FOUNDATIONS OF THE SITUATION CALCULUS

Motivation

- An analogy: The Peano axioms for number theory.
- The second order language (with equality):
  - A single constant 0.
  - A unary function symbol σ (successor function).
  - A binary predicate symbol $<$.
- A fragment of Peano arithmetic:

$$\sigma(x) = \sigma(y) \supset x = y,$$

$$(\forall P).\{P(0) \wedge [(\forall x).P(x) \supset P(\sigma(x))]\} \supset (\forall x)P(x)$$

$$\neg x < 0,$$

$$x < \sigma(y) \equiv x \leq y.$$

  Here, $x \leq y$ is an abbreviation for $x < y \vee x = y$.

- The second sentence is a second order induction axiom. It is a second order way of characterizing the domain of discourse as the *smallest* set such that
  1. 0 is in the set.
  2. Whenever x is in the set, so is $\sigma(x)$.
- Second order Peano arithmetic is *categorical* (it has a unique model).

- First order Peano arithmetic: Replace the second order axiom by an induction *schema* representing countably infinitely many first order sentences, one for each instance of P obtained by replacing P by a first order formula with one free variable.

- First order Peano arithmetic is *not* categorical; it has (infinitely many) *nonstandard* models. This follows from the Gödel incompleteness theorem, which says that first order arithmetic is *incomplete*, i.e. there are sentences true of the principal interpretation of the first order axioms (namely, the natural numbers) which are false in some of the nonstandard models, and hence not provable from the first order axioms.

- So why not use the second order axioms? Because second order logic is incomplete, i.e. there is no "decent" axiomatization of second order logic which will yield all the valid second order sentences!

- So why appeal to second order logic at all? Because *semantically*, but not syntactically, it characterizes the natural numbers. We'll find the same phenomenon in semantically characterizing the situation calculus.

Foundational Axioms for the Situation Calculus

- We use a 3-sorted language: The sorts are *situation*, *object* and *action*. There is a unique situation constant symbol, S_0, denoting the initial situation. It is like the number 0 in Peano arithmetic. Unlike Peano arithmetic which has a unique successor function, we have a family of successor functions.

  $do : action \times situation \rightarrow situation.$

- The axioms:

$$
\begin{aligned}
& do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2 && (1)\\
& (\forall P).P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] &&\\
& \quad \supset (\forall s)P(s) && (2)
\end{aligned}
$$

$$
\begin{aligned}
& \neg s \sqsubset S_0, && (3)\\
& s \sqsubset do(a, s') \equiv s \sqsubseteq s', && (4)
\end{aligned}
$$

  where $s \sqsubseteq s'$ is an abbreviation for $s \sqsubset s' \vee s = s'$.

- Compare with the Peano axioms for the natural numbers.

- Axiom (2) is a second order way of limiting the sort $situation$ to the smallest set containing S_0, and closed under the application of the function do to an action and a situation. Any model of these axioms will have its domain of situations isomophic to the smallest set $\mathcal{S}$ satisfying:

  1. $S_0 \in \mathcal{S}$.

2. If $S \in \mathcal{S}$, and $A \in \mathcal{A}$, then $do(A, S) \in \mathcal{S}$, where $\mathcal{A}$ is the domain of actions in the model.

- These axioms say that the tree of situations is really a tree. No cycles, no merging. It does *not* say that all models of these axioms have isomorphic trees (because they may have different domains of actions).

- Situations are finite sequences of actions. cf. LISP:
  - S_0 is just like NIL.
  - do acts like $cons$. $do(C, do(B, do(A, S_0)))$ is simply an alternative syntax for the LISP list $(C\ B\ A) = cons(C, cons(B, cons(A, NIL)))$.
  - To obtain the action *history* corresponding to this term, namely the performance of action A, followed by B, followed by C, read this list from right to left.
  - Therefore, when situation terms are read from right to left, the relation $s \sqsubset s'$ means that situation s is a proper subhistory of the situation s'.
  - The situation calculus induction axiom is simply the induction principle for lists: If the empty list has property P and if, whenever list s has property P so does $cons(a, s)$, then all lists have property P.

- These 4 axioms are *domain independent*. They will provide the basic properties of situations in any domain specific axiomatization of particular fluents and actions.

- Henceforth, call them Σ.

Some Consequences of these Axioms

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

$s = S_0 \vee (\exists a, s')s = do(a, s').$ (Existence of a predecessor)

$S_0 \sqsubseteq s.$

$s_1 \sqsubset s_2 \supset s_1 \neq s_2.$ (Unique names)

$\neg s \sqsubset s.$ (Anti-reflexivity)

$s \sqsubset s' \supset \neg s' \sqsubset s.$ (Anti-symmetry)

$s_1 \sqsubset s_2 \wedge s_2 \sqsubset s_3 \supset s_1 \sqsubset s_3.$ (Transitivity)

$s \sqsubseteq s' \wedge s' \sqsubseteq s \supset s = s'.$

The Principle of Double Induction

$$
\begin{aligned}
&(\forall R).R(S_0, S_0) \;\wedge \\
&\qquad [(\forall a, s).R(s, s) \supset R(do(a, s), do(a, s))] \;\wedge \\
&\qquad [(\forall a, s, s').s \sqsubseteq s' \wedge R(s, s') \supset R(s, do(a, s'))] \\
&\qquad\qquad \supset (\forall s, s').s \sqsubseteq s' \supset R(s, s').
\end{aligned}
$$

Executable Situations

- A situation is a finite sequence of actions. There are no constraints on the actions entering into such a sequence, so that it may not be possible to actually execute these actions one after the other.
- *Executable* situations: Action histories in which it is actually possible to perform the actions one after the other.

  $$s < s' \stackrel{def}{=} s \sqsubset s' \wedge (\forall a, s^*).s \sqsubset do(a, s^*) \sqsubseteq s' \supset Poss(a, s^*).$$

  $s < s'$ means that s is an initial subhistory of s', and all the actions occurring between s and s' can be executed one after the other.

  $$s \leq s' \stackrel{def}{=} s < s' \vee s = s',$$

  $$executable(s) \stackrel{def}{=} S_0 \leq s.$$

More Consequences of the Axioms

$$executable(do(a,s)) \equiv executable(s) \wedge Poss(a,s),$$

$$executable(s) \equiv$$
$$s = S_0 \vee (\exists a, s').s = do(a,s') \wedge Poss(a,s') \wedge executable(s'),$$

$$executable(s') \wedge s \sqsubseteq s' \supset executable(s).$$

The Principle of Induction for Executable Situations

$$(\forall P).P(S_0) \wedge (\forall a,s)[P(s) \wedge executable(s) \wedge Poss(a,s) \supset P(do(a,s))] \supset (\forall s).executable(s) \supset P(s).$$

The Principle of Double Induction for Executable Situations

$$\begin{aligned}&(\forall R).R(S_0,S_0) \wedge \\ &[(\forall a,s).Poss(a,s) \wedge executable(s) \wedge R(s,s) \supset R(do(a,s),do(a,s))] \wedge \\ &[(\forall a,s,s').Poss(a,s') \wedge executable(s') \wedge s \sqsubseteq s' \wedge R(s,s') \supset R(s,do(a,s'))] \\ &\qquad \supset (\forall s,s').executable(s') \wedge s \sqsubseteq s' \supset R(s,s').\end{aligned}$$

REASONING ABOUT SITUATIONS IN THE SITUATION CALCULUS

Why Prove Properties of World Situations?

- **Reasoning about systems.**

  $(\forall s).light(s) \equiv [open(Sw_1, s) \equiv open(Sw_2, s)]$.

  This has the typical syntactic form for a proof by the simple induction axiom of the foundational axioms.

- **Planning.**

  - The standard logical account of planning views this as a theorem proving task.
  - To obtain a plan whose execution will lead to a world situation s in which the goal $G(s)$ will be true, establish that

    $$Axioms \models (\exists s).executable(s) \wedge G(s).$$

  - Sometimes we would like to establish that no plan could possibly lead to a given world situation. This is the problem of establishing that

    $$Axioms \models (\forall s).executable(s) \supset \neg G(s),$$

    i.e. that in all possible future world situations, G will be false.

Why Prove Properties (Continued)

- **Integrity constraints in database theory**
  Some background:
  - An *integrity constraint* specifies what counts as a legal database state. A property that every database state must satisfy.
    Examples:
    * Salaries are functional: No one may have two different salaries in the same database state.
    * No one's salary may decrease during the evolution of the database.
  - The concept of an integrity constraint is intimately connected with that of database *evolution*.
    No matter how the database evolves, the constraint will be true in all database futures.
    $\Longrightarrow$
    In order to make formal sense of integrity constraints, need a prior theory of database evolution.
  - How do databases change?
    One way is via predefined update *transactions*, e.g.
    * Change a person's salary to \$.
    * Register a student in a course.
  - Transactions provide the only mechanism for such state changes.

– We have a situation calculus based theory of database evolution, so use it!

– Represent integrity constraints as first order sentences, universally quantified over situations.

  * No one may have two different grades for the same course in any database state:

$$(\forall s)(\forall st, c, g, g').S_0 \leq s \wedge grade(st, c, g, s) \wedge grade(st, c, g', s) \\ \supset g = g'.$$

  * Salaries must never decrease:

$$(\forall s, s')(\forall p, \$, \$').S_0 \leq s \wedge s \leq s' \wedge sal(p, \$, s) \wedge sal(p, \$', s') \\ \supset \$ \leq \$'.$$

– **Constraint satisfaction defined:** A database *satisfies* an integrity constraint IC iff

$$Database \models IC.$$

Summary

- Both dynamically changing worlds, and databases evolving under update transactions may be represented in the situation calculus.
- In general, we assume given some situation calculus axiomatization, with a distinguished *initial situation* S_0.
- Objective is to prove properties true of all situations in the future of S_0.
  Examples:
  $$(\forall s).light(s) \equiv [open(Sw_1, s) \equiv open(Sw_2, s)].$$
  $$(\forall s, s', p, \$, \$').executable(s') \wedge s \sqsubseteq s' \wedge sal(p, \$, s) \wedge sal(p, \$', s') \supset \$ \leq \$'.$$
- These are sentences universally quantified over situations. Normally, such sentences requires induction!

Proving Properties of Situations: An Example

$$(\forall s).light(s) \equiv [open(Sw_1, s) \equiv open(Sw_2, s)].$$

- Assume this is true of the initial situation:

$$light(S_0) \equiv [open(Sw_1, S_0) \equiv open(Sw_2, S_0)].$$

- Successor state axioms for $open, light$:

$$open(sw, do(a, s)) \equiv \neg open(sw, s) \wedge a = toggle(sw) \ \vee \\ open(sw, s) \wedge a \neq toggle(sw).$$

$$light(do(a, s)) \equiv \\ \neg light(s) \wedge [a = toggle(Sw_1) \vee a = toggle(Sw_2)] \vee \\ light(s) \wedge a \neq toggle(Sw_1) \wedge a \neq toggle(Sw_2).$$

- Simple induction principle:

$$P(S_0) \wedge [(\forall a, s).P(s) \supset P(do(a, s))] \supset (\forall s).P(s).$$

- So, take $P(s)$ to be:

$$light(s) \equiv [open(Sw_1, s) \equiv open(Sw_2, s)].$$

- QED

Proving Properties of Situations: Another Example

Salaries must never decrease:

$$(\forall s, s', p, \$, \$').executable(s') \wedge s \sqsubseteq s' \wedge sal(p, \$, s) \wedge sal(p, \$', s') \supset \$ \leq \$'.$$

- To change a person's salary, the new salary must be greater than the old:

$$Poss(\mathit{change\text{-}sal}(p, \$), s) \equiv (\exists \$').sal(p, \$', s) \wedge \$' < \$.$$

- Successor state axiom for sal:

$$sal(p, \$, do(a, s)) \equiv a = changeSal(p, \$) \ \vee \\ sal(p, \$, s) \wedge (\forall \$')a \neq changeSal(p, \$').$$

- Initially, the relation sal is functional in its second argument:

$$sal(p, \$, S_0) \wedge sal(p, \$', S_0) \supset \$ = \$'.$$

- *Unique names axiom* for *change-sal*:

$$\mathit{change\text{-}sal}(p, \$) = \mathit{change\text{-}sal}(p', \$') \supset p = p' \wedge \$ = \$'.$$

- Double induction principle:

$$(\forall R).R(S_0, S_0) \wedge \\ [(\forall a, s).Poss(a, s) \wedge executable(s) \wedge R(s, s) \supset R(do(a, s), do(a, s))] \wedge \\ [(\forall a, s, s').Poss(a, s') \wedge executable(s') \wedge s \sqsubseteq s' \wedge R(s, s') \supset R(s, do(a, s'))] \\ \supset (\forall s, s').executable(s') \wedge s \sqsubseteq s' \supset R(s, s').$$

- The sentence to be proved is logically equivalent to:

$$(\forall s, s').executable(s') \wedge s \sqsubseteq s' \supset$$

$$(\forall p, \$, \$').sal(p, \$, s) \wedge sal(p, \$', s') \supset \$ \leq \$'.$$

So, take $R(s, s')$ to be:

$$(\forall p, \$, \$').sal(p, \$, s) \wedge sal(p, \$', s') \supset \$ \leq \$'.$$

- QED

BASIC THEORIES OF ACTIONS

Combining our Axioms

- Recall that Σ denotes the four foundational axioms for situations.
- We now consider some metamathematical properties of these axioms when combined with successor state and action precondition axioms, and unique names axioms for actions. Such a collection of axioms will be called a *basic theory of actions*.
- First we must be more precise about what counts as successor state and action precondition axioms.

The Uniform Formulas

- Let σ be a term of sort $situation$. A formula is $uniform\ in$ σ iff it does not mention the predicates $Poss$ or $\sqsubset$, it does not quantify over variables of sort $situation$, it does not mention equality on situations, and whenever it mentions a term of sort $situation$ in the situation argument position of a fluent, then that term is σ.

AP and SS Axioms

- **Definition: Action Precondition Axiom**
  An action precondition axiom is a sentence of the form:

  $$(\forall x_1, \cdots, x_n, s).Poss(A(x_1, \cdots, x_n), s) \equiv \Pi_A(x_1, \cdots, x_n, s),$$

  where A is an n-ary action function, and Π_A is a formula that is uniform in s and whose free variables are among $x_1, \cdots, x_n, s$.

- **Definition: Successor State Axiom** A successor state axiom for an $(n+1)$-ary fluent F is a sentence of the form:

  $$(\forall a, s)(\forall x_1, \ldots, x_n).F(x_1, \ldots, x_n, do(a, s)) \equiv \Phi_F, \qquad (5)$$

  where Φ_F is a formula uniform in s, all of whose free variables are among $a, s, x_1, \ldots, x_n$.

- Formulas uniform in s = Markov property. The future is determined by the present.

- We do not assume that successor state axioms have the exact syntactic form as those obtained earlier by combining Schubert and Pednault's ideas. The discussion there was meant to motivate one way that successor state axioms of the form (5) might arise, but nothing in the development that follows depends on that earlier approach.

Basic Action Theories

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$$

where

- Σ are the foundational axioms for situations.
- $\mathcal{D}_{ss}$ is a set of successor state axioms.
- $\mathcal{D}_{ap}$ is a set of action precondition axioms.
- $\mathcal{D}_{una}$ is the set of unique names axioms for actions.
- $\mathcal{D}_{S_0}$ is a set of first order sentences with the property that S_0 is the only term of sort $situation$ mentioned by the fluents of a sentence of $\mathcal{D}_{S_0}$. Thus, no fluent of a formula of $\mathcal{D}_{S_0}$ mentions a variable of sort $situation$ or the function symbol do. $\mathcal{D}_{S_0}$ will play the role of the initial situation of the world (i.e. the one we start off with, before any actions have been "executed").

Theorem 1 (Relative Satisfiability)
$\mathcal{D}$ is satisfiable iff $\mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ is.

Regression

- The **Regressable Formulas**.

  The essence of a regressable formula is that each of its *situation* terms is rooted at S_0, and therefore, one can tell, by inspection of such a term, exactly how many actions it involves. It is not necessary to be able to tell what those actions are, just how many they are. In addition, when a regressable formula mentions a $Poss$ atom, we can tell, by inspection of that atom, exactly what is the action function symbol occurring in its first argument position, for example, that it is a $move$ action.

- Assume a background axiomatization that includes a set of successor state and action precondition axioms.

- **The Regression Operator: Simple Version**
  W is a regressable formula of $\mathcal{L}_{sitcalc}$ that mentions no functional fluents.

  1. Suppose W is an atom. Since W is regressable, there are four possibilities:

     (a) W is a situation independent atom. Then

     $$\mathcal{R}[W] = W.$$

     (b) W is a relational fluent atom of the form $F(\vec{t}, S_0)$. Then

     $$\mathcal{R}[W] = W.$$

     (c) W is a regressable $Poss$ atom, so it has the form $Poss(A(\vec{t}), \sigma)$ for terms $A(\vec{t})$ and σ of sort $action$ and $situation$ respectively. Here, A is an action function symbol of $\mathcal{L}_{sitcalc}$. Then there must be an action precondition axiom for A of the form

$$Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s).$$

Assume that all quantifiers (if any) of $\Pi_A(\vec{x}, s)$ have had their quantified variables renamed to be distinct from the free variables (if any) of $Poss(A(\vec{t}), \sigma)$. Then

$$\mathcal{R}[W] = \mathcal{R}[\Pi_A(\vec{t}, \sigma)].$$

In other words, replace the atom $Poss(A(\vec{t}), \sigma)$ by a suitable instance of the right hand side of the equivalence in A's action precondition axiom, and regress that expression. The above renaming of quantified variables of $\Pi_A(\vec{x}, s)$ prevents any of these quantifiers from capturing variables in the instance $Poss(A(\vec{t}), \sigma)$.

(d) W is a relational fluent atom of the form $F(\vec{t}, do(\alpha, \sigma))$. Let F's successor state axiom in $\mathcal{D}_{ss}$ be

$$F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s).$$

Assume that all quantifiers (if any) of $\Phi_F(\vec{x}, a, s)$ have had their quantified variables renamed to be distinct from the free variables (if any) of $F(\vec{t}, do(\alpha, \sigma))$. Then

$$\mathcal{R}[W] = \mathcal{R}[\Phi_F(\vec{t}, \alpha, \sigma)].$$

In other words, replace the atom $F(\vec{t}, do(\alpha, \sigma))$ by a suitable instance of the right hand side of the equivalence in F's successor state axiom, and regress this formula. The above renaming of quantified variables of $\Phi_F(\vec{x}, a, s)$ prevents any of these quantifiers from capturing variables in the instance $F(\vec{t}, do(\alpha, \sigma))$.

2. For non-atomic formulas, regression is defined inductively.

$$\mathcal{R}[\neg W] = \neg \mathcal{R}[W],$$

$$\mathcal{R}[W_1 \wedge W_2] = \mathcal{R}[W_1] \wedge \mathcal{R}[W_2],$$

$$\mathcal{R}[(\exists v)W] = (\exists v)\mathcal{R}[W].$$

- Intuitively, the regression operator eliminates $Poss$ atoms in favour of their definitions as given by action precondition axioms, and replaces fluent atoms about $do(\alpha, \sigma)$ by logically equivalent expressions about σ as given by successor state axioms. Moreover, it repeatedly does this until it cannot make such replacements any further.

- Each $\mathcal{R}$-step reduces the depth of nesting of the function symbol do in the fluents of W by substituting suitable instances of Φ_F for each occurrence of a fluent atom of W of the form $F(t_1, \ldots, t_n, do(\alpha, \sigma))$. Since no fluent atom of Φ_F mentions the function symbol do, the effect of this substitution is to replace each such F by a formula whose fluents mention only the situation term σ, and this reduces the depth of nesting by one.

Theorem 2 (The Regression Theorem) *Suppose W is a regressable sentence of $\mathcal{L}_{sitcalc}$ that mentions no functional fluents, and $\mathcal{D}$ is a basic theory of actions. Then,*

$$\mathcal{D} \models W \quad \textit{iff} \quad \mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \mathcal{R}[W].$$

Executable Action Sequences

- Consider a sequence $\alpha_1, \ldots, \alpha_n$ of ground action terms.
- Problem: Determine whether this is executable. Is it the case that:

$$\mathcal{D} \models executable(do([\alpha_1, \ldots, \alpha_n], S_0))$$

- Exercise:

$$\Sigma \models (\forall a_1, \ldots, a_n).executable(do([a_1, \ldots, a_n], S_0)) \equiv \bigwedge_{i=1}^{n} Poss(\alpha_i, do([\alpha_1, \ldots, \alpha_{i-1}], S_0)).$$

Theorem 3 *Suppose that $\alpha_1, \ldots, \alpha_n$ is a sequence of ground action terms of $\mathcal{L}_{sitcalc}$. Then*

$$\mathcal{D} \models executable(do([\alpha_1, \ldots, \alpha_n], S_0))$$

iff

$$\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \bigwedge_{i=1}^{n} \mathcal{R}[Poss(\alpha_i, do([\alpha_1, \ldots, \alpha_{i-1}], S_0))].$$

- This provides a systematic, regression-based method for determining whether a ground situation $do([\alpha_1, \ldots, \alpha_n], S_0)$ is executable. Moreover, it reduces this test to a theorem-proving task *in the initial database* $\mathcal{D}_{S_0}$, together with unique names axioms for actions.

Example: Executability Testing

- Continue with our previous database example.
- Compute the legality test for the transaction sequence

  $register(Bill, C100), drop(Bill, C100), drop(Bill, C100)$

  which intuitively should fail because the first $drop$ leaves $Bill$ unenrolled in $C100$, so that the precondition for the second $drop$ will be false.

- First compute

  $$\begin{aligned}&\mathcal{R}[Poss(register(Bill, C100), S_0)] \wedge \\ &\mathcal{R}[Poss(drop(Bill, C100), do(register(Bill, C100), S_0))] \wedge \\ &\mathcal{R}[Poss(drop(Bill, C100), do(drop(Bill, C100), \\ &\qquad\qquad do(register(Bill, C100), S_0)))],\end{aligned}$$

  which is

  $$\begin{aligned}&\mathcal{R}[(\forall p).prerequ(p, C100) \supset (\exists g).grade(Bill, p, g, S_0) \wedge g \geq 50] \wedge \\ &\mathcal{R}[enrolled(Bill, C100, do(register(Bill, C100), S_0))] \wedge \\ &\mathcal{R}[enrolled(Bill, C100, do(drop(Bill, C100), \\ &\qquad\qquad do(register(Bill, C100), S_0)))].\end{aligned}$$

- This yields

  $$\begin{aligned}&\{(\forall p).prerequ(p, C100) \supset (\exists g).grade(Bill, p, g, S_0) \wedge g \geq 50\} \wedge \\ &true \wedge \\ &false\end{aligned}$$

- So the transaction sequence is indeed illegal.

Legality Testing: Another Example

- Consider next the sequence

  $change(Bill, C100, 60), register(Sue, C200), drop(Bill, C100).$

- First compute

  $$\begin{array}{l}
\mathcal{R}[(\exists g')grade(Bill, C100, g', S_0) \wedge g' \neq 60] \wedge \\
\mathcal{R}[(\forall p)prerequ(p, C200) \supset \\
\quad (\exists g)grade(Sue, p, g, do(change(Bill, C100, 60), S_0)) \wedge g \geq 50] \wedge \\
\mathcal{R}[enrolled(Bill, C100, do(register(Sue, C200), \\
\qquad\qquad\qquad\qquad do(change(Bill, C100, 60), S_0)))].
\end{array}$$

- This simplifies to

  $$\begin{array}{l}
\{(\exists g').grade(Bill, C100, g', S_0) \wedge g' \neq 60\} \wedge \\
\{(\forall p).prerequ(p, C200) \supset \\
\qquad Bill = Sue \wedge p = C100 \vee (\exists g).grade(Sue, p, g, S_0) \wedge g \geq 50\} \wedge \\
\{Sue = Bill \wedge C200 = C100 \vee enrolled(Bill, C100, S_0)\}.
\end{array}$$

- So the transaction sequence is legal iff this sentence is entailed by the initial database together with uniqe names axioms for actions.

The Projection Problem

- Given an action sequence $\alpha_1, \ldots, \alpha_n$ of *ground* action terms, and a query $Q(s)$ whose only free variable is the situation variable s, what is the answer to Q in that situation resulting from performing this action sequence, beginning with the initial world situation S_0?
- Define this formally as the problem of determining whether

$$\mathcal{D} \models Q(do([\alpha_1, \ldots, \alpha_n], S_0)).$$

- This is the *projection problem.*
- $Q(do([\alpha_1, \ldots, \alpha_n], S_0))$ will normally be regressable.
- So, by the Regression Theorem, regress $Q(do([\alpha_1, \ldots, \alpha_n], S_0))$, and verify it in the initial situation with unique names axioms for actions.

Proj. Prob. Ex: Database Query Evaluation

- Consider again the transaction sequence

  $\mathbf{T} = change(Bill, C100, 60), register(Sue, C200), drop(Bill, C100).$

- Suppose the query is

$$(\exists st).enrolled(st, C200, do(\mathbf{T}, S_0)) \wedge \\ \neg enrolled(st, C100, do(\mathbf{T}, S_0)) \wedge \\ (\exists g).grade(st, C200, g, do(\mathbf{T}, S_0)) \wedge g \geq 50.$$

- Regress this query.

- After some simplification, assuming that $\mathcal{D}_{S_0} \models C100 \neq C200$, we obtain

$$(\exists st).[st = Sue \vee enrolled(st, C200, S_0)] \wedge \\ [st = Bill \vee \neg enrolled(st, C100, S_0)] \wedge \\ [(\exists g).grade(st, C200, g, S_0) \wedge g \geq 50].$$

- The answer to the query is obtained by evaluating this last formula in $\mathcal{D}_{S_0}$, together with unique names axioms for actions.