SCDBR: An Automated Reasoner for Database Updates (Extended Abstract)

Leopoldo Bertossi, Marcelo Arenas and Cristian Ferretti
Pontificia Universidad Catolica de Chile
Escuela de Ingenieria, Dept. de Ciencia de Computacion
Casilla 306, Santiago 22, Chile. E-mail: {bertossi,marenas,cfs}@ing.puc.cl

Abstract

In this paper we describe SCDBR, a system that automates reasoning with specifications of database updates written in the situation calculus, a first–order language originally proposed by John McCarthy for reasoning about actions and change. The specifications are expected to be given according to Ray Reiter's solution to the frame problem. SCDBR is written in PROLOG, and among other features is able to verify that integrity constraints are satisfied by the specification and to answer historical queries.

Keywords: Specifications of Database Updates, Knowledge Representation, Automated Reasoning, Integrity Constraints, Situation Calculus.

1 Introduction

We apply logic programming and automated reasoning techniques for reasoning with/from a formal specification of the dynamics of a database and the knowledge stored in the database. The specification and the database contents are given in first–order logic with an extra second–order induction principle that allows, among other things, to define an accessibility relation between states and to prove database *integrity constraints* (ICs). More precisely, we describe, SCDBR (for *situation-calculus-based database reasoner*), an automated reasoning system we have developed that has such reasoning capabilities[1].

In general, theories of action and change, such as the specification of the evolution of a database, include sentences stating the changes that are introduced in some properties of the world when actions are executed. In most cases, it is desirable to describe the changes in the world that are caused by the executions of actions, but omitting explicit sentences that describe the properties that do not change. The problem of inferring non-change from a logical specification describing change is called the *frame problem* [12].

The basis of our work is Reiter's solution [15] to the frame problem in the *situation calculus* [12], a language of many–sorted first–order logic. The situation calculus contains atomic dynamic properties called fluents, which are predicates with a state argument (think of tables in a database). A specification of the dynamics of a database contains axioms stating the preconditions for executing the named transactions. When an action satisfies its precondition at a given state, we say it is *possible* (or *legal*) at that state.

[1] A preliminary report on SCDBR appeared in [3]. See also [2] for a more detailed description.

In case a preliminary specification of a database describes also the effects of actions (think of atomic transactions in databases) on fluents, then Reiter proposes to materialize at the same object level the assumption that those *effects axioms* must explain all the changes in the fluents when an action is performed. In this way, he derives from the effect axioms the so–called *successor states axioms* that fully characterize when a given fluent becomes true at the successor state, resulting from executing an arbitrary legal action at an arbitrary state.

Furthermore, the preliminary specification might consider a set of ICs, which describe certain conditions that the database must always satisfy [2].

Our system, SCDBR, automatically derives the final database specification of the database dynamics, given in terms of successor state axioms, from a preliminary specification given in terms of effect axioms. The specifications handled by the system are fully first-order. SCDBR is also able to perform other reasoning tasks like: (1) Deciding the legality of sequences of transactions of the form $do(\alpha_1, do(\alpha_2, do(\alpha_3, \ldots) \ldots)))$; (2) "Regressing" queries posed to a virtually updated database to a query about the initial database state. This task could also be seen as a form of hypothetical reasoning; (3) Interfacing with theorem provers and database systems like: ORACLE, OTTER [13], PROLOG, RRL [6], CLP(FD) [5]. Among other things, those systems can be used for proving statements about the initial database; (4) Answering historical queries, when the specification plus a sequence of transactions (executed or to be executed) is given (following the procedures described in [18, 19]); (5) Mechanically modifying the preliminary specification in order to subsume in the final specification a given integrity constraint (in some cases, this is possible [9, 14]); (6) Physical updating of the database from a virtual update [8, 10] (i.e. from the initial database, the successor state axioms and a given sequence of transactions);(7) Automated proofs of integrity constraints. For this, we interfaced the system with OTTER. For this task, we have also used RRL, which has turned out to be very useful for generating appropriate induction schemas [1, 4].

The system is written in SICSTUS PROLOG, which is used only as a programming language; and can be run on the top of a database management system, like ORACLE, adding new and important functionalities to traditional DBMS.

2 Reiter's Solution to the Frame Problem and Database Updates

Characteristic ingredients for a situation calculus language $\mathcal{L}$, besides the usual ingredients of first-order languages, are: (1) Sorts *action*, *state*, and *object*; (2) Predicate symbols of the sort $object \times \ldots \times object \times state$ to denote fluents. These are predicates that depend on the state of the world; (3) Operation symbols of the sort $object \times \ldots \times object \rightarrow action$ for denoting actions with objects as parameters (or applied to objects); (4) A constant S_0 to denote the initial state; (5) An operation symbol *do* of sort $action \times state \rightarrow state$, that executes an action at a given state, producing a successor state. In this language there are variables for individuals of each sort, so it is possible to quantify over objects, actions, and situations.

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

[2] In [16, 17], Reiter starts directly from specifications of database updates given in terms of successor state axioms. Those specifications do not contain explicit ICs.

The immediate (positive or negative) effects of actions when they are possible to apply, in terms of the fluents whose truth values we know are changed by the execution of the action.

In Reiter's formalism, the knowledge contained in items (1) and (2) above is considered the initial database D_{S_0}. The information given in item (3) is formalized by means of an precondition axiom of the form $Poss(A(\vec{x}), s) \equiv \pi_A(\vec{x}, s)$, for each action name A. Finally, item (4) is expressed by effect laws for pairs (transaction, fluent):

Positive Effects Laws: For some pairs formed by a fluent F and an action name A, an axiom of the form:

$$Poss(A(\vec{x}), s) \wedge \varepsilon_F^+(\vec{x}, \vec{y}, s) \supset F(\vec{y}, do(A(\vec{x}), s)). \tag{1}$$

Intuitively, if action A is possible, and the preconditions for fluent F represented by the metaformula $\varepsilon_F^+(\vec{x}, \vec{y}, s)$ is true at state s, then fluent F becomes true at the successor state $do(A(\vec{x}), s)$ obtained after execution of action A at state s. Here, $\vec{x}, \vec{y}$ are parameters for the fluent and action.

Negative Effects Laws: For some pairs formed by a fluent F and an action name A, an axiom of the form:

$$Poss(A(\vec{x}), s) \wedge \varepsilon_F^-(\vec{x}, \vec{y}, s) \supset \neg F(\vec{y}, do(A(\vec{x}), s)). \tag{2}$$

This is the case where action A makes fluent F to become false in the successor state.

Example: Let us consider a library database. For simplicity, every book that can be ordered or appears in the initial database will be assumed to be included in a global (constant) table, $BooksInPrint$, which may be interpreted as a predicate. In a more realistic example, one should interpret it as a fluent because it can change over time. Initially, there are classified and unclassified books. Those that are classified have an id number assigned which includes the ISBN number of the book and the copy number (so that different copies of the same book have different id numbers). The unclassified books are those that were ordered and have not yet been classified. Finally, there is a fluent (table) which records the number of copies available for each book that is classified, $Stock$. More precisely, these are the basic ingredients of the language: (1) A predicate, $BooksInPrint(isbn, title, author, editor, year, edition)$; (2) Fluents (tables): $Classified(isbn, id, s)$, $SoldOut(isbn, s)$, $LostBook(id, s)$, $Unclassified(isbn, copies, s)$, $Stock(isbn, quantity, s)$; (3) Update actions (atomic transactions): $deleteBook(id)$ (decreases $Stock$ by one and only when the $quantity$ is 0 it deletes the book from $Classified$, $Stock$ and $LostBook$ tables), $classifyBook(isbn, id)$ (assigns an id for the book in $Unclassified$ and removes the book from $Unclassified$ and inserts it in $Classified$, it also updates $Stock$ according to the number of $copies$), $order(isbn, copies)$ (adds an item that is not $SoldOut$ in $BooksInPrint$ to $Unclassified$).

Initial Database [3]**:**

- $BooksInPrint('3{-}540{-}18199{-}7', 'Foundations\ of\ Logic\ Programming',$
  $'Lloyd,\ J.W.', 'Springer', '1987', '2')$
- $BooksInPrint('0{-}7167{-}8162{-}X', 'Database\ and\ Knowledge{-}Base\ Systems',$
  $'Ullman,\ J.D.', 'Computer\ Science\ Press', '1989', '1')$
- $BooksInPrint('0{-}412{-}14930{-}3', 'The\ Theory\ of\ Computer\ Science',$
  $'Brady,\ J.M.', 'A\ Halsted\ Press\ Book', '1977', '1')$
- $Classified(isbn, id, S_0) \equiv (isbn =' 0{-}412{-}14930{-}3' \wedge id =' 10') \vee$
  $(isbn =' 0{-}412{-}14930{-}3' \wedge id =' 11') \vee$
  $(isbn =' 0{-}7167{-}8162{-}X' \wedge id =' 12')$

[3] Notice that the formulas in this database represent a relational database.

- $Stock(isbn, int, S_0) \equiv (isbn =' 0{-}412{-}14930{-}3' \wedge int = 2) \vee$
$(isbn =' 0{-}7167{-}8162{-}X' \wedge int = 1) \vee$
$(isbn =' 3{-}540{-}18199{-}7' \wedge int = 0)$
- $Unclassified(isbn, int, S_0) \equiv (isbn =' 0{-}412{-}14930{-}3' \wedge int = 3) \vee$
$(isbn =' 0{-}7167{-}8162{-}X' \wedge int = 2) \vee$
$(isbn =' 3{-}540{-}18199{-}7' \wedge int = 3)$

Action Preconditions:

- $Poss(deleteBook(id), s) \equiv LostBook(id, s)$
- $Poss(order(isbn, copies), s) \equiv (\exists title, author, editor, year, edition)$
$BooksInPrint(isbn, title, author, editor, year, edition) \wedge copies > 0$
- $Poss(classifyBook(isbn, id), s) \equiv \exists copies\ Unclassified(isbn, copies, s) \wedge$
$\neg \exists isbn'\ Classified(isbn', id, s)$

Effect Laws:

- $Poss(deleteBook(id), s) \supset \neg LostBook(id, do(deleteBook(id), s))$
- $Poss(deleteBook(id), s) \supset \neg Classified(isbn, id, do(deleteBook(id), s))$
- $Poss(deleteBook(id), s) \wedge Classified(isbn, id, s) \wedge Stock(isbn, quantity, s) \supset$
$\neg Stock(isbn, quantity, do(deleteBook(id), s))$
- $Poss(deleteBook(id), s) \wedge Classified(isbn, id, s) \wedge Stock(isbn, quantity, s) \wedge$
$quantity > 1 \wedge quantity' = quantity - 1 \supset$
$Stock(isbn, quantity', do(deleteBook(id), s))$
- $Poss(order(isbn, copies), s) \wedge \neg SoldOut(isbn, s) \wedge ((Unclassified(isbn, copies', s) \wedge$
$copies''' = copies' + copies) \vee$
$(\neg \exists copies''\ Unclassified(isbn, copies'', s) \wedge copies''' = copies)) \supset$
$Unclassified(isbn, copies''', do(order(isbn, copies), s))$
- $Poss(order(isbn, copies), s) \wedge \neg SoldOut(isbn, s) \wedge Unclassified(isbn, copies', s) \supset$
$\neg Unclassified(isbn, copies', do(order(isbn, copies), s))$
- $Poss(classifyBook(isbn, id), s) \wedge Unclassified(isbn, copies, s) \wedge copies > 1 \wedge$
$copies' = copies - 1 \supset$
$Unclassified(isbn, copies', do(classifyBook(isbn, id), s)$
- $Poss(classifyBook(isbn, id), s) \wedge Unclassified(isbn, copies, s) \supset$
$\neg Unclassified(isbn, copies, do(classifyBook(isbn, id), s))$
- $Poss(classifyBook(isbn, id), s) \wedge ((Stock(isbn, quantity, s) \wedge quantity' = quantity + 1) \vee$
$(\neg \exists quantity\ Stock(isbn, quantity, s) \wedge quantity' = 1)) \supset$
$Stock(isbn, quantity', do(classifyBook(isbn, id), s))$
- $Poss(classifyBook(isbn, id), s) \wedge Stock(isbn, quantity, s) \supset$
$\neg Stock(isbn, quantity, do(classifyBook(isbn, id), s))$
- $Poss(classifyBook(isbn, id), s) \supset Classified(isbn, id, do(classifyBook(isbn, id), s)$

□

In Reiter's solution to the frame problem [15], for each fluent F we combine its positive effects laws and its negative effects laws, building one general positive axiom and one general negative axiom for F. After that, we suppose that these general effect axioms, both positive and negative, for a given fluent F, are the only causes for changing the fluent's F truth value from a state to a successor state. Actually, for each fluent F we generate its **Successor State Axiom** (SSA):

$$Poss(a, s) \supset [F(do(a, s)) \equiv \gamma_F^+(a, s) \vee (F(s) \wedge \neg \gamma_F^-(a, s))]. \quad (3)$$

This axiom is tacitly universally quantified over actions. Intuitively, if action a is possible, then fluent F is true at the successor state if and only if a is one of the actions leading to F's truth

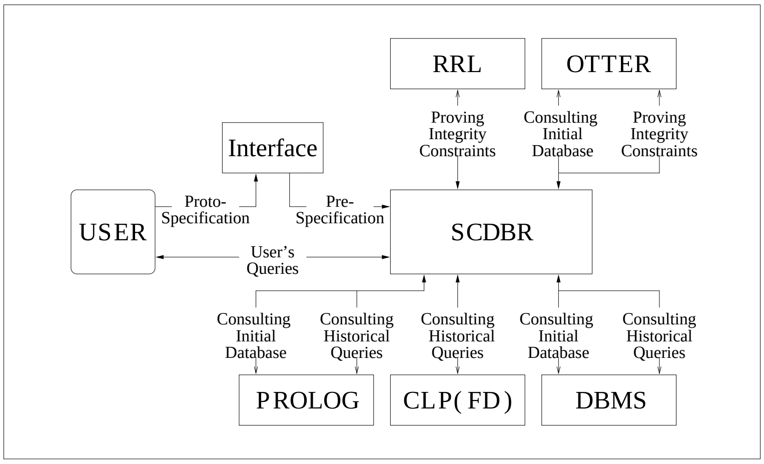

Figure 1: SCDBR's architecture.

(for which the fluent preconditions are true), or F was already true when the action was executed and it is not one of the actions leading to F to be false (or the fluent preconditions were not true for those actions). For example, if we consider the fluent $Unclassified$ in our library database, we obtain the following SSA:

- $Poss(a,s) \supset Unclassified(isbn, quantity, do(a,s)) \equiv$

$$\begin{aligned}
&[\exists copies\ (a = order(isbn, copies) \wedge \neg Soldout(isbn, s) \wedge \\
&(\exists quantity'\ (Unclassified(isbn, quantity', s) \wedge quantity = quantity' + copies) \vee \\
&\neg\exists quantity'\ (Unclassified(isbn, quantity', s) \wedge quantity = copies))) \vee \\
&\exists id\ (a = classifyBook(isbn, id) \wedge \exists quantity'\ (Unclassified(isbn, quantity', s) \wedge \\
&quantity' > 1 \wedge quantity = quantity' - 1))] \vee \\
&Unclassified(isbn, quantity, s) \wedge \neg[\exists copies\ (a = order(isbn, copies) \wedge \\
&\neg Soldout(isbn, s) \wedge Unclassified(isbn, quantity, s)) \vee \\
&\exists id\ (a = classifyBook(isbn, id) \wedge Unclassified(isbn, quantity, s))]
\end{aligned}$$

In order for things to work properly, it is necessary to add *unique names axioms* for actions and states: (1) For distinct action names A and A', $A(\vec{x}) \neq A'(\vec{y})$; (2) For each action name A, $A(x_1, \ldots, x_n) = A(y_1, \ldots, y_n) \supset x_1 = y_1 \wedge \cdots \wedge x_n = y_n$; (3) $S_0 \neq do(a,s)$; (4) $do(a,s) = do(a',s') \supset a = a' \wedge s = s'$.

3 An Overview of SCDBR

Our system implements Reiter's solution and takes advantage of the generated specifications for performing different reasoning tasks. For this purpose, the system is able to call relational database management systems and theorem provers. The SCDBR architecture is shown in figure 1.

The user's database is given to the system in a convenient format (proto–specification) by means of an external interface, which translates this information into SCDBR notation (pre–specification). Next, the system automatically derives the final database specification in terms of SSAs. Some reasoning tasks performed by SCDBR are:

1. In order to reason about the initial database, SCDBR can follow two different approaches: (1) If the initial database is relational, it uses SQL or PROLOG as query languages; (2) If

the initial database is first–order, it uses OTTER as theorem prover.

2. Temporal projection: Given a query referred to a future state, $do(A_n, do(\ldots do(A_1, S_0)\ldots)$, SCDBR is able to generate a new equivalent query about the initial state, by *regressing* the original query to the initial state. The *legality* of the list of transactions $A_1, \ldots A_n$ can be previously checked by SCDBR.

3. Given a transaction α, SCDBR is able to physically update the database, incorporating the changes due to the execution of α.

4. Given a sequence of transactions to be performed at the initial database, SCDBR can reason about the properties that hold in the generated intermediate states, answering historical queries of the kind *"Has a given property been true in all intermediate states of the database"* or *"Has a given property been true in some intermediate state of the database"*, etc...

5. SCDBR can handle Integrity Constraints (IC) in two ways: (1) Proving, from the specification, that a given IC holds in all (legal) states of a database. (2) If an IC is not entailed by the specification, in some cases, SCDBR modifies the preliminary specification in order to subsume the IC.

4 Description of the SCDBR Functionalities

The user's proto specification, given to the system by means of an external interface is the following: (1) The similarity type of the first–order language, in particular, names for fluents, predicates, actions and distinguished individuals; (2) For each action, a specification of its precondition, direct effects on some fluents with their polarity (true or false); (3) A set of formulas about the initial state (the initial database).

The external interface translate this information into SCDBR notation (logical formulas are written in prefix notation). The formulas of language $\mathcal{L}$ are internally represented and manipulated as ground terms of PROLOG. That means, $\mathcal{L}$ is the representation or specification language whose formulas are processed by procedures written in PROLOG.

In order to generate the formulas of the final specification in terms of SSAs, SCDBR relies on an essential and intensive use of unification. Take for example, the successor state axiom (3). For each fluent F, we know the general syntactic form of the axiom. This structure is represented by the procedure defined in PROLOG (notice that the formula in this procedure is written in prefix notation):

```
ssa( F, all( a, implies( poss( a, STATE_VAR ),
                              iff( FSUC, or( PT, and( FACT, not( NT ))))))) :- ...   (4)
```

Here `F`, `FSUC`, `PT`, `FACT`, `NT` stand for the fluent, the fluent at the successor state, the positive gamma formula for F, the fluent at the current state, and the negative gamma formula for F, respectively. All of them appear as PROLOG variables in the arguments of the procedure `ssa` in (4). The intended use of this procedure is the following: given an instantiation of F, a fluent name, we ask for its successor state axiom `A`, by posing the query `? |- ssa(fluent-name, A)`. By unification, `A` will turn out to be a PROLOG term representing a formula, with concrete values for `FSUC`, `PT`, `FACT`, `NT`, obtained from the body of clause (4). The formulas for `PT`, `NT`, can be obtained from procedures in the body of clause (4), which are defined in the program in a similar form in terms of other PROLOG procedures that directly encode the information provided by the user. For example, with `| ?- ssa(lost_book,F).`, we can ask the system to construct the successor state axiom for the fluent *LostBook*, obtaining the answer

```
F = forall(a) : (poss(a,s) => (lost_book(id1, do(a,s)) <=> lost_book(id1,s) & neg
a eq delete_book(id1)))
```

By means of the `una` predicate, SCDBR computes the unique names axioms for actions and states (these axioms are used in several tasks, e.g. temporal projection, proofs of integrity constraints, etc). For example, let us consider the library specification shown in section 2. In order to calculate the unique names axioms for this DB specification, we execute an instruction of the form `| ?- una(U).` A part of the answer looks like this:

```
UI = [neg delete_book(id1) eq order(isbn1,int1),...,do(a, s) eq do(a1,s1) => a eq
      a1 & s eq s1]
```

Given an specification with SSAs of the form $F(x_1, x_2, \ldots, x_n, do(a, s)) \equiv \Phi_F(x_1, x_2, \ldots, x_n, a, s)$, with formula $\Phi_F(x_1, x_2, \ldots, x_n, a, s)$ not mentioning the state $do(a, s)$, the *regression operator* [16], R, applied to a formula φ, computes a new formula in which each occurrence of a fluent of the form $F(t_1, t_2, \ldots, t_n, do(\alpha, \delta))$ is replaced by $\Phi_F|^{x_1,x_2,\ldots,x_n,a,s}_{t_1,t_2,\ldots,t_n,\alpha,\delta}$. Since this operator is defined by recursion, its specification in the system is simple. The operator recursively descends until it finds fluents on which the substitutions can be performed. For example, we can apply the regression operator to a formula in this way:

```
| ?- reg(lost_book('0-7167-8162-X', do(delete_book('0-412-14930-3'), s0)),R).
R = lost_book('0-7167-8162-X',s0) & neg delete_book('0-412-14930-3') eq
    delete_book('0-7167-8162-X')
```

There are three Pruning Operators in SCDBR, which allow to simplify formulas on the basis of unique name axioms for actions, states and objects. For example, we want to simplify according to the UNA for actions and objects, the formula that we obtained in the previous section with the regression operator. For this we use the predicates `prune_una` and `prune_uno`:

```
| ?- i_p(lost_book('0-7167-8162-X',s0) & neg delete_book('0-412-14930-3') eq
     delete_book('0-7167-8162-X'),F),prune_una(F,A),prune_uno(A,B).
A = (lost_book('0-7167-8162-X',s0) & neg '0-412-14930-3' eq '0-7167-8162-X'),
B = lost_book('0-7167-8162-X',s0)
```

Here predicate `i_p` transforms infix notation into prefix notation that is needed by the pruners (there is another predicate, `p_i`, that does the reciprocal translation).

With SCDBR, first–order queries can be transformed into PROLOG clauses, so that we can also use PROLOG as a query language for a relational database. This procedure is called mainly when we want to verify if a given formula, instantiated in the initial state, is true in the initial database, what is very useful in some tasks, e.g. proofs of integrity constraints, verification of legality of sequences of actions, etc.

The predicate `prolog_initial` is called by the system for asking queries to the initial (relational) database. Although any formula of the form $\varphi(S_0)$, with S_0 as the only state term in it, is accepted as a query, the procedure should be provided with *safe* formulas in order to avoid floundering queries and infinite loops when the clauses obtained are run with PROLOG [11]. For example, with the initial database of the specification presented in section 2, we wish the system to give us a list of identifiers *isbn* of those books that have at least one classified copy and one unclassified copy at the initial database. In first–order logic, this query has the form

$$\exists id\ Classified(isbn, id, S_0)\ \wedge\ \exists quantity\ (Unclassified(isbn, quantity, S_0)\ \wedge\ quantity > 0) \quad (5)$$

with a free variable *isbn* that will be instantiated with the answer. This is the call:

```
| ?- i_p(exists(id1):classified(isbn1,id1,s0) & exists(int1):(unclassified(isbn1,
     int1,s0) & int1>0),F),prolog_initial(F).
```

In the previous paragraph we showed how to use PROLOG as a query language, and some limitations of this approach. A more general alternative for similar tasks consists in using an automated theorem prover.

In order to prove statements about the specification provided by the user, SCDBR can call OTTER [13]. Specifically, it takes part in the verification of formulas at the initial database and it takes part in the proof of integrity constraints.

OTTER can be used to verify the truth of formulas of the form $\varphi(S_0)$, being S_0 the only state term in the formula, at the initial database. This approach has the advantage, in comparison with the approach shown below, of not imposing any syntactic constraints on the initial database, that now can be first-order. On the other side, it has the disadvantage of not being directly applicable when we want a list of tuples as an answer to a query. For example, we cannot ask OTTER to give us all the books for which there is at least one copy classified at the initial state, but we can ask OTTER to tell us if there is a book for which there is at least one copy classified at the initial state.

For example: let us assume that in our specification in section 2 we know that there is a positive quantity of unclassified books with *isbn* identifiers $'0{-}412{-}14930{-}3'$, but we do not know exactly how many. In order to obtain an specification with this feature, it suffices to replace in the original specification, the tuple $Unclassified('0{-}412{-}14930{-}3', 3, S_0)$ by the formula:

$$\exists quantity\ (unclassified('0{-}412{-}14930{-}3', quantity, S_0)\ \wedge\ quantity > 0).$$

In this new initial database, which is not relational, we want to ask a query similar to (5), but now we want to know if there is a book which has at least one classified copy and one unclassified copy, everything at the initial state. In SCDBR, this query has the following format:

```
| ?- i_p(exists(isbn1) : (exists(id1) : classified(isbn1,id1, s0) & exists(int1):
     (unclassified(isbn1,int1,s0) & int1>0)), F), otter_initial(F).
```

A sequence of actions $A^{(1)}(\bar{t}^{(1)}), \ldots, A^{(n)}(\bar{t}^{(n)})$ is *legal* if it is possible to execute: $A^{(1)}(\bar{t}^{(1)})$ in the initial state, $A^{(2)}(\bar{t}^{(2)})$ in the state $do(A^{(1)}(\bar{t}^{(1)}), S_0)$, etc. This condition means that, for each action $A^{(i)}$, the precondition for its execution, say $\Pi_{A^{(i)}}$, is satisfied in the state that results from the execution of $A^{(1)}, \ldots, A^{(i-1)}$, the preceding actions in the sequence. This can be checked against the initial database if we apply the regression operator [15]. In order to check the precondition for action $A^{(i)}$ in the state resulting from the $i-1$ preceding actions, we apply $i-1$ times the regression operator, so that we get rid of the *do* symbols in the precondition formula to be checked.

Given a sequence of actions, SCDBR is able to verify if this sequence is legal by means of the predicate `al`, which calls the predicates `reg` and `ap` (that computes the action precondition axioms), for building a list of formulas to be checked against the initial database. For example, in the next query, we ask the system to verify if the sequence of transactions $order('3{-}540{-}18199{-}7', 1)$, $classifyBook('0{-}412{-}14930{-}3', '4')$ is legal. For this purpose we call the predicate `al`, the pruners based on unique names for actions and objects (by means of the predicate `p_l`), and the predicate `prolog_initial` (for verifying if the list of formulas generates by `al` is true in the initial state):

```
| ?- al([order('3-540-18199-7',1), classify_book('0-412-14930-3', '4')],[F1,F2]),
     p_l([ F1, F2 ], prune_una, [ F3, F4]), p_l([ F3, F4], prune_uno, [ F5, F6]),
     prolog_initial(and(F5,F6)).
```

We can verify if a formula φ is true, after executing a sequence of *legal* actions $A_1, \ldots, A_n$ (more precisely: $\Sigma \models \varphi(do(A_n, \ldots, do(A_1, S_0) \ldots)))$ by applying the procedures presented in the previous paragraphs. In order to do this, we first apply the regression operator n times to φ, so that the resulting expression mentions only state S_0. Next, we apply the prune operators, generating a new expression that we have to check against the initial database. This can be done with PROLOG as a query language or the automated theorem prover, OTTER. For example, we want to verify if there is a stock of books with the *isbn* $'0{-}412{-}14930{-}3'$ after the following sequence of actions $[order('3{-}540{-}18199{-}7', 1), classifyBook('0{-}412{-}14930{-}3', '4')]$ is executed. This can be expressed with the following formula:

$$\exists n\ stock('0{-}412{-}14930{-}3', n, do(classifyBook('0{-}412{-}14930{-}3', '4'), do(order('3{-}540{-}18199{-}7', 1), S_0))) \tag{6}$$

In order to answer this query, all we need is to perform the following operations in our system (notice that, after applying the pruners based on unique names for actions and objects, we call the simplifier `csf` [4]):

```
| ?- i_p(exists(int):stock('0-412-14930-3',int, do(classify_book('0-412-14930-3',
     '4'),do(order('3-540-18199-7', 1), s0))), F),reg_n(F,2,F1),prune_una(F1,F2),
     prune_uno(F2,F3),csf(F3, F4),prolog_initial(F4).
```

Now, given an initial realtional database and a sequence of legal transactions, we will briefly describe how to pose and answer historical queries (HQ). The most basic kind of HQ we can think of is "*Given a formula $\varphi_i(s)$ and two states s_{L_i}, s_{U_i} along a legal transaction list T, does φ_i hold in all intermediate states between s_{L_i}, s_{U_i} (including s_{L_i} and excluding s_{U_i})?*". A query of this kind can be expressed by the formula $(\forall s) s_{L_i} \leq s < s_{U_i} \supset \varphi_i(s)$. The states s_{L_i}, s_{U_i} are called the *bounds* of the query. The queries we can express in SCDBR are basically existential quantifications over the bounds of conjunctions of those basic *universal queries*, plus restrictions on the bounds [18].

In order to answer HQs, of the form mentioned above, SCDBR needs to keep the truth value of every formula φ_i in every state along T, hence, in principle, this procedure could be inefficient. However, only a few actions should be able to change the truth value of the formulas φ_i from one state to another, because every formula φ_i mentions only a few fluents. Moreover, only a few fluents should be "tracked" to answer the queries. In [18] there is a characterization of those "relevant" actions and fluents, based on the information provided by the successor state axioms. Using these ideas SCDBR reduces the cost of determining the truth values of the formulas φ_i along T.

Once the truth values are determined, and the constraints on the bounds have been taken into account, SCDBR generates a new query that only mentions equalities and inequalities between pairs of integers. Answering this new query can be seen as a *constraint satisfaction problem* (CSP). SCDBR solves the CSP using constraint logic programming, CLP(FD) [5]. The CSP can be always decided, and there is a "Yes" answer to the original HQ if and only if the CSP is solvable.

For example, given the (legal) sequence of transactions

$$[classifyBook('0{-}412{-}14930{-}3', '13'), classifyBook('0{-}412{-}14930{-}3', '14'), deleteBook('10')],$$

[4]Predicate `csf` simplifies formulas using some rules corresponding to universally true formulas of predicate calculus. For example, we know that $\varphi \wedge \varphi \equiv \varphi$ is a logical truth, so that `csf`, when applied to a formula, changes every occurrence of $\varphi \wedge \varphi$ by φ.

we want to know if there is a state (along the current transaction list) such that there is a book with more than three copies in stock. This is represented by the following formula:

$$\exists s(S_0 \leq s \leq do(T, S_0) \ \wedge \ \exists(isbn, int).\ Stock(isbn, int, s) \ \wedge \ int > 3) \tag{7}$$

Posing this query is done by means of the predicates `set_curr_trans` and `set_curr_query`. After that, we can ask if is it true in our specification, by posing the query `| ?- process_curr_query`.

Our system is able to update the initial database. We have implemented this using the "roll–forward" module, as implemented in GOLOG [7]. The basic idea is to transform the successor state axioms(SSAs) to PROLOG clauses and using the `findall` built–in predicate of PROLOG to find all the fluents that are true after executing an action.

There are other implementations of progression. One of them is based on the following idea: Given an action, to have a description of the tuples that must be added and removed from the database when this action is executed. Currently, SCDBR parses the specification, trying to find out if it fits in the *context–free* specifications [8, 10]. If so, it automatically adds the update description, generating a more efficient algorithm for database update.

For example: we wish to update the initial database of the library specification as result of classify a new copy of the book with the *isbn* number $'0{-}412{-}14930{-}3'$. We can do this by posing the query `| ?- progress(classify_book('0-412-14930-3','13'))`. We obtain a new initial database that contains a set of tuples, some of which represent the following formula (notice that we use the same state symbol S_0 for representing this new initial state).

$$\begin{aligned} Classified(isbn, id, S_0) \equiv\ & (isbn =' 0{-}412{-}14930{-}3' \wedge id =' 10') \vee (isbn =' 0{-}412{-}14930{-}3' \wedge id =' 11') \vee \\ & (isbn =' 0{-}412{-}14930{-}3' \wedge id =' 13') \vee (isbn =' 0{-}7167{-}8162{-}X' \wedge id =' 12') \end{aligned}$$

Given an integrity constraint (IC), it is natural to wonder if it is provable form our specification, or, in other words, if it is true in all legal states, that is, in all states that are accessible from the initial one by means of a finite and legal sequence of actions. If it is not the case, then we might feel tempted to modify our specification so that the desired integrity is entailed by the new specification.

SCDBR is able, with some qualifications, to solve these tasks, namely: (1) to automatically prove ICs that are logical consequences of the specification , and (2) to automatically modify the preliminary specification in order to subsume the IC. In the next paragraphs we describe these two functionalities.

Given a specification Σ, and an IC $\varphi(s)$, with s being the only variable of sort *state* in it (*static integrity constraint*), we can prove that $\varphi(s)$ holds in all accessible states s by means of an inductive proof that, according to a lemma by Lin and Reiter [9], only has to verify the following conditions: (1) $\Sigma \models \varphi(S_0)$, that is, the IC hold at the initial state; (2) $\Sigma \models \forall(a, s).\varphi(s) \wedge Poss(a, s) \supset \varphi(do(a, s))$. This is an inductive step, where it is proved that the IC holds at every legal successor state $do(a, s)$ of every state s where the IC holds. In SCDBR, these two verifications are done with the predicate `prove_ic`. The proof in the base case, $\varphi(S_0)$, is done with PROLOG as a query language. The proof of the inductive step is done by calling OTTER. For example, the library specification entails the following functional dependency:

$$Classified(isbn_1, id, s) \ \wedge \ Classified(isbn_2, id, s) \ \supset \ isbn_1 = isbn_2, \tag{8}$$

which states that two different books cannot be classified with the same identifier id. We will use predicate `prove_ic` to automatically prove this IC.

```
| ?- i_p( forall(isbn1) : forall(isbn2) : forall(id1) :(classified(isbn1,id1,s) &
     classified(isbn2,id1,s) => isbn1 eq isbn2),F),prove_ic(F).
```

For proving ICs by induction, SCDBR may also call RRL [6], a theorem prover based on term rewriting and automated generation of induction schemas. The methodology is reported in [4].

A first attempt to subsume ICs into a preliminary specification, in such a way that the IC be entailed by the new specification, consists in modifying the effect axioms. To provide a general solution is an open problem. In [14], Pinto gives a satisfactory solution for the class of binary ICs [14], i.e. formulas of the form $[\neg]F_1(\vec{x_1}, s) \vee [\neg]F_2(\vec{x_2}, s) \vee \psi$, where F_1 and F_2 represent any two fluents in the specification; and ψ is a formula without state variables.

Our system is able to subsume binary ICs into the (new) specification by means of the predicate `ramification`, which is based on the following procedure proposed by Pinto [14]. In order to simplify the presentation, we will assume that the ICs have the form $F_1(\vec{x_1}, s) \vee F_2(\vec{x_2}, s) \vee \psi$. In this case, for each negative effect axiom for F_1, of the form $Poss(A(\vec{x}), s) \wedge \varepsilon^-_{F_1}(\vec{x}, \vec{y}, s) \supset \neg F_1(\vec{y}, do(A(\vec{x}), s))$, a new positive effect axiom for F_2 is added to the specification: $Poss(A(\vec{x}), s) \wedge \varepsilon^-_{F_1}(\vec{x}, \vec{y}, s) \wedge \neg\psi \supset F_2(\vec{y}, do(A(\vec{x}), s))$. Symmetrically, we obtain, for each negative effect axiom for F_2, a new positive effect axiom for F_1 (of the previous form), but with the formula $\varepsilon^-_{F_1}(\vec{x}, \vec{y}, s)$ replaced by $\varepsilon^-_{F_2}(\vec{x}, \vec{y}, s)$), which is added to the specification. If negations appear, the procedure works in a similar way.

For example, let us assume that the penultimate effect axiom of the library specification was deleted from (or not included in) it. With this, we have a semantically incorrect specification. Now it can be deduced that if at a given state we had *quantity* books with identifier *isbn* in stock, and an extra book was classified with this *isbn*, then we will have simultaneously $quantity + 1$ and *quantity* of those books in stock; what is not intended. More formally, the modified specification does not satisfy the following functional dependency[5]:

$$\neg Stock(isbn, quantity_1, s) \;\vee\; \neg Stock(isbn, quantity_2, s) \;\vee\; quantity_1 = quantity_2. \quad (9)$$

Applying to the procedure described above, SCDBR will generate a new and semantically correct specification that will (semantically) subsume the functional dependency. More precisely, this is done in the system in this way:

```
| ?- i_p(neg stock(isbn1, int1, s) v neg stock(isbn1, int2,s) v int1 eq int2,R1),
     ramification(R1,R2).
```

The new set of effect axioms were computed with the predicate `ramification` and they are contained in the variable `R2`. Basically, this new set differs from the previous one by the following axiom provided by the system:

$$\begin{aligned} &Poss(classifyBook(isbn, id), s) \;\wedge\; \neg quantity_1 = quantity_2 \;\wedge\; ((Stock(isbn, quantity_3, s) \;\wedge \\ &\quad quantity_1 = quantity_3 + 1) \;\vee\; (\forall quantity\; \neg Stock(isbn, quantity, s) \;\wedge\; quantity_1 = 1)) \;\supset \\ &\quad\quad \neg Stock(isbn, quantity_2, do(classifyBook(isbn, id), s)). \end{aligned}$$

Finally, if we add this new axiom to our specification, we will restore correctness.

A second alternative to (semantical) subsumption of ICs into a new specification consists in modifying action precondition axioms. In [9] there is simple general solution: given an action A with precondition axiom $\Pi_A(s)$, and an IC, $\varphi(s)$, if we want to be sure that $\varphi(s)$ will hold after executing A, it suffices to replace $\Pi_A(s)$ by $\Pi_A(s) \wedge \varphi(do(A, s))$. This could be the new precondition for A. Nevertheless, this axiom, with occurrences of both s and $do(A, s)$), is not acceptable. This problem can be easily solved with one application of the regression operator. The new precondition

[5]In usual database notation, $isbn \longrightarrow quantity$.

will be $\Pi_A(s) \wedge \mathcal{R}[\varphi(do(A,s))]$. This methodology can be applied for every named action A. In this way the IC will hold after executing any legal action (in the new sense, i.e. when the new preconditions for their executions have been considered).

For example, we can change the library specification by replacing the precondition axiom for action *classifyBook* by the formula:

$$Poss(classifyBook(isbn, id), s) \equiv (\exists copies) Unclassified(isbn, copies, s) \tag{10}$$

Now we have an incorrect specification, because there may exist two different books with the same id. That is, the specification does not entail the functional dependency (8). In order to compute a new specification from which the IC follows, we apply predicate `qualification` in this way:

```
| ?- i_p(forall(isbn1) : forall(isbn2) : forall(id1) : (classified(isbn1,id1,s) &
     classified(isbn2,id1,s) => isbn1 eq isbn2),F),qualification(F,R).
```

The new precondition axiom for $classifyBook$ is represented by the formula:

$$Poss(classifyBook(isbn, id), s) \equiv \exists copies\ Unclassified(isbn, copies, s)\ \wedge\ \varphi(s)\ \wedge \\ \forall isbn_2\ (classified(isbn_2, id, s)\ \supset\ isbn = isbn_2).$$

Here, $\varphi(s)$ is IC (8). Notice that this new formula basically adds to (10) the fact that $classifyBook$ can be executed if, among other conditions, every time we try to classify an exemplar of a book, with an id that was used before, then the same exemplar must be classified.

Acknowledgments: We are grateful to Bernardo Siu, Pablo Saez, Mauricio Strello, M. Alejandra Delaporte for stimulating discussions and all their work in SCDBR. We are also very grateful to Ray Reiter, Deepak Kapur, Javier Pinto, Tania Bedrax, Erik Sandewall, Jack Minker, Richard Scherl, Mahadevan Subramaniam, Aris Zakinthinos, Daniel Marcu, Fangzhen Lin and Alvaro Campos for interesting conversations, help, advice, information, joint work, etc. This research has been supported partly by DIUC, and FONDECYT (#193–0554) grants.

References

[1] L. Bertossi, J. Pinto, P. Saez, D. Kapur, and M. Subramaniam. Proving Integrity Constraints with a Rewrite Rule based Theorem Prover. Reporte Tecnico RT-PUC-DCC-95-1, Catholic University of Chile, School of Engineering, Department of Computer Science, Casilla 306, Santiago 22, Chile, 1995.

[2] Leopoldo Bertossi, Marcelo Arenas, and Cristian Ferretti. SCDBR: An Automated Reasoner for Specifications of Database Updates. *To be 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, Bonn, Germany, 1994. Springer.

[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.* Springer, LNAI 1079., 1996.

[5] D. Díaz. *clp(FD) 2.21 User's Manual.* INRIA–Rocquencourt, France, 1994.

[6] D. Kapur and H. Zhang. An Overview of Rewrite Rule Laboratory (RRL). *J. of Computer and Mathematics with Applications*, 1995.

[7] 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.

[8] F. Lin and R. Reiter. How to Progress a Database (and Why) I: Logical Foundations. In E. Sandewall J.Doyle and P. Torasso, editors, *Proc. Fourth Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 425–436. Morgan Kaufmann, 1994.

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

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

[11] J. W. Lloyd. *Foundations of Logic Programming.* Springer Verlag, 1987.

[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] W.W. McCune. *OTTER 3.0 Reference Manual and Guide.* Argonne National Laboratory, Technical Report ANL-94/6, 1994.

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

[15] 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.

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

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

[18] 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://ing.puc.cl/dcc/techReports, 1996.

[19] 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.