Scaling up reasoning about actions using relational database technology

Giuseppe De Giacomo and Toni Mancini
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
{degiacomo,tmancini}@dis.uniroma1.it

Abstract

Reiter's variant of the Situation Calculus is tightly related to relational databases, when complete information on the initial situation is available. In particular, the information on the initial situation can be seen as a relational database, and actions, as specified by the preconditions and successor state axioms, can be seen as operations that change the state of the database. In this paper, we show how to exploit such a correspondence to build systems for reasoning about actions based on standard relational database technology. Indeed, by exploiting standard relational DBMS services, a system may be able to perform both Projection, exploiting DBMS querying services, and Progression, exploiting DBMS update services, in very large action theories. A key result towards such a realization, is that under very natural conditions Reiter's basic action theories turn out to be made of "safe formulas" (where basically negation is used as a form of difference between predicates only) and that regression and progression preserve such a safeness. This is a fundamental property to efficiently exploit relational database technology for reasoning. We then show that, even when action theories are not "safe", they can be made so while trying to retain efficiency as much as possible. Finally, we briefly discuss how such results can be extended to certain forms of incomplete information.

Introduction

Typically, in Cognitive Robotics, we assume that the cognitive agent, the robot, is equipped with a representation of the world and a specification of how its and other agents actions affect the world. There are several choices for the representation and reasoning formalism to adopt for this task. Among them the Situation Calculus as revised by Reiter and others is emerging as a general tool to express action theories (Reiter 1991; 2001).

Often in modelling the world we have to deal with incomplete information of various forms. In the Situation Calculus we typically have incomplete information on the initial situation. However, at least in certain domains, the main issue is not how to cope with incomplete information, but how to deal with hundred thousand facts that describe the current state of the world. In order to do this, we need to be able to scale up action theories, and especially to be able to reason with so many facts.

It is known that Reiter's basic action theories are tightly related to relational databases, when complete information on the initial situation is available. In particular, the information on the initial situation can be seen as a relational database, and actions, as specified by the preconditions and successor state axioms, can be seen as operations that change the state of the database (Reiter 2001; Fangzhen & Reiter 1997).

In this paper we show how to exploit such a correspondence to build systems for reasoning about actions based on standard relational database technology (cf., e.g., (Abiteboul, Hull, & Vianu 1995)). In particular, by exploiting a relational DBMS, systems may be able to perform in very large action theories both Projection, i.e., evaluate a certain (either open or closed) formula in the world resulting from executing a sequence of actions, and Progression, i.e., progress from the initial situation to the situation resulting from executing a sequence of actions. In order to perform the first task, one can exploit standard DBMS querying services, while to perform the second task one can exploit standard DBMS update services.

A key result towards such a realization, is that under very natural conditions Reiter's basic action theories turn out to be made of "safe formulas" (e.g., negation can be used as a form of difference between predicates only) and that regression and progression preserve safeness. This is a fundamental property to exploit relational database technology for reasoning, since these are the only formulas that a DBMS can evaluate. It is possible, under very general assumptions (domain closure), to transform any formula into a safe formula, however this requires the use of very large relations (suitable Cartesian products of a relation containing all domain elements), which put in serious jeopardy the ability of the DBMS to efficiently evaluate queries (formulas). In the paper, we discuss such a problem showing methods to try to retain efficiency as much as possible when transforming a formula into an equivalent safe one.

It is important to stress that the crux of this setting (i.e., using standard relational database technology) is performing reasoning through query evaluation (i.e., formula evaluation), not deduction. This can certainly be done in the case we have complete information of the initial state of the reasoning about the action system, and this is the case we will focus on in this paper. But this can also be done in case missing information can be supplied by sensors (see the notion of *just-in-time-histories* in (De Giacomo & Levesque 1999)). Our results can in principle be extended to such cases of incomplete information.

Copyright © 2004, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Finally, we observe that the ability of performing Projection allows for using a reasoning about action system based on relational technology in conjunction with high level robot language interpreters so as to exploit its formula evaluation capabilities for evaluating tests and action preconditions (Levesque *et al.* 1997; De Giacomo, Lespérance, & Levesque 2000).

Situation Calculus and basic action theories

Our account of action and change is formulated in the language of Situation Calculus (McCarthy & Hayes 1969; Reiter 2001). We will not go over the language here, except to note the following components. The language is multi-sorted, in particular we have a sort *action* for actions, a sort *state* for situations, and a sort *object* for all other terms. In fact, we may allow for specializing the sort *object*, creating sorts corresponding to the various types of objects in the domain, but, for simplicity, here we will stick to a single sort *object*. A special constant S_0 is used to denote the *initial situation*, namely the one in which no actions have yet occurred. There is a distinguished binary function symbol do where $\mathrm{do}(a, s)$ denotes the successor situation to s resulting from performing action a. Relations whose truth values vary from situation to situation are called (relational) *fluents*[1], and are denoted by predicate symbols taking a situation term as their last argument. A special predicate $\mathrm{Poss}(a, s)$ is used to state that action a is executable in situation s.

Within this language, we can formulate action theories that describe how the world changes as the result of the available actions. In this paper, we focus on Reiter's basic action theories (Reiter 1991; 2001). A basic action theory $\mathcal{D}$ is formed by:

- Axioms (denoted by $\mathcal{D}_{S_0}$) describing the *initial situation* S_0 which form the initial database D_0.
- *Action precondition axioms* ($\mathcal{D}_{ap}$) of the form:

  $$\mathrm{Poss}(a(\vec{x}), s) \equiv \phi_a(\vec{x}, s)$$

  one for each primitive action a, characterizing $\mathrm{Poss}(a, s)$.
- *Successor state axioms* ($\mathcal{D}_{ss}$) of the form:

  $$F(\vec{x}, \mathrm{do}(\alpha, s)) \equiv \gamma_F^+(\vec{x}, \alpha, s) \vee \big(F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, \alpha, s)\big)$$

  one for each fluent F, stating under what conditions $F(\vec{x}, \mathrm{do}(\alpha, s))$ holds as function of what holds in situation s. For successor state axioms the following consistency requirement must hold:

  $$\neg\exists \vec{x}, \alpha, s.[\gamma_F^+(\vec{x}, \alpha, s) \wedge \gamma_F^-(\vec{x}, \alpha, s)]$$

  which states that it is never the case that both γ_F^+ and γ_F^- hold for the same arguments. Successor state axioms take the place of the so called *effect axioms*, and provide also a solution to the frame problem (Reiter 1991). In fact they are generated automatically starting from standard effect axioms of the form:

  $$\gamma_F^+(\vec{x}, \alpha, s) \supset F(\vec{x}, \mathrm{do}(\alpha, s))$$
  $$\gamma_F^-(\vec{x}, \alpha, s) \supset \neg F(\vec{x}, \mathrm{do}(\alpha, s))$$

  by applying the so called *causal completeness assumption* that intuitively says that the first (second) effect axiom above characterize all the conditions under which action a causes F to become true (false) in the successor situation (cf. (Reiter 2001) for details).
- *Unique names axioms* ($\mathcal{D}_{una}$) for the primitive actions, plus some foundational, domain independent axioms.

In this paper we will only consider Situation Calculus formulas $\phi(s)$ that are *uniform* in s. Intuitively, $\phi(s)$ is uniform in s if it talks only about facts that hold in the situation s (cf. (Reiter 2001)).

One of the main reasoning services that a reasoning about actions system must support is solving the so-called *projection problem*: Given a formula $\phi(s)$ uniform in s, and a sequence of actions $a_1, \ldots, a_n$, determine whether a given formula $\phi(\sigma)$ is entailed by the action theory, where $\sigma = \mathrm{do}(a_n, \mathrm{do}(a_{n-1}, \cdots \mathrm{do}(a_1, S_0) \cdots))$, also written in the following as $\mathrm{do}([\alpha_1, \ldots, \alpha_n], S_0)$, in the (ground) situation resulting by performing the sequence of actions $a_1, \ldots, a_n$ from the initial situation S_0 (cf. (Reiter 2001)). That is, to decide whether:

$$\mathcal{D} \models \phi\left(\mathrm{do}([a_1, \ldots, a_n], S_0)\right).$$

One of the main results of Reiter's variant of Situation Calculus is that the projection problem can be reduced to checking whether a certain formula $\mathcal{R}[\phi(\sigma)](S_0)$ holds in the initial situation, formally:

$$\mathcal{D}_{S_0} \cup D_{una} \models \mathcal{R}[\phi(\sigma)](S_0)$$

where $\mathcal{R}[\cdot](S_0)$ is the so called *regression operator* (cf. (Reiter 2001)) which, intuitively, recursively eliminates Poss atoms in favor of their definitions in $\mathcal{D}_{ap}$, and replaces fluent atoms about $\mathrm{do}(a_i, \sigma_{i-1})$ by logically equivalent expressions about σ_{i-1} as given in $\mathcal{D}_{ss}$. Applying the regression operator to a uniform formula $\phi(\sigma)$ results into a formula uniform in the initial situation S_0, such that: $\mathcal{D} \models \phi(\sigma) \iff \mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \mathcal{R}[\phi(\sigma)](S_0)$ (Reiter 2001). Notice that the regression operator can be applied to open formulas as well, and in this case the above statement holds for all instantiations of the open formula (Reiter 2001).

Relational databases

At the highest abstraction level, a relational database is a first-order relational structure DB formed by a finite set of relations of different arities, each associated to a predicate name and formed by a finite set of tuples over a fixed domain. Such a domain can in general be infinite[2]. Unique name assumption and domain closeness is enforced on such a domain, i.e., we have a distinct constant for each object in the domain.

A first-order query $q(\vec{x})$ over a database DB is an open first-order function-free formula with free variables $\vec{x}$, whose predicates are among those associated with the relations of DB. The answer to a query $q(\vec{x})$ is the set of tuples constants (domain elements) such that

$$\{\vec{c} \mid DB \models q(\vec{c})\}$$

that is, substituting $\vec{c}$ to the free variables in $q(\vec{x})$, the (closed) formula $q(\vec{c})$ is true in DB.

[1] We do not consider functional fluents here.

[2] As long as, we limit our interest to safe query only, see later.

An important issue in databases is the concept of *safe formula* (Abiteboul, Hull, & Vianu 1995). As mentioned, relations in a database may be over an infinite domain (e.g., including the naturals, or strings). Due to this, the problem of avoiding those queries that return infinite answers arises. Suppose, as an example, to deal with a database made of a single monadic relation $R(\cdot)$ on the naturals, with extension $R(0)$, and suppose to write a query q that asks for those tuples that do not belong to R, i.e., $q = \{x|\neg R(x)\}$. The answer to the query q will be made of an infinite number of tuples, namely $\{1, 2, \ldots\}$. The same happens if we have the additional relation $S(\cdot, \cdot)$ and want to compute $q' = \{\langle x, y\rangle | R(x) \vee S(x, y)\}$.

It is usual in databases to consider queries that always return answers that are both finite and independent on the database domain. To this end, restrictions on their syntax is needed, mainly on the use of negation and disjunction. We don't go into details here, but just observe that, intuitively, negation is safe when can be written as difference, while disjunction is safe when the two operands relations have the same schema. We will present a formal definition of safe formula specialized for uniform Situation Calculus formulas on a given situation, in the next section.

Before closing the section, we need to mention that usually first-order queries in databases are not expressed in first-order logic but in relational algebra (RA). This is because relational algebra can be seen as a direct abstraction for queries written in SQL, the standard query language adopted by virtually all relational database systems. It is well known (Abiteboul, Hull, & Vianu 1995) that a strong relationship can be identified in general between FO formulas and RA expressions. In particular, Maier (1983) proposes a very simple extension to original Codd's RA, in order to make translation between FO formulas and RA expressions more effective. This extension relies on the concept of *indexed relational table*, i.e., a pair $\langle T, \vec{x}\rangle$ where T is a set of n-tuples of constants, for some n, and $\vec{x}$ is an n-tuple of distinct variables. The intuitive idea behind indexed relational tables is that they represent a first-order query $\phi(\vec{x})$ with free variables $\vec{x}$ by $\langle T, \vec{x}\rangle$, thus allowing to refer both to tuples in T and to the free variables $\vec{x}$ of $\phi(\vec{x})$, T was computed from. This extension of RA is equivalent to the original one proposed by Codd. In particular, all traditional RA operators (i.e., $\cup, \sigma, \pi, \bowtie, -, \ominus$) can be straightforwardly defined by means of operations in the domain of indexed relational tables.

Situation Calculus and relational databases

In this section we show how to relate the Situation Calculus formalism and relational database technology. To do so, we need to introduce some more constraints on the basic action theories we consider.

- We reinforce $\mathcal{D}_{una}$ by requiring unique name assumption and closure of the sort $object$. In this way we force each element of $object$ to be uniquely denoted by a single distinct constant.
- We additionally require that, for each fluent F and each tuple of objects $\vec{c}$, the theory $\mathcal{D}_{S_0}$ either logically implies $F(\vec{c}, S_0)$ or $\neg F(\vec{c}, S_0)$. That is, we have *complete information on the initial situation*. Under this assumption we have that F is characterized, in the initial situation, simply by the set of facts of the form $F(\vec{c}, S_0)$ that are logically implied by the theory, $\mathcal{D}_{S_0}$ and $\mathcal{D}_{una}$.

The above constraints are quite severe and certainly are not suitable for certain applications (Reiter 2001). However, if our application does allow us to make such assumptions, then we can use, without loss of generality, formula evaluation instead of logical implication (exchanging entailment with truthness) thus getting a quite efficient method to base reasoning on (De Giacomo & Levesque 2000; Liu & Levesque 2003).

These observations indicate that we can exploit standard relational database technology as the base of a system for reasoning about actions. In particular, we can use relational database querying techniques to perform formula evaluation, thus being able to exploit sophisticated optimization techniques developed for databases in order to deal with very large amounts of facts. Relational database technology promises to be a good vehicle to scaling up reasoning about actions (though of a limited form) to realistic domains that require dealing with a lot of information.

In order to develop such a system, we need to decide how to tackle the following issues: (i) how to represent fluents and their values as database states; (ii) how to represent uniform Situation Calculus formulas as queries on the database. We deal with each of these issues in turn.

Representing system states as database states We assume that the database does not explicitly store objects of sort *state* inside the tables, but the database itself may be considered as a "snapshot" taken of each fluent in the current situation. In other words, the database keeps track of the truth values of the fluents for all objects in the current situation.

Here we consider, as the current situation, the initial situation S_0, and represent it as a database DB_{S_0} defined as follows: each fluent $F(\vec{x}, s)$ of arity $k + 1$ defined in the action theory is associated a table F composed by k columns $(f1, ..., fk)$. Table F is populated by the k-tuples of objects that make the fluent F true in the situation S_0. That is:

$$F = \{\vec{c} \mid \mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models F(\vec{c}, S_0)\}$$

Uniform Situation Calculus formulas as queries For the moment, we concentrate on safe formulas.

Definition 1 (Safe Situation Calculus formulas). *The set of* safe Situation Calculus formulas *is the smallest set of formulas uniform in s such that:*

- *Atoms of the form $x = c$, $a(\vec{x}) = b(\vec{c})$, $F(\vec{x}, s)$ are safe, where with x and $\vec{x}$ are variables of sort object, c and $\vec{c}$ are constants of sort object, a and b are (possibly distinct) function symbols of sort action, and F is a fluent.*
- *If ϕ, ϕ' and ϕ'' are safe, then also $\phi \wedge (x_1 = x_2)$, $\phi \wedge (x_1 \neq x_2)$, $\phi' \wedge \phi''$, $\phi' \wedge \neg\phi''$ (provided that every free variable of sort object of ϕ'' is also free in ϕ'), $(\exists x)\phi$ are safe, with x_1 and x_2 variables of sort object occurring free in ϕ.*
- *If ϕ' and ϕ'' are safe, then also $\phi' \vee \phi''$ is safe, provided that the free variables of sort object in ϕ'' are the same as in ϕ'.*

Let now ϕ be a safe Situation Calculus formula uniform in S_0. Then, we can easily define a function $||\cdot||$, which maps ϕ into an indexed relational table (or, equivalently, into a relational algebra expression).

Definition 2 (The function $||\cdot||$). *Let ϕ be a safe Situation Calculus formula uniform in S_0. Then $||\phi||$ is defined as follows:*

1. *If ϕ is atomic of the form:*
   - *$x = c$, then $||\phi|| = \langle\{c\}, x\rangle$;*
   - *$a(\vec{c}) = a(\vec{x})$, then $||\phi|| = \langle\{\vec{c}\}, \vec{x}\rangle$;*
   - *$a(\vec{c}) = b(\vec{x})$, then $||\phi|| = \langle\{\}, \vec{x}\rangle$;*
   - *$F(\vec{y}, S_0)$, then $||\phi|| = \langle\{\vec{c}^{(1)}, \ldots, \vec{c}^{(n)}\}, \vec{y}\rangle$;*

   where x, $\vec{x}$ and $\vec{y}$ are variables of sort object, c, $\vec{c}$, and $\vec{c}^{(1)}, \ldots, \vec{c}^{(n)}$ are constants of sort object, a and b are distinct function symbols of sort action, and the axiom for fluent F in $\mathcal{D}_{S_0}$ is $F(\vec{x}, S_0) \equiv \vec{x} = \vec{c}^{(1)} \vee \cdots \vee \vec{c}^{(n)}$.
2. *If ϕ is of the form:*
   - *$\phi' \wedge (x = y)$ (resp. $x \neq y$), then $||\phi|| = \underset{x=y}{\sigma}(||\phi'||)$ (resp. $x \neq y$);*
   - *$\phi' \wedge \phi''$, then $||\phi|| = ||\phi'|| \bowtie ||\phi''||$;*
   - *$\phi' \wedge \neg\phi''$, provided that every free variable of sort object of ϕ'' is also free in ϕ', then $||\phi|| = ||\phi'|| \ominus ||\phi''||$;*
   - *$\phi' \vee \phi''$, provided that the free variables of sort object in ϕ'' are the same as is ϕ', then $||\phi|| = ||\phi'|| \cup ||\phi''||$;*
   - *$(\exists x)\phi'$, where ϕ' is a safe formula, then $||\phi|| = \underset{x}{\pi}(||\phi'||)$;*

   where x and y are variables of sort object.

With these definitions in place we formally express how we can make inference on the initial situation by evaluating queries over the corresponding database DB_{S_0}.

Theorem 1. *Let $\mathcal{D}_{S_0}$ and $\mathcal{D}_{una}$ be as above, and let $\phi(\vec{x}, S_0)$ be a safe Situation Calculus formula uniform in the initial situation S_0, and $Q_\phi = ||\phi(\vec{x}, S_0)||$. Then*

$$\{\vec{c} \mid \mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \phi(\vec{c}, S_0)\} = ans(Q_\phi, DB_{S_0}).$$

In other words, the answer over the database DB_{S_0} to the query modelled by the indexed table corresponding to a safe Situation Calculus formula ϕ over the initial situation S_0 are exactly all the tuples of constants that substituted to the free variables of ϕ make the resulting formula entailed from $\mathcal{D}_{S_0} \cup \mathcal{D}_{una}$, i.e., entailed by the information on the initial situation and the assumptions on constants above.

Safe basic action theories

If we aim at using formula evaluation as the reasoning engine, it is natural to assume to be able to evaluate at least those formulas for checking if an action is possible, and how a fluent changes by performing such an action. Hence in the relational setting proposed here it is natural to assume that queries for checking action precondition and effect axioms are safe. In other words, it is natural to consider basic action theories where:

- The precondition axioms have the form

$$\mathrm{Poss}(a(\vec{x}), s) \equiv \phi_a(\vec{x}, s)$$

  where $\phi_a(\vec{x}, s)$ is safe;

- The effect axioms have the form

$$\gamma_F^+(\vec{x}, \alpha, s) \supset F(\vec{x}, \mathrm{do}(\alpha, s))$$
$$\gamma_F^-(\vec{x}, \alpha, s) \supset \neg F(\vec{x}, \mathrm{do}(\alpha, s))$$

  where both $\gamma_F^+(\vec{x}, \alpha, s)$ and $\gamma_F^-(\vec{x}, \alpha, s)$ are safe.

Let us concentrate for a moment on the effect axioms. Does the safeness of the premises of the effect axioms guarantee the safeness of the corresponding successor state axiom obtained by applying the causal completeness assumption? Interestingly, the answer to this question is positive.

Theorem 2. *Let $\gamma_F^+(\vec{x}, \alpha, s) \supset F(\vec{x}, \mathrm{do}(\alpha, s))$ and $\gamma_F^-(\vec{x}, \alpha, s) \supset \neg F(\vec{x}, \mathrm{do}(\alpha, s))$ be the effect axioms for the fluent F, and let both $\gamma_F^+(\vec{x}, \alpha, s)$ and $\gamma_F^-(\vec{x}, \alpha, s)$ be safe. Then the formula*

$$\gamma_F^+(\vec{x}, \alpha, s) \vee \big(F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, \alpha, s)\big)$$

in the corresponding successor state axiom $F(\vec{x}, \mathrm{do}(\alpha, s)) \equiv \gamma_F^+(\vec{x}, \alpha, s) \vee \big(F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, \alpha, s)\big)$ is also safe.

Observe that assuming precondition and effect axioms having safe premises seems quite in the spirit of the idea of checking conditions by evaluating them on a database. It is not so for successor state axioms, since they embed a quite sophisticated technique to deal with the frame problem. The theorem above comes about as a nice and welcome surprise.

After this discussion, we feel comfortable in introducing the notion of safe basic action theory.

Definition 3 (Safe basic action theory). *A safe basic action theory is a basic action theory in which the right part of each action precondition axiom and the right part of each successor state axiom are safe formulas.*

Safe basic action theories have a very interesting property, namely that regression preserves safeness.

Theorem 3 (Regression in safe basic action theories). *Let $\mathcal{D}$ be a safe basic action theory, and $\phi(\vec{x}, \sigma)$ a safe Situation Calculus formula uniform in the (ground) situation $\sigma = \mathrm{do}([a_1, \ldots, a_n], S_0)$. Then, the regression operator $\mathcal{R}$ applied to $\phi(\vec{x}, \sigma)$, returns a formula $\mathcal{R}[\phi(\vec{x}, \sigma)](S_0)$, uniform in S_0, that is safe.*

Observe that this theorem, together with Theorem 1, gives us the mean to exploit database technology to deal with the projection problem.

Theorem 4. *Let $\mathcal{D}$ be a safe basic action theory, $\phi(\vec{x}, \sigma)$ be a safe Situation Calculus formula uniform in the situation $\sigma = \mathrm{do}([a_1, \ldots, a_n], S_0)$, $\mathcal{R}[\phi(\vec{x}, \sigma)](S_0)$ the formula on the initial situation obtained by regression, and $Q_{\phi_r} = ||\mathcal{R}[\phi(\vec{x}, \sigma)](S_0)||$. Then*

$$\{\vec{c} \mid \mathcal{D} \models \phi(\vec{c}, \sigma)\} = ans(Q_{\phi_r}, DB_{S_0}).$$

This theorem tells us how to exploit relational databases to solve the projection problem, i.e., to get all tuples of constants $\vec{c}$ such that $\phi(\vec{c}, \mathrm{do}([a_1, \ldots, a_n], S_0)$ is entailed by the action theory $\mathcal{D}$. In particular, we can proceed as follows:

1. Apply regression to get a Situation Calculus formula on the initial situation, (we are guaranteed that such a new formula is safe);

2. Transform it into an SQL query (which is immediate once the formula has been transformed into relational algebra);
3. Compute and return the answer of the resulting query over the database DB_{S_0}.

Dealing with unsafe formulas

For now we have a nice solution in case of safe formulas. What about if we cannot enforce safeness? Then Database Theory tells us that we need to enforce the finiteness of the domain, and in fact introduce an additional relation in our database corresponding to the domain itself.

Under the hypothesis of finite domain, every query can be made safe in a trivial way: as an example, the query $q = \{x|\neg R(x)\}$ can be rewritten as $q'' = \{x|U(x) \wedge \neg R(x)\}$, where the relation U is the relation that stores all domain values for x.

Observe however that the new relation U, and even more Cartesian products such as $U \times U \times \cdots \times U$ that may be involved in a query, are likely to store an huge number of tuples, thus making query answering efficiency critical.

To extend our approach to deal with unsafe formulas, we need to further reinforce the assumption of $\mathcal{D}_{una}$ and state that the sort $object$ is finite. This corresponds to introduce a new fluent $obj(\cdot, s)$, which contains initially (i.e., in S_0) a *finite* number of elements (the same in every model), each denoted by a distinct constant, and such that its successor state axiom states that it does not change with actions. Predicate $obj(\cdot, s)$ defines the extension of the sort $object$. To reflect such an assumption, we introduce in the database DB_{S_0} a relation `Obj` which denotes exactly the objects in $obj(\cdot, s)$.

With $obj(\cdot, s)$ and `Obj` in place, it is straightforward to make every uniform Situation Calculus formula safe. However caution must be put in order to retain al least some efficiency of query answering once we reduce reasoning to query evaluation on the initial database.

The problem is that, to make a query safe, we are forced to introduce occurrences of the `Obj` table, which is typically formed by a huge number of tuples. Unfortunately, we cannot in general overcome this, but can often carefully choose, among the different alternatives, the best way to translate an unsafe uniform Situation Calculus formula into a query on the current database.

Of course, several functions $||\cdot||'$ can be defined that reduce to $||\cdot||$ given in Definition 2 when dealing with safe formulas, and several heuristics can be defined for choosing the most efficient translation. In this paper, we give a possible heuristic, under the (reasonable) assumption that the size of the `Obj` relation is much larger than the size of any other relation in the database. Under this assumption, our heuristic allows us to choose at each step whether to apply De Morgan laws to our formula or not, by computing a *cost* for each alternative. Let the `Obj` relation consists of n tuples: the cost of evaluating a RA formula is given by the sum of $n^{|\vec{x}|}$ factors, one of each time a table $\langle \underbrace{Obj \times \cdots \times Obj}_{|\vec{x}| times}, \vec{x}\rangle$ (denoted in the following as $\langle Obj^{|\vec{x}|}, \vec{x}\rangle$) has to be built. Moreover, the cost is given in an incremental fashion, i.e., by considering constant the cost of evaluating subformulas. In this way, we are allowed to independently choose at each level of the formula tree whether to apply De Morgan laws or not.

In the following, both $\phi_1(\vec{x}, \vec{y}, S_0)$ and $\phi_2(\vec{y}, \vec{z}, S_0)$ are uniform Situation Calculus formulas in the initial situation S_0, which do not have '$\neg$' as the outermost operator, and such to have $\vec{y}$ as common free variables of sort *object*. Moreover, ϕ_1 has $\vec{x}$ as additional free variables of sort *object* that not occur in ϕ_2, and ϕ_2 has $\vec{z}$ as additional free variables of sort *object* not occurring in ϕ_1. We describe in detail only one case, the others are analogous.

Let the Situation Calculus formula $\phi(\vec{x}, \vec{y}, \vec{z}, S_0)$ to be translated be

$$\neg\left(\phi_1(\vec{x}, \vec{y}, S_0) \vee \phi_2(\vec{y}, \vec{z}, S_0)\right). \tag{1}$$

We identify two possible alternatives for translating it: the first is to translate it directly, the other is to apply De Morgan laws in order to obtain the equivalent formula

$$\neg\phi_1(\vec{x}, \vec{y}, S_0) \wedge \neg\phi_2(\vec{y}, \vec{z}, S_0). \tag{2}$$

Translation of option (1) is the following:

$$\begin{aligned} ||\phi||' \;=\; ||\neg(\phi_1 \vee \phi_2)||' = \langle Obj^{|\vec{x}|+|\vec{y}|+|\vec{z}|}, \vec{x}; \vec{y}; \vec{z}\rangle - \\ \Big(||\phi_1(\vec{x}, \vec{y}, S_0)||' \times \langle Obj^{|\vec{z}|}, \vec{z}\rangle \cup \\ \langle Obj^{|\vec{x}|}, \vec{x}\rangle \times ||\phi_2(\vec{y}, \vec{z}, S_0)||'\Big) \end{aligned} \tag{3}$$

while translation of option (2) is:

$$\begin{aligned} ||\phi||' \;=\; ||\neg\phi_1 \wedge \neg\phi_2||' = \\ \Big(\langle Obj^{|\vec{x}|+|\vec{y}|}, \vec{x}; \vec{y}\rangle - ||\phi_1(\vec{x}, \vec{y}, S_0)||'\Big) \bowtie \\ \Big(\langle Obj^{|\vec{y}|+|\vec{z}|}, \vec{y}; \vec{z}\rangle - ||\phi_2(\vec{y}, \vec{z}, S_0)||'\Big). \end{aligned} \tag{4}$$

Assuming that RA expressions $||\phi_1||'$ and $||\phi_2||'$ have already been computed, the cost for formula (3) is $n^{|\vec{x}|+|\vec{y}|+|\vec{z}|} + n^{|\vec{x}|} + n^{|\vec{z}|}$, while that for formula (4) is $n^{|\vec{x}|+|\vec{y}|} + n^{|\vec{y}|+|\vec{z}|}$. If we assume for simplicity that $|\vec{x}| \approx |\vec{y}| \approx |\vec{z}|$, the cost of the second alternative is lower, so the formula is rewritten by pushing negation in.

Using these calculations, in the full paper we rewrite all kind of unsafe formulas in order to achieve the most efficient translation (i.e., the one with less occurrences of Cartesian products of the `Obj` table). Then, we define the function $||\cdot||'$ by firstly applying our heuristic, and then by inserting suitable Cartesian products of the `Obj` table. Details are omitted for lack of space and will be given in the full paper.

Performing actions and changing the database

Normally, we are interested in querying the current database state, possibly exploiting regression, in order to make some projection on the future. On the other hand, from time to time we want to perform progression, i.e., to transform the action theory by updating the current situation according to the results of performing a given sequence of actions. This results in changing the database content, in order to reflect this new situation.

Under the assumptions we have made (i.e., unique name assumption, closure of the object domain, and complete information on the initial situation), results in (Fangzhen & Reiter 1997) show that progression, which is very difficult in general, becomes simple. Moreover, relational technology can potentially allow to perform progression by exploiting the standard update mechanisms developed for databases,

and take advantage of the transactional support to guarantee consistency, even in case of a failure during the progression.

With these observations in mind, we develop an account of progression that is based on the relational setting proposed so far. In order to do this, we need to clarify how the successor state axioms can be exploited as a specification of the updates to perform.

Let the successor state axiom for the fluent F be:

$$F(\vec{x}, do(\alpha, s)) \equiv \gamma_F^+(\vec{x}, \alpha, s) \vee F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, \alpha, s).$$

We can extract from it *effect axioms* instantiated for each (ground) action a:

$$\gamma_F^-(\vec{x}, a, s) \supset \neg F(\vec{x}, do(a, s))$$
$$\gamma_F^+(\vec{x}, a, s) \supset F(\vec{x}, do(a, s)).$$

In order to change the database accordingly, assuming `GammaFminus_a`, and `GammaFplus_a` being the tables corresponding to $\gamma_F^-(\vec{x}, a, s)$ and $\gamma_F^+(\vec{x}, a, s)$ respectively, we need to perform the following SQL commands as a unique transaction:

```
delete from F where exists (
  select x1, ..., xn from GammaFminus_a
  where x1 = f1 and ... and xn = fn );

insert into F (
  select x1 as f1, ... xn as fn
  from GammaFplus_a );
```

That is, a tuple is contained in table F after the update associated with action a if and only if, either the update has inserted it in the table, or the tuple was already in the table and the update has not deleted it. This clearly corresponds to what specified by the successor state axiom instantiated for action a. Since the following consistency requirement holds:

$$\neg\exists\vec{x}, s.[\gamma_F^+(\vec{x}, a, s) \wedge \gamma_F^-(\vec{x}, a, s)],$$

tables `GammaFplus_a` and `GammaFminus_a` are disjoint. Hence, exchanging the order of the two statements would not change the result.

Once we have defined the above updates, by exploiting the result on progression mentioned above we get the following theorem:

Theorem 5. *Let $\mathcal{D}$ be a safe basic action theory, with $\mathcal{D}_{S_0}$ and $\mathcal{D}_{una}$ defined as usual. Let $\phi(\vec{x}, \sigma)$ be a safe Situation Calculus formula uniform in the (ground) situation $\sigma = \mathrm{do}([a_1, \ldots, a_n], S_0)$, DB_σ be the database obtained from DB_{S_0} by performing the updates corresponding to each action a_i in σ in order, and $Q_\phi = ||\phi(\vec{x}, S_0)||$. Then*

$$\{\vec{c} \mid \mathcal{D} \models \phi(\vec{c}, \sigma)\} = ans(Q_\phi, DB_\sigma).$$

In other words, in order to evaluate a formula on the situation resulting from executing a sequence of actions, we can update the database, using the above update commands, and evaluate the formula on the resulting database. That is, the above update commands can be used to *progress*, in the sense of (Fangzhen & Reiter 1997), the database and hence the action theory it represents.

It is worth noting that the transactional support of the relational DBMS can be exploited to avoid leaving the database in an undesired state in case the progression fails, by issuing a *commit* statement only when the progression is successfully completed. Actually, it can be exploited further. In those cases where regression generates a query that is too big and complex to be efficiently evaluated, progression can be used as an alternative, with the proviso that after the evaluation, the transaction corresponding to the progression gives an explicit *roll back* command to restore the initial situation. In fact, this technique has shown to be quite effective in practice in dealing with projections involving very long sequences of actions. We also observe that state-of-the-art DBMSs do most of their computations in main memory cashes and not on disks directly, so the updates mentioned so far are usually performed in a fast way in main memory before issuing the *commit* statement.

Conclusions

In this paper we have described how reasoning about actions, under complete information, can be potentially scaled up exploiting standard relational technology. A prototype of such a system, based on the implementation work reported in (De Giacomo & Palatta 2000), is currently under construction, and first experimental results are very promising.

It is important to stress that more advanced form of reasoning that can still rely on formula evaluation can potentially take advantage from the relational technology proposed here. Along this line, we are currently working on extending such an approach to reasoning with incomplete information when missing information can be supplied by sensors as in the setting of (De Giacomo & Levesque 1999).

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison Wesley Publ. Co.

De Giacomo, G., and Levesque, H. 1999. Projection using regression and sensors. In *Proc. of IJCAI'99*, 160–165.

De Giacomo, G., and Levesque, H. 2000. Two approaches to efficient open-world reasoning. In *Logic-based artificial intelligence*. Kluwer. 59–78.

De Giacomo, G., and Palatta, F. 2000. Exploiting a relational dbms for reasoning about actions. In *Proc. of CogRob'00*.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. 2000. ConGolog, a concurrent programming language based on the Situation Calculus. *Artificial Intelligence* 121(1–2):109–169.

Fangzhen, L., and Reiter, R. 1997. How to progress a database. *Artificial Intelligence* 92:131–167.

Levesque, H.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *J. of Logic Programming* 31:59–84.

Liu, Y., and Levesque, H. 2003. A tractability result for reasoning with incomplete first-order knowledge bases. In *Proc. of IJCAI'03*, 83–88.

Maier, D. 1983. *The Theory of Relational Databases*. Computer Science Press, Rockville, Maryland.

McCarthy, J., and Hayes, P. J. 1969. Some philosophical problems from the standpoint of aritificial intelligence. *Machine Intelligence* 4:463–502.

Reiter, R. 1991. The frame problem in the situation calculus. In *Artificial Intelligence and Mathematical Theory of Computation*. Academic Press. 359–380.

Reiter, R. 2001. *Knowledge in action: Logical foundations for describing and implementing dynamical systems*. MIT Press.