Situation Calculus Based Specifications of Action and Change

Leopoldo Bertossi
Carleton University
School of Computer Science

The Situation Calculus

A family of languages of many–sorted predicate logic

Widely used in logic based knowledge representation in AI

Introduced in the 60's by John McCarthy to represent knowledge and reason about dynamic domains

Domains subject to the execution of actions and discrete change

Regained popularity in the 90's due mainly to the work of Raymond Reiter:

- A simple solution to the *frame problem* in the SC

  Given a specification in terms of preconditions for and effects of actions

  How to obtain a compact, succinct specification of the many things that are not changed by action executions

- Extensions of SC and Reiter's formalism to: cognitive actions, explicit time, natural and physical phenomena, concurrent actions, etc.

The Situation Calculus Languages

Domain individual, states and actions at the same first–order object level

First–order quantifications on all these sorts of individuals are possible: $\forall \bar{x}$, $\forall a$, $\forall s$

Domain dependent elements:

- Action Names, e.g. $promote(x, p)$
- Fluents (evolving predicates, with one state argument)
  Think of database base tables, e.g. $Enrolled(x, p, s)$
- Names for domain individuals, e.g. $john$
- Other predicates, e.g. state independent

Some fixed ingredients:

- S_0 denotes the initial state
- A function name do:

  $do(a, s)$ denotes the successor state that results from executing action a at state s
- Predicate $Poss$:

  $Poss(a, s)$ says that action a is possible at state s

Foundational Axioms for the SC

Unique Names Axioms for Actions: $A_i(\bar{x}) \neq A_j(\bar{y})$, for all different action names A_i, A_j

Example: $delete(id) \neq classifyBook(isbn, id')$

Unique Names Axioms for States:

$$S_0 \neq do(a, s)$$

$$do(a_1, s_1) = do(a_2, s_2) \ \rightarrow \ a_1 = a_2 \wedge s_1 = s_2$$

For some reasoning tasks the
Induction Axiom on States (IA):

$$\forall P\ (P(S_0) \land \forall s \forall a\ (P(s) \rightarrow P(do(a,s))) \quad \rightarrow \quad \forall s\ P(s))$$

IA restricts the domain of situations to S_0 plus the situations obtained by executing actions

We are usually interested in reasoning about states that are *accessible* from S_0 by executing a finite sequence of legal actions

Accessibility relation on states, $\leq$, can be defined from the IA plus the conditions:

$$\neg s < S_0$$

$$s < do(a,s') \equiv Poss(a,s') \land s \leq s'$$

An Example: Specifying DB Updates

A Bibliographical Database:

Predicate:

- $BooksInPrint(isbn, title, author, editor, year, edition)$:

Fixed table containing all printed books (that may be ordered)

Tables (Fluents):

- $Unclassified(isbn, copies, s)$:

Contains books that have been ordered, but not classified yet

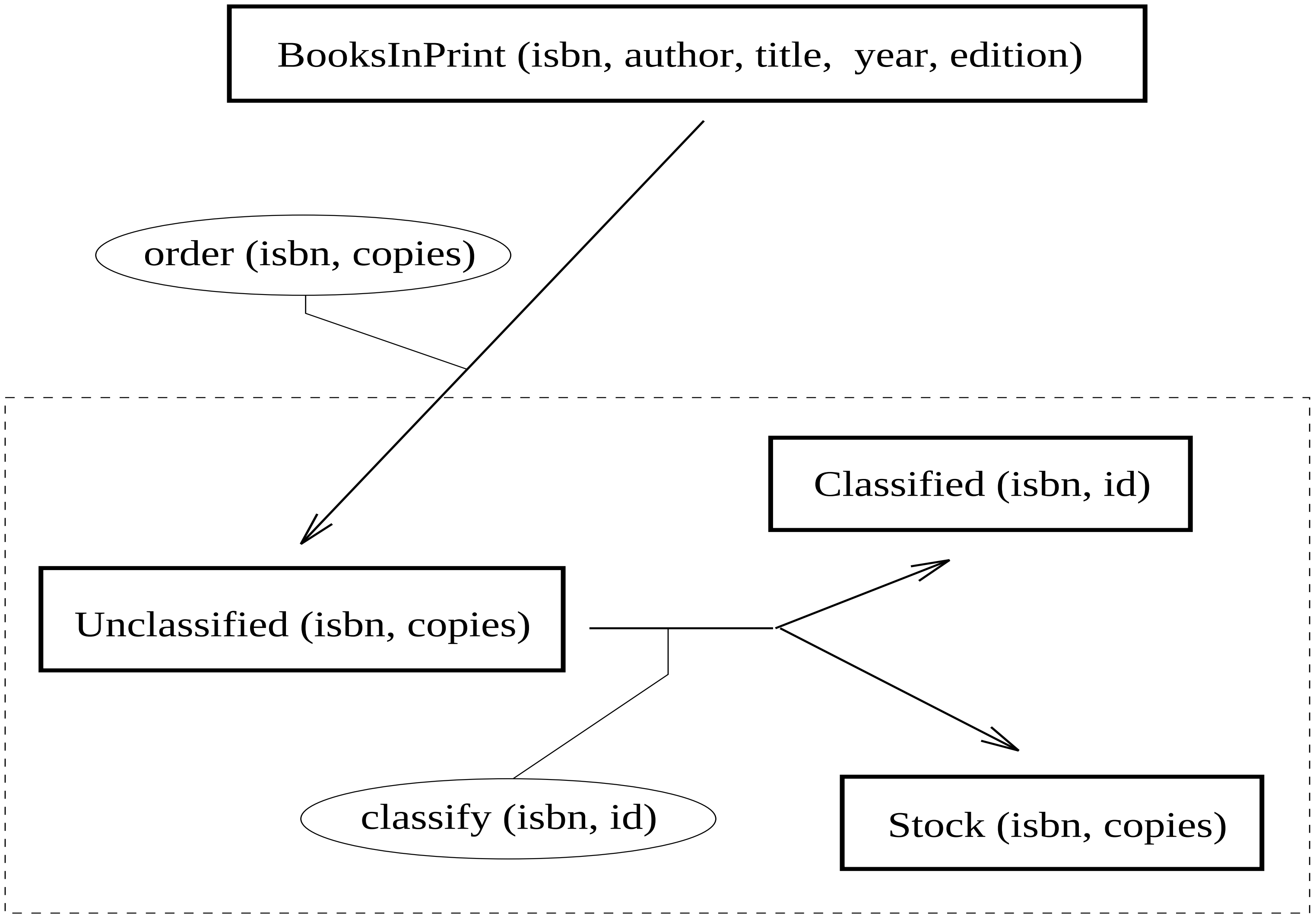
- $Classified(isbn, id, s)$:

Contains an entry for each copy of a book

Each copy has its own internal classification number id

- $Stock(isbn, copies, s)$:

Contains an entry for each classified book, indicating how many copies there exist


BooksInPrint (isbn, author, title, year, edition)
order (isbn, copies)
Classified (isbn, id)
Unclassified (isbn, copies)
classify (isbn, id)
Stock (isbn, copies)

Example Continued: Actions

Update Actions (atomic actions):

- $order(isbn, copies)$:

Orders an item from $BooksInPrint$ and adds it into $Unclassified$

- $classifyBook(isbn, id)$:

Deletes an $isbn$ from $Unclassified$, it assigns an id to it, and adds the tuple formed this way into $Classified$

It also updates $Stock$ incrementing by one the number of copies

- $deleteBook(id)$:

Deletes a book corresponding to id from $Classified$

It also decreases the number of copies in $Stock$

Action Precondition Axioms:

- $Poss(order(isbn, copies), s) \quad \equiv \quad \exists\, title, author, editor, year, edition$
  $BooksInPrint(isbn, title, author, editor, year, edition)$

- $Poss(classifyBook(isbn, id), s) \equiv$
  $\neg(\exists\ isbn')\ Classified(isbn', id, s)\ \wedge$
  $(\exists\ copies)\ Unclassified(isbn, copies, s)$

Example Continued: Effect Axioms

Effect Axioms for *Classified*:

- $Poss(deleteBook(id), s) \land Classified(isbn, id, s) \rightarrow$

  $$\uparrow$$

  $$\neg Classified(isbn, id, do(deleteBook(id), s))$$

  $$\uparrow$$

  - □ The precondition for the action (specified before)
  - □ The (pre)condition on the world at execution state (aka. fluent precondition)
  - □ The effect at the successor state

    In this case, a *negative* effect axiom

- $Poss(classifyBook(isbn, id), s) \rightarrow$

  $$Classified(isbn, id, do(classifyBook(isbn, id), s)$$

  A *positive* effect axiom

Effect Axioms for *Stock*:

- $Poss(classifyBook(isbn, id), s) \wedge$
  $((Stock(isbn, copies, s) \wedge copies' = copies + 1) \vee$
  $(\neg Stock(isbn, copies, s) \wedge copies' = 1)) \rightarrow$
  $Stock(isbn, copies', do(classifyBook(isbn, id), s))$

- $Poss(classifyBook(isbn, id), s) \wedge Stock(isbn, copies, s) \rightarrow$
  $\neg Stock(isbn, copies, do(classifyBook(isbn, id), s))$

A negative and a positive effect axiom for $Stock$

SCDBR: A Pre-Specification

User's pre-specification given to the system:

1. Similarity type of the FOL: names for fluents, predicates, actions, distinguished individuals, ...
2. For each action:

ACTION	$Action$
PRECONDITION	$ActionPrecondition$
EFFECTS	$StateCondition_1 : EffectPolarity_1\ Fluent_1$
	...
	$StateCondition_n : EffectPolarity_n\ Fluent_n$

   $Action$ affects either positively ($+$) or negatively (-) the $Fluent_i$ (at the successor state) if the $StateCondition_i$ (on the fluent) holds (at the current state)

   Like a possibly "sparse matrix" of Actions vs. Fluents, with entries $+$, $-$, or, most of them, empty
3. A set of formulas about the initial state (the initial data-base)

Successor State Axioms

So far: Specifications of things that change when actions are executed

Frame Problem: How to specify the (many!) things that do not change

Solution (R. Reiter 1991): Generate a new specification from the previous one, indicating exactly under which conditions each fluent becomes true at an arbitrary successor state (SS) $do(a, s)$

For each fluent $R(s)$, consider all its effect axioms (EAs), negative and positive

Apply metalevel assumption: EAs axioms provide all the conditions under which R becomes true or false, resp., at a legal SS

So, construct an axiom of the form

$$\forall a \forall s\ Poss(a,s) \rightarrow [R(do(a,s)) \equiv \gamma_R^+(a,s) \vee (R(s) \wedge \neg\gamma_R^-(a,s))]$$

$\gamma_R^+(a,s)$: actions and their conditions, retrieved from EAs, that make R true

$\gamma_R^-(a,s)$: actions and their conditions, retrieved from EAs, that make R false

That is: R is true at a successor state iff it is made true or it was already true and it is not made false

SSAs can be automatically computed from the effect axioms!

This solution relies on the possibility of quantifying over actions

This solution works for deterministic actions only

With this transformation, the commonsense (nonmonotonic) metalevel assumption was materialized into a full first–order formalism

The formula on the RHS of $[...]$ does not contain $do(a, s)$, but only s

Example Continued: A SSA

$$
\begin{array}{l}
\forall a \forall s\ Poss(a,s) \rightarrow \\
\quad (Stock(isbn, j, do(a,s)) \equiv \\
\qquad a = delete\_book(id) \wedge \\
\qquad\qquad (Classified(isbn, id, s) \wedge \\
\qquad\qquad\qquad \exists i\ (Stock(isbn, i, s) \wedge i > 1 \wedge j = i - 1)) \vee \\
\qquad a = classify\_book(isbn, id) \wedge \\
\qquad\qquad (\exists i\ (Stock(isbn, i, s) \wedge j = i + 1) \vee \\
\qquad\qquad\qquad \neg \exists i\ (Stock(isbn, i, s)) \wedge j = 1) \vee \\
\qquad Stock(isbn, j, s) \quad \wedge \neg(\\
\qquad\qquad\qquad a = delete\_book(id) \wedge \\
\qquad\qquad\qquad\qquad Classified(isbn, id, s) \wedge \\
\qquad\qquad\qquad\qquad\qquad Stock(isbn, j, s) \vee \\
\qquad\qquad\qquad a = classify\_book(isbn, id) \wedge \\
\qquad\qquad\qquad\qquad Stock(isbn, j, s)))
\end{array}
$$

SCDBR: More SSA's

SCDBR computes the SSA for *Stock*:

```
| ?- ssa( stock, A ),p_i( A, R ).

R = (forall(a):
     (poss(a,s) =>
      (stock(isbn1,i,do(a,s)) <=>
       (exists(id) : (a eq delete_book(id) &
         classified(isbn,id,s) &
         exists(j) : (stock(isbn,j,s) &
                      j>1 & i eq j-1)) v
        exists(id) : (a eq classify_book(isbn,id) &
         (exists(k) : (stock(isbn,k,s) &
                       i eq k+1) v
          forall(k) : (neg stock(isbn,k,s)) &
                       i eq 1))) v
      stock(isbn,i,s) &
       neg (exists(id) : (a eq delete_book(id) &
             classified(isbn,id,s) &
             stock(isbn,i,s)) v
            exists(id) : (a eq classify_book(isbn,id) &
             stock(isbn,i,s)))))) ?
```

An Example (Cont'd): Obtaining the SSA's

The result provided by the system when asked for the successor state axiom for the fluent $Classified$ is

```
| ?- ssa( classified, A ),p_i( A, R ).

R = (forall(a) : (poss(a,s) =>
                   (classified(isbn1,id1,do(a,s)) <=>
                      a eq classify_book(isbn1,id1) v
                        classified(isbn1,id1,s) &
                          neg a eq delete_book(id1)))) ?
```

This PROLOG term represents the desired SSA

$$\begin{aligned}\forall a \forall s\ (Poss(a,s) \rightarrow \quad & [Classified(isbn, id, do(a,s)) \ \equiv \\ & a = classifyBook(isbn, id) \\ & \vee \\ & Classified(isbn, id, s)\ \wedge\ \neg a = deleteBook(id)])\end{aligned}$$

SCDBR: The Specification Language

Formulas of specification language $\mathcal{L}$ are:

- internally represented by Prolog ground terms
- written in prefix notation
- processed by Prolog procedures

In consequence, the system uses two kinds of variables:

- usual Prolog variables (starting with upper-case letters)
- variables of the language $\mathcal{L}$, starting with lower–case letters, (treated as constants by Prolog)

The system contains procedures for translating formulas in prefix notation into infix notation:

```
p_i(·,·)
```

and in the other direction with

```
i_p(·,·)
```

Generating the SSA's

We know the general syntactic form of the successor state axioms:

$$Poss(a, s) \supset [R(do(a, s)) \equiv \gamma_R^+(a, s) \vee (R(s) \wedge \neg\gamma_R^-(a, s))].$$

This structure is represented by the PROLOG procedure:

```
 ssa(R, all(a,
         implies( poss(a, STATE_VAR ),
          iff(RSS, or(PGR,
                    and(RS, not(NGR))))))):- ...
```

`R`, `RSS`, `PGR`, `RS`, `NGR` represent the fluent, the fluent at the successor state, the formula γ_R^+, the fluent at the current state and the formula γ_R^-, respectively.

By means of unification, the PROLOG call `ssa(`$fluent$`,A)` will leave in `A` a formula instantiated with concrete values for `RSS`, `PGR`, `RS`, `NGR`, which are obtained by executing the procedures in the body of the clause.

The Regression Operator

- Introduced by Ray Reiter (1991)
- Crucial for most of the reasoning tasks
- E.g. it can be used for:
  - Testing legality of transactions: $do([T_1, \ldots, T_n], S_0)$
  - Queries to a virtually updated DB: $\varphi(do([T_1, \ldots, T_n], S_0)$?
  - Basis for planning
  - ...
- After successive applications, an arbitrary $\mathcal{L}$-formula can be "regressed" to a logically equivalent formula that does not mention the $do(a, s)$ operation, but only state variables and state S_0.

The regression operator uses the **successor state axioms** in an essential way:

$$\forall a \forall s \; Poss(a,s) \supset (F(do(a,s)) \equiv \Phi_F(a,s))$$

Remember that formula $\Phi_F(a,s)$ does not mention states of the form $do(a,s)$

Each application of the operator replaces, for each fluent F, the occurrences of $R(do(a,s))$ by $\Phi_F(a,s)$

An Example: Applying the Regression Operator

We can invoke the regression operator on the formula

$$LostBook('10', do(deleteBook('11'), S_0))$$

where we find the fluent $LostBook$ in a successor state

```
| ?- reg(lost_book('10',
                do(delete_book('11'),s0)),F),p_i(F,R).

R = lost_book('10',s0) &
            neg delete_book('10') eq delete_book('11')
```

The result is stored in variable R and represents the following formula:

$$LostBook('10', S_0) \ \wedge \ \neg(deleteBook('10') = deleteBook('11'))$$

Checking Legality of Transactions

Is the following transaction legal at the initial state S_0:

$$[order(isbn, copies), classifyBook(isbn, id)]?$$

Check possibility of $order$ at S_0 and the possibility of $classifyBook$ at the state that results from executing $order$ at S_0:

$$Poss(order(isbn, copies), s_0) \wedge \\ \qquad Poss(classifyBook(isbn, id), do(order(isbn, copies, s_0)))?$$

We may consider the equivalent formula

$$(\exists\ title, \ldots)\ BooksInPrint(isbn, title, \ldots) \wedge \\ (\exists\ copies)\ Unclassified(isbn, copies, do(order(isbn, copies), s_0)) \wedge \\ \neg(\exists\ isbn')\ Classified(isbn', id, do(order(isbn, copies, s_0))) \qquad \texttt{(*)}$$

Applying regression to $(*)$, we may transform it into an equivalent formula that only mentions the initial state S_0

The transaction is legal if and only if the regressed formula follows from the initial database (plus unique names axioms)

SCDBR applies regression to generate the formula to be checked against the IDB

When formulas are regressed to the initial database, comparisons between actions and comparisons between individuals in the domain are generated

SCDBR has three *pruning operators* that simplify formulas on the basis of the unique name axioms for actions, states and objects, obtaining logically equivalent formulas:

$$\mathcal{P}_{una}, \ \mathcal{P}_{uns}, \ \ \mathcal{P}_{uno}$$

Example: Checking Legality of Transactions

Is the following transaction legal at S_0

$$[\mathit{deleteBook}(13), \mathit{deleteBook}(12)]\ ?$$

With SCDBR:

```
 | ?-
al([delete_book('13'),delete_book('12')],F),
        p_l(F,prune_una,F1),p_l(F1,prune_uno,F2),
                      p_i(F,R),p_i(F1,R1),p_i(F2,R2).

R  = [lost_book('13',s0), lost_book('12',s0) &
         neg delete_book('13') eq delete_book('12')],
R1 = [lost_book('13',s0), lost_book('12',s0) &
                                  neg '13' eq '12'],
R2 = [lost_book('13',s0), lost_book('12',s0)] ?
```

This means that the transaction is legal if

$$D_{S_0}\ \models\ \mathit{LostBook}(13, S_0)\ \wedge\ \mathit{LostBook}(12, S_0)$$

Planning and Regression

The planning task is about constructively proving that

$$Spec \models \exists s(S_0 \leq s \wedge G(s)),$$

where $G(s)$ is a goal formula with only s as a free and situational variable

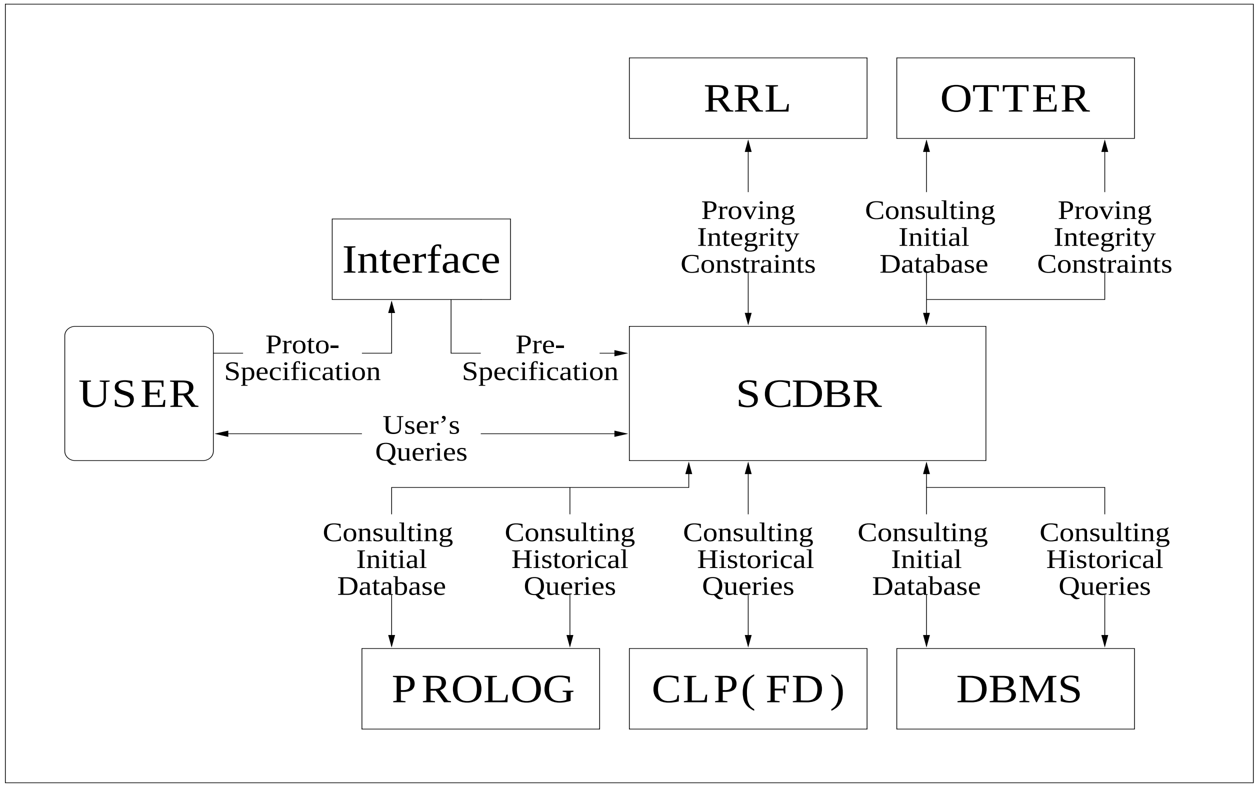
A plan can be syntactically synthesized by applying the regression operator to $G(s)$ producing a formula to be evaluated against the initial situation

If there is a such a situation s, it will be constructed in the regression process as a history of action executions (each of them possible at the execution situation)

And a plan is a history of action executions!

Exercise: Illustrate this kind of planning with a simple example

An Overview of SCDBR


RRL
OTTER
Proving Integrity Constraints
Consulting Initial Database
Proving Integrity Constraints
Interface
USER
Proto-Specification
Pre-Specification
SCDBR
User's Queries
Consulting Initial Database
Consulting Historical Queries
Consulting Historical Queries
Consulting Initial Database
Consulting Historical Queries
PROLOG
CLP(FD)
DBMS

Architecture of SCDBR

Example: Simplify the formula we obtained before by application of the regression operator.

```
| ?- 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),p_i(A,A1),p_i(B,B1).

A1 = (lost_book('0-7167-8162-X',s0) &
      neg '0-412-14930-3' eq '0-7167-8162-X'),
B1 = lost_book('0-7167-8162-X',s0)
```

Modifying the Specification: State Constraints

How to modify the specification in order to entail desired *state constraints*
Sentences that have to be true in all (admissible or legal) states of the world

They are of the form: $\forall s(\varphi(s))$

Example: $\forall s \forall x \forall y \neg(Classified(x,y,s) \wedge Unclassified(x,s))$

It should hold: Spec. in terms of SSAs $\models \forall s \varphi(s)$

BUT, if we missed something when writing down the axiom precondition and/or effect axioms, this may not be true

What to do? Instead of starting from scratch, figuring out what went wrong or missing?

Impose the state constraints on the existing specification and recompile the effect axioms (ramifications) or precondition axioms (qualifications)

Modifying the Specification: Ramifications

First attempt: modify the effect axioms (or SSAs)

A form of solution to the ramification problem: include the side effects of actions into the specification

General solution is an open problem

Pinto gives a solution for binary ICs: $\forall s \geq S_0\ \varphi(s)$, with $\varphi(s)$ of the form:

$$[\neg]F_1(\vec{x_1}, s) \vee [\neg]F_2(\vec{x_2}, s) \vee \psi$$

Functional dependencies are a special case

Example: Delete the following effect axiom from the library specification

$$Poss(classifyBook(isbn, id), s) \wedge Stock(isbn, quantity, s) \supset \\ \neg Stock(isbn, quantity, do(classifyBook(isbn, id), s))$$

A new, but semantically incorrect, specification is obtained: the modified specification does not satisfy the functional dependency (binary IC):

$$\neg Stock(isbn, quantity_1, s) \vee \neg Stock(isbn, quantity_2, s) \\ \vee quantity_1 = quantity_2$$

The SCDBR procedure `ramification` generates a new specification that entails the functional dependency

```
| ?- i_p(neg stock(isbn1,int1,s) v
             neg stock(isbn1,int2,s) v int1 eq int2, R1),
        ramification(R1,R2),p_i(R2,R3).
```

`R2` contains the new effect axioms. It recovers the missing effect axiom

Modifying the Specification: Qualifications

Lin and Reiter: alternative and general methodology for embedding ICs into a specification that entails them:

Modifying the action precondition axioms

Further constraints on the actions that qualify to be executed

Given an action A, with precondition axiom $\Pi_A(s)$, and an IC, $\varphi(s)$, to be sure that $\varphi(s)$ will hold after A, replace $\Pi_A(s)$ by $\Pi_A(s) \wedge \varphi(do(A, s))$

To avoid the occurrences of both s and $do(A, s)$, apply the regression operator

New precondition becomes $\Pi_A(s) \wedge \mathcal{R}[\varphi(do(A, s))]$

This is done for each named action A

The IC will hold after executing any legal action (now, satisf-
ying the new preconditions)

Example: Replace the precondition axiom for $classifyBook$ by:

$$Poss(classifyBook(isbn, id), s) \equiv$$

$$(\exists copies)Unclassified(isbn, copies, s)$$

Now specification is incorrect: there may appear two different books with the same id

Specification does not entail the FD:

$$Classified(isbn_1, id, s) \ \wedge \ Classified(isbn_2, id, s)$$

$$\supset \ isbn_1 = isbn_2$$

The new specification can be computed:

```
| ?- 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 set of axioms is stored in `R`

Predicate `qualification` automatically simplifies the formula resulting from the regression according to the unique names axioms for actions

We obtain:

$$Poss(classifyBook(isbn, id), s) \equiv$$

$$\exists copies\ Unclassified(isbn, copies, s)\ \wedge$$

$$\forall isbn_2\ (classified(isbn_2, id, s)\ \supset\ isbn = isbn_2)$$

In essence, the new axiom adds to the fact that $classifyBook$ can be executed if, among other conditions, every time we try to classify a copy of a book, with an id that was used before, then we must be classifying the same copy