F'(\bar{c}',\bar{x}')\sigma$ is subsumed by $CRT(F(\bar{c},\bar{x}))$.*

Corollary 2 *If there are ground of a tuple $F'(\bar{c}',\bar{x}')$ that are not subsumed by $CRT(F(\bar{c},\bar{x}))$, then $F'(\bar{c}',\bar{x}')$ is not semantically relevant to $F(\bar{c},\bar{x})$.*

The next example shows that in the theorem it is not possible to obtain the conclusion "$F'(\bar{c}',\bar{x}')$ is subsumed by $CRT(F(\bar{c},\bar{x}))$", but this is not critical, since the theorem tells us that all the information about $F'(\bar{c}',\bar{x}')$ that is relevant to $F(\bar{c},\bar{x})$ is contained in $CRT(F(c,x))$.

Example 8 Consider a language with constants A and B. Assume that we have the following SSA for F:

$$F(x,y,do(a,s)) \equiv F(x,y,s) \vee F(x,B,s).$$

We want to compute the tuples that are relevant to $F(x,A)$. So, we instantiate the SSA,obtaining

$$F(x,A,do(a,s)) \equiv F(x,A,s) \vee F(x,B,s).$$

We obtain $\{F(x,A), F(x,B)\}$ as the set of relevant tuples (the fix–point is reached after one step).

Now we ask whether $F(x,y) \rightsquigarrow F(x,A)$. For answering this, we have to check, for every ground substitution $x \mapsto t$, if

- $F(t,A) \rightsquigarrow F(t,A)$,
- $F(t,B) \rightsquigarrow F(t,A)$.

This is easy to do. For example, in the second item we can take the model M, where at S_0, $F(t,B)$ and $\neg F(t,A)$ are true, and a model M' where $\neg F(t,B)$ and $\neg F(t,A)$ are true. In M we have $F(t,A)$ at the successor state, but in M' we have $\neg F(t,A)$ at the successor state.

So, we have that $F(x,y)$ is semantically relevant to $F(x,A)$, but $CRT(F(x,A))$ does not subsume $F(x,y)$. Nevertheless it is true that every ground instance of $F(x,y)$ is subsumed by $CRT(F(x,A))$. ■

5.2 Comparing the notions of relevance of actions

It is not always the case that the actions that are semantically relevant to a tuple $F(\bar{c},\bar{x})$ are subsumed by $CRA(F(\bar{c},\bar{x}))$, the set of computationally relevant actions. In example 2, though action $B(x)$ is semantically relevant to $F(x)$, it is not subsumed by $CRA(F(x))$: $CRT(F(x))$ is $\{F(x)\}$, so $CRA(F(x))$ is $\{A(x)\}$.

The reason for this asymmetry between tuples and actions is that tuples have a truth value by themselves, while actions are terms in our language, so they must appear in (in)equalities in the SSAs (there is no other way they can appear, because of the restriction of simplicity on the SSAs). These (in)equalities can be such that they do not actually mention the relevant actions, as action $B(x)$ in the previous example.

5.3 An important particular case

SSAs like (2) in example 2 are not very natural. One usually positively knows which are the few actions that make the fluent become true in the successor state. This knowledge can be represented by means of *effect axioms*, and from them, SSAs can be derived [14]. We will show that for this important syntactical class of specifications, all the semantically relevant actions are subsumed by the computational counterpart.

Let us consider formulas of the form

$$a = A_1(\bar{x_1}) \wedge \varphi_1 \vee \cdots \vee a = A_n(\bar{x_n}) \wedge \varphi_n, \tag{5}$$

where a is an action variable and $A_1, \ldots, A_n$ are action names. We can use this formulas to give a sufficient condition on the syntactical form of the SSAs that ensures that $CRA(F(\bar{c}, \bar{x}))$ subsumes the semantically relevant actions:

Theorem 4 *If all the SSAs are of the form*

$$Poss(a, s) \supset F(\bar{x})[do(a, s)] \equiv \exists\, \gamma_F^+[s] \vee (F(\bar{x})[s] \wedge \neg\, \exists\, \gamma_F^-[s]), \tag{6}$$

with γ_F^+, γ_F^- of the form (5) and $\exists$ denoting the existential closure of all free variables different from $\bar{x}, a, s$, then, for all the actions $A(\bar{c}', \bar{x}')$ that are semantically relevant to $F(\bar{c}, \bar{x})$ and every substitution σ such that $A(\bar{c}', \bar{x}')\sigma$ is ground, one has that $CRA(F(\bar{c}, \bar{x}))$ subsumes $A(\bar{c}', \bar{x}')\sigma$.

We will show that the specifications obtained from effect axioms [14] satisfy the hypothesis of the theorem. Basically, a specification based on effect axioms contains the following axioms:

Positive Effects Axioms: For pairs formed by a fluent F and an action name A, where the action causes the fluent to become true, an axiom of the form:

$$Poss(A(\bar{x}), s) \wedge \varphi(\bar{x}, \bar{y})[s] \supset F(\bar{y})[do(A(\bar{x}), s)]. \tag{7}$$

That is, if action A is possible, and the preconditions on the fluent F, represented by the meta-formula $\varphi(\bar{x}, \bar{y})[s]$, are true at state s, then F becomes true in the successor state $do(A(\bar{x}), s)$ obtained after executing action A in state s. Here, $\bar{x}$ and $\bar{y}$ are parameters of the action and the fluent, respectively.

Negative Effects Axioms: For pairs formed by a fluent F and an action name A, where the action causes the fluent to become false, an axiom of the form:

$$Poss(A(\bar{x}), s) \wedge \psi(\bar{x}, \bar{y})[s] \supset \neg F(\bar{y})[do(A(\bar{x}), s)]. \tag{8}$$

The syntactic completion process performed on this preliminary specification is the following [14]: For each fluent we combine its positive effect axioms into a single general positive effect axiom. To see how, assume that we have only two positive effects axioms for fluent F:

$$\begin{array}{l} Poss(A_1(\bar{x_1}), s) \wedge \varphi_1(\bar{x_1}, \bar{y})[s] \supset F(\bar{y})[do(A_1(\bar{x_1}), s)], \\ Poss(A_2(\bar{x_2}), s) \wedge \varphi_2(\bar{x_2}, \bar{y})[s] \supset F(\bar{y})[do(A_2(\bar{x_2}), s)]. \end{array}$$

They can be combined into a single general positive effect axiom for F:

$$\begin{array}{r} Poss(a, s) \wedge [(\exists\, \bar{x_1}, \bar{x_2})(a = A_1(\bar{x_1}) \wedge \varphi_1(\bar{x_1}, \bar{y})[s] \vee a = A_2(\bar{x_2}) \wedge \varphi_2(\bar{x_2}, \bar{y})[s])] \\ \supset F(\bar{y})[do(a, s)]. \end{array}$$

Thus, for each fluent, F, we obtain a general positive effect axiom of the form:

$$Poss(a, s) \wedge \gamma_F^+[s] \supset F[do(a, s)].$$

The resulting formula $\gamma_F^+[s]$ is of the form (5). Similarly, we obtain a general negative effect axiom for each fluent of the form:

$$Poss(a,s) \wedge \gamma_F^-[s] \supset \neg F[do(a,s)].$$

If we make the assumption that the general effect axioms for a fluent provide all the conditions under which a fluent may change its truth value, we obtain a SSA of the form (6) for each fluent F. Having explained this transformation, we immediately obtain the conclusion:

Corollary 3 *Given a specification in terms of SSAs that are generated from effect axioms, for all tuples $F(\bar{c},\bar{x})$ and all the actions $A(\bar{c}',\bar{x}')$ that are semantically relevant to $F(\bar{c},\bar{x})$ and every substitution σ such that $A(\bar{c}',\bar{x}')\sigma$ is ground, one has that $CRA(F(\bar{c},\bar{x}))$ subsumes $A(\bar{c}',\bar{x}')\sigma$.*

In addition to this result, it can also be shown that in the even more particular case of SSAs that are *context–free* [11], the semantical and computational notions of relevance (for both tuples and actions) coincide. In this case, all of the SSAs are of the form[4]

$$\begin{array}{rl} Poss(a,s) \supset F(\bar{x})[do(a,s)] \equiv & [(\exists\,\bar{u})a = A_1(\bar{x},\bar{u}) \vee \cdots \vee (\exists\,\bar{v})a = A_k(\bar{x},\bar{v})] \vee \\ & F(\bar{x})[s] \wedge \neg\,[(\exists\,\bar{w})a = B_1(\bar{x},\bar{w}) \vee \cdots \vee (\exists\,\bar{t})a = B_j(\bar{x},\bar{t})] \end{array} \tag{9}$$

For example, the following SSA

$$\begin{array}{rl} Poss(a,s) \supset emp(x,d)[do(a,s)] \equiv & a = hire(x,d) \vee \\ & emp(x,d)[s] \wedge \neg\,(a = fire(x) \vee a = quit(x)). \end{array}$$

is context–free. Specifications with this kind of SSAs have the property that the truth value of a tuple $F(\bar{x})$ does not depend on the truth value of other tuples, but just on the executed actions. This is obviously a limitation, e.g. the specification in example 1 is not context–free, and so, it would not be admissible.

The SSA, $Poss(a,s) \supset F(\bar{x})[do(a,s)] \equiv \Phi_{F(\bar{x})}$, for $F(\bar{x})$ is called *variable independent* if all the variables not appearing in $\bar{x}$ appear at most once in a same occurrence of an action in $\Phi_{F(\bar{x})}$. All the SSAs we have showed so far are variable independent. Not variable independent SSAs are not frequent, but they might appear. The following is a context–free axiom that is not variable independent:

$$Poss(a,s) \supset F(\bar{x})[do(a,s] \;\equiv\; \exists\, y \;\, a = A(y,y,\bar{x}) \vee (F(\bar{x})[s] \wedge \neg a = B(x)).$$

For this kind of axioms, it could happen that computationally relevant actions are not semantically relevant, e.g. an action of the form $A(c,d,\bar{x})$, derived from the computed $A(\_,\_,\bar{x})$, is not semantically relevant, because the first and second arguments are different.

Theorem 5 *For specifications based on variable independent context–free successor state axioms, the sets of computationally relevant actions and semantically relevant actions coincide.*

All the examples of algorithms presented in [21] are for variable independent context–free specifications. It would be interesting to find more expressive classes of specifications for which the semantical and syntactical approaches to relevance coincide.

[4] The notation $A(\bar{x},\bar{u})$ below means that the variables in the action are among $\bar{x},\bar{u}$.

6 Possible Extensions and Related Approaches

In this section we state and discuss some possible alternative approaches and some open problems.

6.1 On the notions of relevance of actions

An interesting topic of research consists in finding other syntactical classes of specifications such that $CRA(F(\bar{c}, \bar{x}))$ subsumes all actions that are semantically relevant to a tuple $F(\bar{c}, \bar{x})$. This does not seem to be easy, since there are many reasonable formulas that mention few or none actions and still every action is semantically relevant. Here we present two examples:

- Suppose that the only actions are $A(x)$ and $B(x)$, with x variable. If Φ_F is $a \neq A(c)$, with c a constant, then the actions $B(\_)$ are semantically relevant to F. However, any action $A(x)$, with $x \neq c$, is also relevant. So, we should find in CRA the action $A(x)$ with the additional restriction $x \neq c$. This is not possible, because the set CRA admits only (partially instantiated) actions, but no conditions. The only solution consists in admitting every action as computationally relevant.

- Another case is when Φ_F does not mention any action. In this case, the truth value of fluent $F(x)$ can change from state S_0 to $do(a, S_0)$ regardless of the action a, so any action is semantically relevant. A related case appears when $\Phi_F = \varphi \vee \psi$, with only ψ mentioning actions. In this case, every action is semantically relevant, since any action is relevant to φ.

Another interesting topic of research consists in finding a syntactical transformation of formulas the Φ_F, leading to equivalent formulas such that $CRA(F(\bar{c}, \bar{x}))$ subsumes all the actions relevant to $F(\bar{c}, \bar{x})$. A trivial transformation is $\Phi_F \mapsto [\Phi_F \wedge \exists(a = A_1(\bar{x_1}) \vee \cdots \vee a = A_n(\bar{x_n})]$, with $A_1(\bar{x_1})$, ..., $A_n(\bar{x_n})$ being all the actions in the specification. This is not very efficient, since it forces $CRA(F(\bar{c}, \bar{x}))$ to sanction as relevant every action. It would be interesting to find a smarter transformation that does not obtain that many actions.

6.2 On the computational notion of relevance of tuples

We have shown that our computational notion of relevance cannot characterize exactly the semantically relevant tuples. However, we would like to compute a superset of the set of semantically relevant tuples that is as small as possible. In the definition of operator the T, presented in section 4, some "slack" between the sets is introduced by renaming the variables in the tuples to _. This renaming can make the operator T to regard as relevant more tuples than we would obtain without the renaming. This is illustrated in the following example:

Example 9 Suppose that we have the following specification:

$$Poss(a, s) \supset F(x)[do(a, s)] \equiv (\exists\, y) P(y, y),$$
$$Poss(a, s) \supset P(x, y)[do(a, s)] \equiv P(x, y)[s],$$

and we want to compute the tuples relevant to $F(x)$. We obtain the set $\{F(x), P(\_, \_)\}$. This means that any entry in fluent P is relevant to $F(x)$. However, the only semantically relevant tuples are of the form $P(y, y)$, with y variable, i.e. all the entries in fluent P that have both arguments with the same value. This difference between the semantically and computationally relevant tuples could be avoided in this case by not renaming the variable y to _. ■

The renaming of the variables is required in the proof of Theorem 2, stating that the operator T reaches a finite fix–point. If we could show that operator T does reach a fix–point, even if we do not rename the variables to _, we would obtain a better computational characterization of the semantically relevant tuples. In many examples (some of them can be found in [18]) there is no difference if we rename the variables to _ or not.

6.3 Considering action preconditions

An advantage of the computational notions of relevance is that the process of computing the relevant tuples and actions to a generic tuple must be done just once, prior to checking any query against any transaction list [18]. This is because the computation is done just by looking at the SSAs, which are static. In particular, action precondition axioms are not considered. In its turn, the semantical notion of relevance does not consider action preconditions either. However, one would think that a more accurate computational characterization of the intuitively relevant tuples and actions should include other elements of the action theory, in particular, the action preconditions axioms.

One might think that considering the action preconditions would have the effect of restricting the transaction lists that can appear in the definition of semantical relevance, thus reducing the number of relevant tuples. Unfortunately, this intuition is wrong. There are cases where this is the case, but others where the contrary happens.

Example 10 Consider the following axioms:

$$\begin{array}{l} Poss(A(x), s) \equiv F(x)[s] \\ Poss(a, s) \supset F(x)[do(a, s)] \equiv (a = A(x) \wedge P(x)[s]) \vee F(x)[s]. \end{array}$$

Without including the the precondition axioms in the definition of relevance, the tuple $P(x)$ is semantically relevant to tuple $F(x)$: Consider two models that interpret $F(x)[S_0]$ as false and differ in $\{P(x)\}$ at S_0. The models interpret differently $F(x)[do(A(x), S_0)]$. However, this construction is not possible if we include action preconditions: Action $A(x)$ is possible iff the models interpret $F(x)[S_0]$ as true. Therefore, if we do not include the action preconditions, we regard as semantically relevant tuples that would not be such if we include the preconditions. ■

Example 11 Consider the following specification:

$$\begin{array}{l} Poss(A(x), s) \equiv (P(x) \equiv R(x))[s] \\ Poss(a, s) \supset F(x)[do(a, s)] \equiv a = A(x) \wedge P(x)[s] \\ Poss(a, s) \supset P(x)[do(a, s)] \equiv P(x)[s]. \end{array}$$

We would like to regard only $P(x)$ as relevant to $F(x)$, because the truth value of $F(x)$ in any successor state is obtained only from the truth value of this tuple. However, if we consider preconditions, we cannot construct two models that differ only in $\{P(x)\}$ at S_0: $Poss(A(x), S_0)$ would be false in one of the models, so in this case we have to regard as relevant the set $\{P(x), R(x)\}$.

Notice that the tuple $R(x)$ is not captured by $CRT(F(x))$. ■

Examples like the last one, with action precondition axioms considered, motivated our introduction of *sets* of relevant tuples, instead of just tuples (or singleton sets of tuples). In this example we obtained the two elements set $\{P(x), R(x)\}$ as relevant (because P and R are coupled together). Nevertheless, in all the examples before than do not consider action preconditions for relevance, only singleton sets of relevant tuples have been obtained. Of course there may be many singleton sets.

We can see from the two previous examples, that considering precondition axioms can make a difference wrt the definition of semantical relevance of tuples we gave. We chose not to consider preconditions, because the tuples appearing only in the preconditions play no role in checking a formula against a legal transaction list.

If we modify the semantical definition of relevance of tuples by including the action preconditions, a natural question is how to modify the operator T, presented in section 4, so that we can obtain at least the (new) semantically relevant tuples. The idea is to make the auxiliary operator $AuxT(S)$ collect the tuples appearing in the precondition axioms of the actions appearing in the SSAs of the tuples in S, keeping only the most general ones. This is illustrated in the following example:

Example 12 Consider the following specification:

$$\begin{array}{l} Poss(A(x), s) \equiv (\exists\, y) F(y)[s] \wedge y = x + 1, \\ Poss(a, s) \supset F(x)[do(a, s)] \equiv (a = A(x) \wedge P(x)[s]) \vee F(x)[s], \\ Poss(a, s) \supset P(x)[do(a, s)] \equiv a = A(x) \vee P(x)[s]. \end{array}$$

The set of tuples that are relevant to $F(x)$ should be is $\{F(\_), P(x)\}$ (instead of only $P(x)$ and $F(x)$), because of the $F(y)$ appearing in the precondition axiom for action A. Let us compute the set of tuples that are relevant to $F(x)$ with the modified operator T: We start with $T^0(F(x)) = \{F(x)\}$. Then, we have that

$$\begin{array}{l} AuxT(\{F(x)\}) = \{F(\_), P(x)\} \\ T(\{F(x)\}) = \{F(\_), P(x)\}. \end{array}$$

We add $F(y)\sigma^* = F(\_)$, because it appears in the action precondition of action $A(x)$, which in its turn appears in the SSA of $F(x)$. In the second iteration, we obtain:

$$\begin{array}{l} AuxT(\{F(\_), P(x)\}) = \varnothing \\ T(\{F(\_), P(x)\}) = \{F(\_), P(x)\}. \end{array}$$

■

The precondition axioms should satisfy the restriction of not allowing function symbols over non–state variables in the fluents. It is easy to see that the new operator T coincides with the operator presented in section 4 when all the action preconditions are of the form $Poss(A, s) \equiv true$.

We have not proven that this new operator reaches a finite fix–point and that this fix–point subsumes all the semantically relevant tuples (in the new sense, considering preconditions). We think that this new operator has these properties, because it keeps the most general tuples and it includes the tuples appearing in the action preconditions of the actions appearing in the SSAs.

6.4 Relevance via a given transaction list

As we mentioned in the introduction, our original motivation for introducing relevance was the problem of efficiently answering historical queries wrt a given transaction list [18]. Nevertheless, neither the semantical nor the computational notions of relevance we presented consider the transaction list against which we are checking the query. The transaction list is considered after the relevant tuples and actions have been obtained. This has the advantage of computing the relevant objects only once, and using them for different queries wrt to different transaction lists.

One might think of restricting the actions in those definitions to the actions appearing in the given list. In particular, in the computational notion an optimization could be performed by pruning from the the SSAs the information associated to actions not appearing in the transaction list. In this way, less tuples would be obtained and the computation would be speeded up.

This approach should be further explored, but some care is necessary. In Example 1, if we are given a transaction list that does not include the action *fire*, we would not update the table *boss*, since no boss is firing anyone. We should keep updated any tuple of the form $emp(\_, 13)$, because anyone in department 13 can be promoted to boss and fire Sue. In the new setting, none of these entries would be relevant to $emp(sue, 13)$, in particular, neither $emp(sue, 13)$ would be relevant, which is incorrect.

We are currently working in this interesting subject [19].

6.5 Extension to complex actions

In Reiter's seminal papers [14, 17] only *primitive actions* are considered. Following that approach, in our work we have presented notions of relevance wrt primitive actions. The Cognitive Robotics Group at the University of Toronto has presented an extension of Reiter's work that considers complex actions [8]. Basically, given a set of primitive actions, they extend them by introducing some extra–logical symbols, such as **while**, **if**, etc. In this way, it is possible to execute complex actions such as transferring a department's personnel to another department (*While there is someone employed in department* 13*, fire he/she from that department and hire he/she in department* 9*.*).

A semantical definition of relevance of complex actions should be as for primitive actions: An action A, either primitive or complex, will be relevant to a tuple $F(\bar{c}, \bar{x})$ iff it can change the truth value of a tuple that is relevant to $F(\bar{c}, \bar{x})$ from one state s to $do(A, s)$.

Although we have not worked out the details, intuitively, a complex action will be computationally relevant to a tuple iff at least one of the primitive actions appearing in the complex action is relevant to the tuple (in the sense presented in section 4). For example, action $[a_1; \ldots; a_n]$, denoting the sequential execution of $a_1, \ldots, a_n$, will be relevant to a tuple $F(\bar{c}, \bar{x})$ iff at least one of the a_i's is relevant to $F(\bar{c}, \bar{x})$. Action $[a_1 \mid \ldots \mid a_n]$, denoting the nondeterministic execution of one of the a_i, will be relevant to a tuple $F(\bar{c}, \bar{x})$ iff at least one of the a_i's is relevant to $F(\bar{c}, \bar{x})$, and so on. It remains to be proven that the complex actions that are computationally relevant to a tuple subsume all the complex actions that are semantically relevant to that tuple.

6.6 Comparing our approach with Lin and Reiter's forgetting

In [9], Lin and Reiter formalized the notion of forgetting. Given a theory Σ and a set of ground instances of predicates, S, they denote by $forget(\Sigma, S)$ the theory resulting from forgetting the elements of S in Σ. This new theory can be obtained by a syntactical manipulation if Σ is finite and the cardinality of S is finite. Lin and Reiter consider general theories, without treating the specific case of theories of action and change. Semantically, a structure M is a model of $forget(\Sigma, S)$ if it agrees with a model of Σ on everything except for the truth values of the elements in S.

They say that a set S of ground instances of predicates is *irrelevant* to a query Q if

$$\Sigma \models Q \textit{ iff } forget(\Sigma, S) \models Q \quad \text{and} \quad \Sigma \models \neg\, Q \textit{ iff } forget(\Sigma, S) \models \neg\, Q.$$

This means that a set S is irrelevant to Q iff the entailment of Q from Σ can be obtained regardless of the truth values of the elements in S. Notice that in order to check if S is irrelevant to Q, we must have $forget(\Sigma, S)$. Then we need S prior to checking relevance.

Our notion of relevance of tuples can be expressed in terms of forgetting: $L \rightsquigarrow_{A_1,\ldots,A_n} F(\bar{c})$ iff there are L_1, ..., L_n, sets of ground tuples (without the state argument), and a model M such that $M \models \mathit{forget}(\Sigma^-, S)$, where Σ^- is as in section 3, and

$$S = L[S_0] \cup L_1[do(A_1, S_0)] \cup \ldots \cup L_n[do([A_1, \ldots, A_n], S_0)].$$

Here, $L[s]$ denotes the set of tuples in L with their state arguments set to s. Furthermore, the following conditions must hold: $F(\bar{c}) \in L_n$, and for $1 \leq k < n$, $F(\bar{c}) \notin L_k$. Notice that S can be of infinite cardinality, so it might be the case that none of the syntactical transformations presented in [9] can be applied.

Lin and Reiter also show how to forget about an entire relation. However, we are usually interested in forgetting only a part of a relation, though that part could be of infinite cardinality (e.g. when the irrelevant tuples are partially instantiated, and the variables appearing in them can take an infinite number of values).

From a computational point of view, there is a fundamental difference between their approach and ours: Their approach does not allow to directly compute a list of relevant tuples to a query. Instead they should guess a set S, computationally generate $\mathit{forget}(\Sigma, S)$, and then check irrelevance, probably by a exhaustive process. Our approach, on the other hand, starts by considering the query, and then computes the tuples that are (ir)relevant to the query.

Another important difference is that they cannot accommodate relevance of actions, since relevance is only defined for (instances of) predicates. We think that not only the relevant information is important (in the situation calculus, the fluents), but also the *mechanisms* that make relevant information change (in the situation calculus, the actions). We think that this task is easier in the situation calculus (with a solution to the Frame Problem), since the effects of the actions are explicitly shown.

7 Conclusions

(1) Using Reiter's solution to the frame problem in the situation calculus and his application to specifications of database updates [14, 17], we have presented semantical notions of relevance of actions and entries in a fluent.
(2) As shown in [20, 18], the notions of relevance presented in sections 3 and 4 can be applied to optimize the process of answering historical queries in databases. A naïve algorithm for answering historical queries requires checking the truth value of a given formula along a given, legal, transaction list T. The computational notions of relevance can be used to optimize the algorithm.
(3) We showed that the semantical notions of relevance have syntactical, computable, counterparts that may sanction as relevant more things than the respective semantical notions. When answering historical queries, this can cause only an extra computational cost (in comparison with a possible application of the semantical notion), but it does not produce incorrect answers to the queries. On the other side, the cost is lower than not applying any computational concept of relevance.
(4) We showed that perfect computational counterparts of the semantical notions of relevance are not always possible. Nevertheless, in many interesting cases, including those specifications arising from effect axioms and context–free SSAs, there are satisfactory results.
(5) The computational notions of relevance and the optimized algorithms for answering historical queries have been implemented in the database reasoner SCDBR [3, 1].
(6) The semantical notion of relevance for entries in fluents can be seen in terms of *forgetting*, presented by Lin and Reiter [9]. Related approaches in the database context appear in [5] and [6], and are discussed in [18].

(7) Future research includes changing the definition of semantical relevance by including action preconditions, and obtaining a more accurate procedural characterization of the semantically relevant actions and entries in tables. Other open problems were presented in section 6.
(8) A research topic of high interest has to do with semantical historical query optimization, that is, with taking advantage of integrity constraints for efficient answering of historical queries in databases. We think that the combination of the approaches to integrity constraint management, as presented in [16, 10, 4] in the context of Reiter's database updates specifications, with the techniques developed in this work is very promising.
(9) More speculatively, we foresee an interesting application of our notions of relevance to modeling autonomous agents. We can think that an autonomous agent has a *utility function* $\psi : state \mapsto \mathbb{N}$. The rule is "the higher the value of $\psi(S)$, the happier the agent is in S". A natural assumption is that the autonomous agent is rational and will execute a legal transaction list $T = [A_1, \ldots, A_n]$ that maximizes $\Psi(T)$, where

$$\Psi(T) = \psi(S_0) + \psi(do(A_1, S_0)) + \psi(do([A_1, A_2], S_0)) + \cdots + \psi(do([A_1, \ldots, A_n], S_0)).$$

Assumptions like this are fundamental in microeconomics. Intuitively, only a few actions A can change the value of function ψ from state S to state $do(A, S)$, so the agent will try to execute only these "relevant" actions. Notice that, in this case, we should take into account the action preconditions, because the agent should be restricted to legal transaction lists and there is no guarantee that the transaction list T that maximizes $\Psi(T)$ will be legal. The notion of relevance that includes the action preconditions, presented in section 6.3, might be appropriate for modeling that aspect of an autonomous agent.

Modeling many agents is more complicated. Indeed, the existence of more that a one agent means that the notion of relevance of actions should consider possible transaction lists with interaction between agents, aside from the SSAs and the action preconditions.

Acknowledgments

We wish to thank Cristian Ferretti, Javier Pinto, Pablo Sáez and Mauricio Strello for stimulating conversations, advice and help. Leopoldo Bertossi visited Toronto on a couple of occasions and is grateful to Ray Reiter, Daniel Marcu, Fangzhen Lin, Richard Scherl and Aris Zakinthinos for stimulating conversations and relevant information. This research has been supported partly by DIPUC, and FONDECYT (#193–0554) grants.

References

[1] L. Bertossi, M. Arenas, and C. Ferretti. SCDBR: An Automated Reasoner for Database Updates (Extended Abstract). In *Proc. VXI International Conference of the Chilean Computer Science Society (SCCC'96), M. V. Zelkowitz and P. Straub (eds.).*

[2] Leopoldo Bertossi, Marcelo Arenas, and Cristian Ferretti. SCDBR: An Automated Reasoner for Specifications of Database Updates. *Submitted for publication*, 1996.

[3] Leopoldo Bertossi and J. Cristian Ferretti. SCDBR: A Reasoner for Specifications in the Situation Calculus of Database Updates. In *Temporal Logic. Proceedings First International Conference, ICTL '94, Bonn, Germany, July 1994, LNAI 827*, pages 543–545. Springer, 1994.

[4] Leopoldo Bertossi, Javier Pinto, Pablo Saez, Deepak Kapur, and Mahadevan 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] J. Blakeley, N. Coburn, and P. Larson. Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates. *ACM Transactions on Database Systems*, 14(3):369–400, September 1989.

[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] W. Hodges. *Model Theory.* Cambridge Univ. Press, 1993.

[8] Y. Lésperance, H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl. "A Logical Approach to High–Level Robot Programming – A Progress Report". In *Proc. AAAI Fall Symposium of Control of the Physical World by Intelligent Systems*, New Orleans, LA, November 1994.

[9] F. Lin and R. Reiter. Forget It! In *Proc. AAAI Fall Symposium on Relevance, New Orleans*, 1994.

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

[11] F. Lin and R. Reiter. How to Progress a Database II: The STRIPS Connection. In *Proc. IJCAI'95*, pages 2001–2007, 1995.

[12] 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 University Press, 1969.

[13] Javier Pinto. *Temporal Reasoning in the Situational Calculus.* PhD thesis, Department of Computer Science, University of Toronto, 1994.

[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. On Specifying Database Updates. Technical Report KRR-TR-92-3, University of Toronto, Department of Computer Science, Toronto, Canada, 1992.

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

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

[18] B. Siu and L. Bertossi. Answering Historical Queries in Databases using Relevance. *Submitted for publication*, 1996.

[19] B. Siu, L. Bertossi, and M. Arenas. Relevance via Transaction Lists. *In preparation*, 1997.

[20] Bernardo Siu and Leopoldo Bertossi. Answering Historical Queries in Databases. Technical Report RT-PUC-DCC-2-96, Pontificia Universidad Catolica de Chile, Escuela de Ingenieria, Departamento de Ciencia de Computacion, Casilla 306, Santiago 22, Chile. ftp://ftp.ing.puc.cl/pub/escuela/deptos/dcc/techReports, 1996.

[21] A. Zakinthinos. Improved Runtime Complexity for Historical Queries with Aplication to Secure Systems. Department of Computer Science, University of Toronto, 1993. Term Proyect for CSC2532.

A Proofs

In this appendix we present proofs for all the results presented in this paper.

Proofs for section 3.1

Lemma 1 *Let F, P be 0-ary fluents with the following SSAs:*

$$Poss(a,s) \supset F[do(a,s)] \equiv (a = A \wedge P[s] \wedge \varphi) \vee F[s],$$
$$Poss(a,s) \supset P[do(a,s)] \equiv a = A \vee P[s],$$

where φ is a formula that does not mention any state term. Then, P is relevant to F iff φ is satisfiable.

Proof ($\Leftarrow$) Suppose that φ is satisfiable. So, let us choose models M and M' such that: (1) $M \models \varphi$, $M' \models \varphi$, (2) They differ at S_0 only in P, that is, $M \models P[S_0]$, $M' \models \neg P[S_0]$, but the models interpret everything else identically at S_0, (3) They interpret $F[S_0]$ as false. Under these conditions it clear that $M \models F[do(A, S_0)]$ and $M' \models \neg F[do(A, S_0)]$. In consequence, P is indeed relevant to F.
($\Rightarrow$) Suppose that φ is not satisfiable, but $P \rightsquigarrow_{A_1,\ldots,A_n} F$. By definition of relevance, we have that

$$M \models F[do([A_1,\ldots,A_{n-1}], S_0)] \textit{ iff } M' \models F[do([A_1,\ldots,A_{n-1}], S_0)],$$

but

$$M \models F[do([A_1,\ldots,A_n], S_0)] \textit{ iff } M' \models \neg F[do([A_1,\ldots,A_n], S_0)],$$

However, as φ s not satisfiable, both models must interpret $F[do([A_1,\ldots,A_n], S_0)]$ the same way they interpret $F[do([A_1,\ldots,A_{n-1}], S_0)]$, a contradiction. ■

Proof of Theorem 1 We will show that if the set of triples (Σ, L, F) were recursively enumerable, then the set of first–order satisfiable formulas would be recursively enumerable, what is known to false.

Suppose that there is an algorithm that recursively enumerates the set of triples (Σ, L, F). We can use a parser to detect if the SSAs of the specification in the outputs of the algorithm have the form presented in the previous lemma. If the specification has that form, then φ is satisfiable, and the algorithm outputs φ. Clearly, this algorithm generates all the satisfiable first–order formulas. ■

Proof of Corollary 1 Immediate from Theorem 1, because every decidable problem is also recursively enumerable. ■

Proofs for section 4

Proof of Theorem 2

The proof consists in showing that, since there is a finite number of constants in the specification (no symbols for functions on objects are allowed in the fluents), then there is a finite number of different expressions that can appear in $T^i(F(\bar{c}, \bar{x}))$.

A sufficient condition for the existence of the fix–point is that there is an integer n such that $AuxT^n(F(\bar{c},\bar{x})) = \varnothing$.

Let N_A be the maximum (finite) arity of any fluent, N_C be the (finite) number of constants appearing in the fluents and N_F the (finite) number of fluents of the specification. Clearly, the (finite) number of combinations between fluents and constants and the symbol _ (denoting a variable) is $N = O(N_F \times ((N_C + N_A + 1)^{N_A}))$.

If $T^i(F(\bar{c},\bar{x}))$ does not reach a fix–point, In the worst of the cases, there is an n such that the cardinality of $T^n(F(\bar{c},\bar{x}))$ is not bigger than N. Since in this case every combination of fluents and constants has been collected, $AuxT^n(F(\bar{c},\bar{x})) = \varnothing$. ■

Proofs for section 5.1

There are some concepts and notation related to our definition of semantical relevance that will be useful.

Definition 9 *Relation $M \sim_{L,s} M'$, with M, M' models, L a set of ground tuples and s a term of sort state is true iff the following conditions hold:*

1. *M, M' share the same domain and interpret each function symbol, including actions and constants, in identical way.*
2. *For each ground tuple $F(\bar{c}) \in L$, $M \models F(\bar{c})[s]$ iff $M' \models \neg F(\bar{c})[s]$.*
3. *For each ground tuple $G(\bar{c}) \notin L$, $M \models G(\bar{c})[s]$ iff $M' \models G(\bar{c})[s]$.*
4. *For each other state–independent predicate $P(\bar{c})$, $M \models P(\bar{c})$ iff $M' \models P(\bar{c})$.*

$M \sim_{L,s} M'$ holds if M, M' interpret the set of tuples L in a different way in state s, interpreting everything else the same way in state s. It is inspired on the definition of $\sim$ presented by Lin and Reiter in [9] in their account of forgetting.

Definition 10 *Relation $M =_{L,s} M'$, with M, M' models, L a set of ground tuples and s a term of sort state is true iff the following conditions hold:*

1. *M, M' share the same domain and interpret each function symbol, including actions and constants, in identical way.*
2. *For each tuple $F(\bar{c}) \in L$, $M \models F(\bar{c})[s]$ iff $M \models F(\bar{c})[s]$.*

$M =_{L,s} M'$ holds if M, M' interpret the set L in the same way in state s. This does not restrict the way other tuples, not subsumed by L, are interpreted in s by M, M'.

Definition 11 *Relation $M \neq_{L,s} M'$, with M, M' models, L a set of ground tuples and s a term of sort state, is true iff the following conditions hold:*

1. *M, M' share the same domain and interpret each function symbol, including actions and constants, in identical way.*
2. *For each tuple $F(\bar{c}) \in L$, $M \models F(\bar{c})[s]$ iff $M \models \neg F(\bar{c})[s]$.*

This definition is very similar to that of $M \sim_{L,s} M'$, but this one allows other tuples different from $F(\bar{c})$ to be interpreted in a different way by M, M' in s.

Definition 12 (superrelevance of sets of ground tuples) *The set of ground tuples L is superrelevant to a ground tuple $F(\bar{c})$ (wrt the action theory $\Sigma = \Sigma_{bd} \cup \Sigma_{ap} \cup \Sigma_{una} \cup \Sigma_{ssa} \cup \Sigma_{S_0}$) iff there exist models M, M' and a list of ground basic action names $[a_1, \ldots, a_n]$ such that:*

1. *$M \models \Sigma^-$ and $M' \models \Sigma^-$.*
2. *$M \sim_{L,S_0} M'$.*
3. *If $S_0 < s < do([a_1, \ldots, a_n], S_0)$, then $M =_{\{F(\bar{c})\},s} M'$,*
4. *If $s = do([a_1, \ldots, a_n], S_0)$, then $M \neq_{\{F(\bar{c})\},s} M'$*

Notice that a set of ground tuples L is relevant to a ground tuple $F(\bar{c})$ iff the following conditions hold: (1) The set L is superrelevant to $F(\bar{c})$, (2) There is no proper subset L' of L that satisfy the same conditions that make L superrelevant to $F(\bar{c})$ (that is, with the same models and actions).

The definition of superrelevance of a set of tuples to another set of tuples is similar to the definition of relevance for the same kind of objects (see definition 2).

Lemma 2 *Let $F(\bar{c})$ be a tuple and L be a finite set of ground tuples such that there exist M, M' models of Σ^- with:*

1. *$M \sim_{L,S_0} M'$.*
2. *$M \neq_{\{F(\bar{c})\},do(A,S_0)} M'$*

i.e. L is "directly superrelevant" to $F(\bar{c})$ via an action A. Then, there is a set $L' \subseteq L$ such that $L' \leadsto_A F(\bar{c})$.

Proof The class $\{L' \subseteq L \mid L' \text{ is superrelevant to } F(\bar{c}) \text{ via action } A\}$ is non empty (it contains L). Since L is finite, it has a minimal element, that is non empty (the empty set cannot be relevant to anything, because in this case the models would be the same at the initial situation and the equality would be propagated to future states). Any minimal element will be relevant to $F(\bar{c})$ via A and will be a subset of L. ■

Notice that in the proof of Lemma 2 we relied on L being finite. The same proof works in the case where we have a set of ground tuples, L', instead of the singleton $\{F(\bar{c})\}$. We will see that the lemma is true when L is not necessarily finite. For this we need the following

Lemma 3 *Let L be a set of ground tuples and $F(\bar{c})$ be a ground tuple. If L is superrelevant to $F(\bar{c})$ via a basic action A, then there is finite $L' \subseteq L$, such that L' is superrelevant to $F(\bar{c})$*

Proof For each fluent symbol G of the language $\mathcal{L}$, consider the new fluent symbol $\bar{G}$ of the same type. Generate language $\bar{\mathcal{L}}$, by replacing the original fluent symbols by the new ones (and keeping the rest intact).

Let M and M' the models in the definition of superrelevance of L to $F(\bar{c})$. Consider the following sets of sentences:

$$AD(M) = \{\varphi \in \mathcal{L} \mid \varphi \text{ is ground literal and } M \models \varphi\}$$

$$AD(M') = \{\overline{\varphi} \in \overline{\mathcal{L}} \mid \varphi \text{ is ground literal in } \mathcal{L} \text{ and } M' \models \varphi\}$$

Notice that for tuples $T \in L$:

$$T \in AD(M) \Leftrightarrow \overline{T} \notin AD(M')$$

The other properties of the tuples are also reflected in these "algebraic diagrams" [7].

Consider the theory $\Delta = AD(M) \cup AD(M') \cup \Sigma^- \cup \overline{\Sigma^-}$. It holds $\Delta \models F(\bar{c})[do(A, S_0)] \wedge \neg\overline{F(\bar{c})}[do(A, S_0)]$. In fact, let N be a model of Δ. Then, $\langle \mid M \mid, (F_i^M)_i, (\bar{F}_i^{M'})_i, ...\rangle \subset N$. Since $\langle \mid M \mid, (F_i^M)_i, (\bar{F}_i^{M'})_i, ...\rangle \models F(\bar{c})[do(A, S_0)] \wedge \neg\overline{F(\bar{c})}[do(A, S_0)]$, we have $N \models F(\bar{c})[do(A, S_0)] \wedge \neg\overline{F(\bar{c})}[do(A, S_0)]$.

In consequence, there is a finite subset Δ_0 of Δ such that $\Delta_0 \models F(\bar{c})[do(A, S_0)] \wedge \neg\overline{F(\bar{c})}[do(A, S_0)]$. The set of tuples $L' = \{T \in L \mid T \text{ and } \neg\overline{T} \text{ appear in } \Delta_0\}$ is superrelevant to $F(\bar{c})$. ■

Notice that this lemma can be extended to the case $L \rightsquigarrow L'$, when L' is finite. It suffices to take the union of the finite subsets of L obtained for each $F \in L'$ as in the previous lemma.

Corollary 4 *With the hypothesis of lemma 2, but with $F(\bar{c})$ replaced by a finite set L' and L not necessarily finite, it holds that there is a finite subset L'' of L such that $L'' \rightsquigarrow_A L'$.*

Proof Use lemma 3 first, and then lemma 2. ■

Lemma 4 *Let L' be a set of ground tuples such that $L' \rightsquigarrow_{a_1,\ldots,a_n} F(\bar{c})$. Then there are $L_1, L_2, \ldots, L_{n-1}$ finite sets of tuples such that:*

$$L' \rightsquigarrow_{a_1} L_1 \rightsquigarrow_{a_2} \cdots \rightsquigarrow_{a_{n-1}} L_{n-1} \rightsquigarrow_{a_n} F(\bar{c}).$$

Proof Let M and M' be the models in the definition of $L' \rightsquigarrow_{a_1,\ldots,a_n} F(\bar{c})$. Let us consider state $do([a_1, \ldots, a_{n-1}], S_0)$. There must exist a list of fluents L'_{n-1} such that $M \sim_{L'_{n-1}, do([a_1,\ldots,a_{n-1}], S_0)} M'$, i.e. M, M' differ in L'_{n-1} in state $do([a_1, \ldots, a_{n-1}], S_0)$. If this were not the case, M and M' would coincide at state $do([a_1, \ldots, a_{n-1}], S_0)$, and then also at $do([a_1, \ldots, a_{n-1}, a_n]S_0)$, what is not possible. Clearly, L'_{n-1} is superrelevant to $F(\bar{c})$. By corollary 4, there must exist a finite list L''_{n-1} such that $L''_{n-1} \rightsquigarrow_{a_n} F(\bar{c})$ and $L''_{n-1} \subseteq L'_{n-1}$. We make $L_{n-1} = L''_{n-1}$. Repeating this same reasoning, there must exists the lists $L_{n-2}, \ldots, L_1$ mentioned above.

At this point we have $L_1 \rightsquigarrow_{a_2} L_2$. By construction, M and M' differ in L_1 at $do(a_1, S_0)$. By hypothesis, M and M' differ only in L' (at S_0). Then, L' is superrelevant with a_1 to L_1. By lemma 2 again, there is $L_0 \subseteq L'$ with $L_0 \rightsquigarrow_{a_1} L_1$.

Because $L_0 \rightsquigarrow_{a_1} L_1 \rightsquigarrow_{a_2} L_2 \rightsquigarrow \cdots \rightsquigarrow L_{n-1} \rightsquigarrow_{a_n} F(\bar{c})$ and we used the same models M and M' in the construction, we have $L_0 \rightsquigarrow_{a_1,\ldots,a_n} F(\bar{c})$. Since L' is minimal, $L_0 = L'$. ■

Corollary 5 *For L, F ground, if $L \rightsquigarrow_{a_1,\ldots,a_n} F$, then there is a finite set of tuples L' such that $L \rightsquigarrow_{a_1} L' \rightsquigarrow_{a_2,\ldots,a_n} F$.*

Proof From the previous lemma and its proof. ■

Lemma 5 *If $F'(c'), F(c)$ are ground tuples and $F'(c') \overset{n}{\rightsquigarrow} F(c)$, then $F'(c')$ is subsumed by $T^n(F(c))$.*

Proof There is a set of ground tuples L' such that $F' \in L'$ and $L' \overset{n}{\rightsquigarrow} F$.

The proof is by induction on n. When $L' \overset{1}{\rightsquigarrow} F$, with L' minimal and $F' \in L'$, F' must appear in the SSA of F (as $F'(c')$ or in more general form). If not, we could change (F')'s truth values without affecting F's truth values. (Remember that all information for F's change in truth values must be contained in the SSAs. In particular, we assumed that there are no explicit ramification constraints.)

By the definition of operator T, it captures $F'(c')$ or a more general version; so $F'(c')$ is subsumed by $T^n(F)$.

Inductive step: Let us assume that for every L' with $F'(c')$, if $L' \overset{n}{\rightsquigarrow} F(c)$, then L' is subsumed by $T^n(F(c))$. Let us now assume that $L' \overset{n+1}{\rightsquigarrow} F(c)$. By corollary 5, there is L'' such that $L' \overset{1}{\rightsquigarrow} L'' \rightsquigarrow_n^F$ (c).

In consequence, we have L'' is subsumed by $T^n(F(c))$. Now, by the same argument given in the base case, L is subsumed by $T(T^n(F(c))$, that is, L is subsumed by $T^{n+1}(F(c))$. ■

Corollary 6 *For ground tuples F, F', if $F' \rightsquigarrow F$, then F' is subsumed by $CRT(F)$.*

Lemma 6 *Let F be a tuple (non necessarily ground) and σ a ground variable substitution. Then, $CRT(F\sigma)$ is subsumed by $CRT(F)$.*

Proof The sets $CRT(F)$ and $CRT(F\sigma)$ contain exactly the same tuples, but with the variables in F replaced by their images under σ. Then, every element in $CRT(F\sigma)$ is subsumed by some element in $CRT(F)$. ■

Proof of Theorem 3 Let σ be any ground variable substitution. Then, $F'(\bar{c}', \bar{x}') \rightsquigarrow F(\bar{c}, \bar{x})$ implies that $F'(\bar{c}', \bar{x}')\sigma \rightsquigarrow F(\bar{c}, \bar{x})\sigma$. By corollary 6, $F'(\bar{c}', \bar{x}')\sigma$ is subsumed by $CRT(F(\bar{c}, \bar{x})\sigma)$. By the previous lemma, $F'(\bar{c}', \bar{x}')\sigma$ is subsumed by $CRT(F(\bar{c}, \bar{x}))$. ■

Proofs for section 5.3

Proof of Theorem 4 Let $A(\bar{c}\prime, \bar{x}')$, σ and $F(\bar{c}, \bar{x})$ be as in the hypothesis of the theorem. We have to show that $CRA(F(\bar{c}, \bar{x})$ subsumes $A(\bar{c}\prime, \bar{x}')\sigma$.

There is a fluent $F'(\bar{c}'', \bar{x}'')$ such that $F'(\bar{c}'', \bar{x}'') \rightsquigarrow F(\bar{c}, \bar{x})$ and for every substitution θ that makes $A(\bar{c}\prime, \bar{x}')$ and $F'(\bar{c}'', \bar{x}'')$ ground, there exists a model M_θ in which $A(\bar{c}\prime, \bar{x}')\theta$ changes the truth value of $F'(\bar{c}'', \bar{x}'')\theta$ from an state to the successor state (see definition 4).

In definition 4, give θ the value σ', an extension of σ, that makes $F'(\bar{c}'', \bar{x}'')$ ground. So, we have:

$$M_{\sigma'} \models F'(\bar{c}'', \bar{x}'')[S_0]\sigma\prime \quad \textit{iff} \quad M_{\sigma'} \models \neg F'(\bar{c}'', \bar{x}'')[do(A(\bar{c}\prime, \bar{x}')\sigma\prime, S_0]\sigma\prime$$

On the other side, by theorem 3, we know that $CRF(F(\bar{c}, \bar{x}))$ subsumes $F'(\bar{c}'', \bar{x}'')\sigma\prime$. Then, there is $F'(\bar{c}''', \bar{x}''') \in CRF(F(\bar{c}, \bar{x}))$ and a substitution γ such that $F'(\bar{c}''', \bar{x}''')\gamma \equiv F'(\bar{c}'', \bar{x}'')\sigma\prime$.

From the observations in the two previous paragraphs plus the syntactical form of the SSAs, we know that there are $\bar{d}$ and $\bar{y}$ such that $A(\bar{d}, \bar{y}$ appears in the SSA of $F'(\bar{c}''', \bar{x}''')$. Then, $A(\bar{d}, \bar{y}) \in CRA((F(\bar{c}, \bar{x}))$.

Finally, $A(\bar{d}, \bar{y})\gamma$ appears in the SSA of $F'(\bar{c}''', \bar{x}''')\gamma$, that is, in the SSA of $F'(\bar{c}'', \bar{x}'')\sigma\prime$. Since the action $A(\bar{c}\prime, \bar{x}')\sigma\prime$ can change the truth value of $F'(\bar{c}'', \bar{x}'')\sigma\prime$, in its SSA there must be a substitution δ such that $A(\bar{d}, \bar{y})\gamma\delta \equiv A(\bar{c}\prime, \bar{x}')\sigma\prime$. In consequence, $A(\bar{c}\prime, \bar{x}')\sigma$ is subsumed by $CRA(F(\bar{c}, \bar{x}))$ (because $\sigma\prime$ extends σ). ■

Proof of Theorem 5 Let $F(\bar{x})$ be a tuple. If (9) is the SSA for $F(\bar{x})$, then the fix–point of $F(\bar{x})$, $T^P(F(\bar{x}))$, is $\{F(\bar{x})\}$: $T^0(\{F(\bar{x})\}) = T^1(\{F(\bar{x})\}) = \{F(\bar{x})\}$. Then, the set of actions that are computationally relevant to $F(\bar{x})$ is:

$$CRA(F(\bar{x})) = \{A_1(\ldots)\sigma^*, \ldots, A_k(\ldots)\sigma^*, B_1(\ldots)\sigma^*, \ldots, B_j(\ldots)\sigma^*\}$$

(this is done by collecting the actions appearing in the SSA of $F(\bar{x})$, as shown in section 4.)

By theorem 4, the semantically relevant actions are all included among the computationally relevant ones. So, we only need to show that that every computationally relevant action is semantically relevant. In order to do this we can exhibit the semantically relevant actions and then check that every action in CRA is one of them.

By theorem 3, the set of tuples that are semantically relevant to $F(\bar{x})$ is contained in the set of tuples that are computationally relevant to $F(\bar{x})$, that is, in $\{F(\bar{x})\}$. In this case, the set of semantically relevant tuples is exactly $\{F(\bar{x})\}$, because $F(\bar{x})$ is semantically relevant to $F(\bar{x})$. Therefore, the only tuple semantically relevant to $F(\bar{x})$ is $F(\bar{c}, \bar{x})$ itself. From this, and the definition of semantically relevant action, we obtain that the only actions that are semantically relevant to $F(\bar{x})$ are are the actions that can modify the truth value of the tuple $F(\bar{x})$. Any of the (instances of) actions in the set $CRA(F(\bar{x}))$ has this property. This establishes that all the computationally relevant actions are semantically relevant. ■