Regression and Induction on Situations

Adrian Pearce

23 June 2009

includes slides by Ray Reiter & Ryan Kelly

Outline

Regression and Induction on Situations

- Techniques for effective inductive reasoning over situations

Outline

Outline

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 $x < y \vee x = y$.

Motivation (continued)

The second sentence (reproduced below) is a second order induction axiom.

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

It is a second order way of characterising 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).

Motivation (notes)

- 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.

Motivation (notes)

- So why not use the second order axioms? Because second order logic is incomplete, i.e. there is no "decent" axiomatisation 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 characterises the natural numbers. We'll find the same phenomenon in semantically characterising 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's like the number 0 in Peano arithmetic).
- We have a family of successor functions (unlike Peano arithmetic which has a unique successor function).

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

Foundational Axioms (continued)

The axioms:

$$do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2 \quad (1)$$

$$(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \supset (\forall s)P(s) \quad (2)$$

$$\neg s \sqsubset S_0, \quad (3)$$

$$s \sqsubset do(a, s') \equiv s \sqsubseteq s', \quad (4)$$

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

Any model of these axioms will have its domain of situations isomorphic 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.

Foundational Axioms (notes)

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.
- 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).

Foundational Axioms (continued)

Situations are finite sequences of actions.

- $do(C, do(B, do(A, S_0)))$
- 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 foundation axioms are *domain independent*.
  - They will provide the basic properties of situations in any domain specific axiomatisation of particular fluents and actions.

Henceforth, call the 4 foundation axioms 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)

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

More Consequences of the Axioms

Definition

The Principle of Double Induction

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

The Principle of Double Induction (notes)

- These inductive invariants might be e.g. integrity constraints in databases or safety properties in robotics or planning problems.

Executable Situations

Definition

Executable bf situations

Action histories in which it is actually possible to perform the actions one after the other.

$$s < s' \stackrel{\text{def}}{=} s \sqsubset s' \wedge$$

$$(\forall a, s^*).s \sqsubset do(a, s^*) \sqsubseteq s' \supset Poss(a, s^*).$$

$s < s'$ 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.

$$s \leq s' \stackrel{\text{def}}{=} s < s' \vee s = s', \qquad executable(s) \stackrel{\text{def}}{=} S_0 \leq s.$$

Executable Situations (notes)

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.

Induction for executable situations

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

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

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

Definition

The Principle of Induction for *Executable* Situations

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

Double induction for executable situations

Definition

The Principle of *Double* Induction for Executable Situations

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

Frequently, we want to prove sentences (or *inductive invariants*) of the form

$$(\forall s, s').S_0 \sqsubseteq s \ \wedge \ s \sqsubseteq s' \supset R(s, s').$$

Outline

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.

Why Prove Properties (notes)?

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.

Why Prove Properties (notes)?

- 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!

Why Prove Properties (continued)?

Integrity constraints in database theory

Represent integrity constraints, IC, 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

$$Database \models IC.$$

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.

Programming

- Model checking for executions of program relative to (potentially non-deterministic) domain.
- Synthesis of programs and orchestration.
- High-level programming, including Golog, ConGolog, IndiGolog, MIndiGolog

Proving Invariants in Programs & Plans

Goal Impossibility: Given a goal G, establish that there is no legal situation in which that goal is achieved:

$$\mathcal{D} \models \forall s : S_0 \leq_{Legal} s \rightarrow \neg G(s)$$

Goal Futility: Given a goal G and situation σ, establish that the goal cannot be achieved in any legal future of σ:

$$\mathcal{D} \models \forall s : \sigma \leq_{Legal} s \rightarrow \neg G(s)$$

Checking State Constraints: Given a state constraint SC, show that the constraint holds in every legal situation:

$$\mathcal{D} \models \forall s : S_0 \leq_{Legal} s \rightarrow SC(s)$$

This can be seen as a variant of goal impossibility, by showing that the constraint can never be violated.

Proving Invariants in Programs & Plans (Continued)

Note that we define a relation $<_\alpha$ for each *action description predicate* α, with the following definitions:

$$\neg (s <_\alpha S_0)$$
$$s <_\alpha do(a, s') \equiv s \leq_\alpha s' \wedge \alpha(a, s')$$

For example, by stating that $s <_{Poss} s'$ we assert that not only is s' in the future of s, but that all actions performed between s and s' were actually possible;

Why Prove Properties of World Situations? (notes)

Note how futility differs from goal impossibility: while the agent may have initially been able to achieve its goal, the actions that have subsequently been performed have rendered the goal unachievable. Agents would be well advised to avoid such situations.

Summary (so far)

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 axiomatisation, 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:

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

- 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*:

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

Proving Properties of Situations: Another Example

- 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

Outline

Basic theories of actions

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.

Uniform Formulae

Let σ be a term of sort *situation* .

Definition (Uniform Formulae)

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.

- Formulas uniform in s = Markov property. The future (states) are determined by the present (state).

Action precondition and successor state 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 $a, s, x_1, \ldots, x_n$.

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 (Relative Satisfiability)

$\mathcal{D}$ is satisfiable iff $\mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ is.

Relative satisfiability assures us that provided the initial database together with the unique names axioms for actions are satisfiable, then unsatisfiability cannot be introduced by augmenting these with the foundational axioms, together with the action preconditions and successor state axioms. This as

Outline

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.

Examples of regressable formulae: can the actions in the sequence

$walk(A, B, Adrian), train(B, C, Adrian), walk(C, D, Adrian)$ be executed one after the other beginning in S_0?

$Poss(walk(A, B, Adrian), S_0) \wedge$
$Poss(train(B, C, Adrian), do(walk(A, B, Adrian), S_0)) \wedge$
$Poss(walk(C, D, Adrian), do([walk(A, B, Adrian), train(B, C, Adrian)], S_0)$

The following are not regressable:

$(\exists a) Poss(a, S_0),\ \ holding(x, do(walk(A, B, Adrian), s))$.

Another example of regressable formulae: The Gold Thief

The (potential) thief would like to know whether the following action sequence

$walk(A, B, Bruce), enter(bank(Bruce)), crackSafe(Bruce)$

can be executed one after the other beginning in S_0?

$Poss(walk(A, B, Bruce), S_0) \wedge$
$Poss(enter(bank(Bruce)), do(walk(A, B, Bruce), S_0)) \wedge$
$Poss(crackSafe(Bruce), do([walk(A, B, Bruce), enter(bank(Bruce)], S_0)).$

The following are not regressable:

$(\exists a)Poss(a, S_0), \ holding(Gold, do(pickup(Gold, Bruce), s)).$

Regression (notes)

- 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 axiomatisation 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 If W is an atom, there are four possibilities:

1.1 W is a situation independent atom. Then $\mathcal{R}[W] = W$.

1.2 W is a *relational fluent* atom of the form $F(\vec{t}, S_0)$. Then $\mathcal{R}[W] = W$.

1.3 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

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

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

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

The Regression Operator: Simple Version (notes)

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 $Poss(A(\vec{t}), \sigma)$.

The Regression Operator: Simple Version (notes)

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 $F(\vec{t}, do(\alpha, \sigma))$.

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 $F(\vec{t}, do(\alpha, \sigma))$.

The Regression Operator: Simple Version

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

2.1 $\mathcal{R}[\neg W] = \neg \mathcal{R}[W]$,

2.2 $\mathcal{R}[W_1 \wedge W_2] = \mathcal{R}[W_1] \wedge \mathcal{R}[W_2]$,

2.3 $\mathcal{R}[(\exists v)W] = (\exists v)\mathcal{R}[W]$.

The Regression Operator: Simple Version (notes)

- 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.

The regression theorem

Theorem (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,

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

Consider a sequence $\alpha_1, \ldots, \alpha_n$ of ground action terms.

- **Problem**: Determine whether this is executable. Is it the case that:

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

Executable Action Sequences

It is straightforward to show that:

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

Theorem

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

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

Executable Action Sequences (notes)

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: Executibility Testing (notes)

Another 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.

Executibility Testing (Continued)

Legality test for the transaction sequence
$register(Bill, C100), drop(Bill, C100), drop(Bill, C100)$.

First compute

$$\mathcal{R}[Poss(register(Bill, C100), S_0)] \wedge$$
$$\mathcal{R}[Poss(drop(Bill, C100), do(register(Bill, C100), S_0))] \wedge$$
$$\mathcal{R}[Poss(drop(Bill, C100), do(drop(Bill, C100), do(register(Bill, C100), S_0)))],$$

which is

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

This yields $\{(\forall p).prerequ(p, C100) \supset$
$(\exists g).grade(Bill, p, g, S_0) \wedge g \geq 50\} \wedge true \wedge false$

transaction sequence is illegal!

Another Example: Legality Testing

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

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

- This simplifies to

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

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

The Projection Problem

Definition (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? Formally, the problem of determining whether

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

- Note that $Q(do([\alpha_1, \ldots, \alpha_n], S_0))$ will normally be regressable formulae.
- 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.

Projection problem example: Database Query Evaluation

- Consider again the transaction sequence

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

- Suppose the query is

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

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

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

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

Projection problem example: Planning

For example, a projection query for the sequence of actions

$walk(A, B, Adrian), train(B, C, Adrian), walk(C, D, Adrian)$

might be: Will Adrian get home from the Jazz Festival tonight?

$$\mathcal{D} \models getHome(Adrian, do([walk(VillaCelimontana, B, Adrian), train(B, C, Adrian), walk(C, D, Adrian)], S_0)).$$

Regression (re-cap)

Regression operates, intuitively, by unwinding actions one at a time:

$$\mathcal{R}(Holding(agt, obj, do(c, s))) \Rightarrow$$
$$pickup(agt, obj) \in c$$
$$\vee Holding(agt, obj, s) \wedge \neg (drop(agt, obj) \in c)$$

By repeatedly applying it, we get a query about S_0:

$$\mathcal{D} \models \phi[do(c_1, do(c_2, \ldots, S_0))] \text{ iff } \mathcal{D} \models \mathcal{R}^*(\phi)[S_0]$$

If you don't know the current situation, you cannot reason using regression.

Effective reasoning in the situation calculus

- Effective reasoning in the situation calculus is generally based on *syntactic* manipulation of a query into a form that is more amenable to automated reasoning.
- In the general case, answering a query about a basic action theory $\mathcal{D}$ is a theorem-proving task in second-order logic (denoted SOL) due to the induction axiom included in the foundational axioms:

$$\mathcal{D} \models_{SOL} \psi$$

- If a query only performs *existential* quantification over situation terms, it can be answered without the induction axiom (denoted I) and thus using only first-order logic (FOL):

$$\mathcal{D} \models_{SOL} \exists s : \psi(s) \quad \text{iff} \quad \mathcal{D} - \{I\} \models_{FOL} \exists s : \psi(s)$$

Effective reasoning in the situation calculus

- Simpler still, queries uniform in the initial situation, can be answered using only first-order logic and a limited set of axioms:

$$\mathcal{D} \models_{SOL} \phi[S_0] \quad \text{iff} \quad \mathcal{D}_{S_0} \cup \mathcal{D}_{bg} \models_{FOL} \phi[S_0]$$

Outline

Regression: Another example

Recall our thief wants prove

$$\mathcal{D} \models (\exists a, b) : walk(a, b), enter(bank(b), Bob), crackSafe(bank(b), Bob)$$

As owners of the gold, we would like to ensure that the thief cannot steal it. Unfortunately this is not possible, as nothing prevents him from simply cracking the safe and taking the gold.

We can, however, ensure that the thief cannot steal the gold *undetected*.

$$\mathcal{D} \models (\forall s) : s_0 \leq_{Undetected} s \rightarrow \neg Stolen(s)$$

which will be true when the following is true

$$\mathcal{D}_{bg} \cup \mathcal{D}_{S_0} \models \neg Stolen(S_0) \wedge [\neg SafeOpen(S_0) \vee LightOn(S_0)]$$

Regression: Another example (continued)

Unfortunately, this is not possible within the framework of regression presented thus far for two reasons

- $(\forall s) : s_0 \leq_{Undetected} s \supset \neg Stolen(s)$ is not a regressable formulae, since it quantifies over situations.
- The difficulty stems from the second-order induction axiom to define the set of all situations - meaning we can't easily regress over formulae that includes universal quantification.

Property Persistence

Suppose we could "factor out" the quantification. Then we could get on with the business of doing regression.

Definition (Persistence condition)

The *persistence condition* $\mathcal{P}[\phi, \alpha]$ of a formula ϕ and action conditions α to mean: assuming all future actions satisfy α, ϕ will remain true.

$$\mathcal{P}[\phi, \alpha](s) \equiv \forall s' : s \leq_\alpha s' \rightarrow \phi(s')$$

Like $\mathcal{R}$, the idea is to transform a query into a form that is easier to deal with.

Property Persistence

The persistence condition can be calculated as a fixpoint:

$$\mathcal{P}^1[\phi, \alpha](s) \stackrel{\text{def}}{=} \phi(s) \wedge \forall c : \alpha(c) \rightarrow \mathcal{R}[\phi(do(c, s))]$$

$$\mathcal{P}^n[\phi, \alpha](s) \stackrel{\text{def}}{=} \mathcal{P}^1[\mathcal{P}^{n-1}[\phi, \alpha], \alpha]$$

$$\left(\mathcal{P}^n[\phi, \alpha] \rightarrow \mathcal{P}^{n+1}[\phi, \alpha]\right) \Rightarrow \left(\mathcal{P}^n[\phi, \alpha] \equiv \mathcal{P}[\phi, \alpha]\right)$$

This calculation can be done using *static domain reasoning* and provably terminates in several important cases.

Property persistence theorem

The property persistence theorem guarentees that if $\mathcal{P}_{\mathcal{D}}^{n}(\phi, \alpha)$ implies $\mathcal{P}_{\mathcal{D}}^{n+1}(\phi, \alpha)$, then $\mathcal{P}_{\mathcal{D}}^{n}(\phi, \alpha)$ is the persistence condition for ϕ under α.

Theorem

Given a basic action theory $\mathcal{D}$, uniform formula ϕ and action description predicate α, then for any n:

$$\mathcal{D}_{bg} \models \forall s : \mathcal{P}_{\mathcal{D}}^{n}(\phi, \alpha)[s] \rightarrow \mathcal{P}_{\mathcal{D}}^{n+1}(\phi, \alpha)[s]$$

$$\text{iff}$$

$$\mathcal{D} - \mathcal{D}_{s_0} \models \forall s : \mathcal{P}_{\mathcal{D}}^{n}(\phi, \alpha)[s] \equiv \mathcal{P}_{\mathcal{D}}(\phi, \alpha)[s]$$

Calculating Persistence

Define $\mathcal{P}^1[\phi, \alpha]$ to be "persistence to depth 1":

$$\mathcal{P}^1[\phi, \alpha](s) \stackrel{\text{def}}{=} \phi(s) \wedge \forall c.[\alpha(c, s) \Rightarrow \mathcal{R}[\phi(do(c, s))]]$$

We can assert that a property holds to depth 2, 3, ... by repeatedly applying $\mathcal{P}^1$:

$$\mathcal{P}^n[\phi, \alpha] = \mathcal{P}^1[\mathcal{P}^{n-1}[\phi, alpha], \alpha]$$

We want persistence for *any* n: need the least-fixed point of $\mathcal{P}^1$. Fixed-point theory guarantees we can calculate this by trans-finite iteration.

Proving Invariants in Programs & Plans (Continued)

Need for Cooperation: Given an agent *agt*, goal G and situation σ, establish that no sequence of actions performed by that agent can achieve the goal. Suppose we define *MyAction* to identify the agent's own actions:

$$MyAction(a, s) \stackrel{\text{def}}{=} actor(a) = agt$$

Then the appropriate query is:

$$\mathcal{D} \models \forall s : \sigma \leq_{MyAction} s \rightarrow \neg G(s)$$

If this is the case, the agent will need to seek cooperation from another agent in order to achieve its goal.

Proving Invariants in Programs & Plans (Continued)

Knowledge with Hidden Actions: An agent reasoning about its own knowledge in asynchronous domains must account for arbitrarily-long sequences of hidden actions. To establish that it knows ϕ, it must establish that ϕ cannot become false through a sequence of hidden actions:

$$\mathcal{D} \models \forall s : \sigma \leq_{Hidden} s \rightarrow \phi[s]$$

References

- Ray Reiter. Knowledge in Action: Logical Foundations for Specifying and implementing Dynamical Systems, The MIT Press, 2001
- Ray Reiter. The frame problem in situation the calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy, pages 359-380. Academic Press Professional, Inc., 1991.
- Fiora Pirri and Ray Reiter. Some contributions to the metatheory of the situation calculus. Journal of the ACM, 46(3):325-361, 1999.
- Ray Reiter. Proving Properties of States in the Situation Calculus. Artificial Intelligence, 64:337-351, 1993.

References (continued)

- Ryan F. Kelly and Adrian R. Pearce. Property Persistence in the Situation Calculus. In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07), pages 1948-1953, 2007
- Ryan Kelly. "Asynchronous Multi-Agent Reasoning in the Situation Calculus", PhD Thesis, The University of Melbourne, 2009

Summary