Situations

Adrian Pearce

13 July 2011

includes slides by Ray Reiter & Ryan Kelly

Outline

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.

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

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).

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

Foundational Axioms (continued)

The axioms:

$$
\begin{aligned}
do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2 && (1)\\
(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] &&\\
\supset (\forall s)P(s) && (2)\\
\neg s \sqsubset S_0, && (3)\\
s \sqsubset do(a, s') \equiv s \sqsubseteq s', && (4)
\end{aligned}
$$

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

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.

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

$s < s'$ 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.

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

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

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

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

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

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

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

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

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

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:

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

- The sentence to be proved is logically equivalent to:

$$\begin{aligned}&(\forall s, s').executable(s') \land s \sqsubseteq s' \supset \\ &\qquad (\forall p, \$, \$').sal(p, \$, s) \land sal(p, \$', s') \supset \$ \leq \$'.\end{aligned}$$

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

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

- QED

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 Kelly. ”Asynchronous Multi-Agent Reasoning in the Situation Calculus”, PhD Thesis, The University of Melbourne, 2009

Summary

1. Foundations of The Situation Calculus

2. Reasoning about situations