Situation Calculus

- A way of referring to actions and their resulting worlds inside of a single KB
- Worlds also referred to as situations (or states)
- There are two ways to refer to a world:
  - constant init denotes the initial world
  - function do(A,S) denotes the world resulting from doing action A in world S

Worlds and actions are objects in the domain, and are referred to as terms in the logic.

Overview

Planning

⇒ Situation Calculus

Review

Strategy 1: Strips
- Uses logic to reason about how actions change the world
- Does not use logic to reason about how actions change the world
- Uses logic to reason inside of a world

Strategy 2: Situation Calculus
- View domain as having world objects in it that can be referred to
- constant init refers to the initial world
- static relations are true of all worlds
- primitive relations are true of just some worlds
Axiomatizing using Situation Calculus

**Primitive relations**
The situation argument

**Derived relations**
are defined using clauses with a variable in

**Static relations**
are defined without reference to the situation

**Derived relations**
are defined without reference to the situation

You specify what is true in the initial state using axioms with

\[ \text{do}(A,S) \]

Using the Situation Terms

**Example Atoms**

\[ \text{at}(\text{rob},o109,\text{init}) \]
\[ \text{at}(\text{rob},o103,\text{do}(\text{move}(\text{rob},o109,o103),\text{init})) \]
\[ \text{at}(<1>,\text{mail},\text{do}(\text{move}(\text{rob},o109,o103),\text{init})) \]

**Example Axioms**

**Add state variable to primitive & derived predicates**

**Example States**

\[ \text{init} \]
\[ \text{do}(\text{move}(0,100,109),\text{init}) \]
\[ \text{do}(\text{move}(0,103,100),\text{init}) \]

**Example States**
Axiomatizing Primitive Relations

Example: Unlocking the door makes the door unlocked
unlocked(Door, do(unlock(Ag, Door), S)) ← poss(unlock(Ag, Door), S)

Same as:
unlocked(Door, NewS) ← NewS = do(unlock(Ag, Door), PrevS) ∧ poss(unlock(Ag, Door), PrevS)

Frame Axiom:
- Captures under what circumstances a predicate remains true
- For unlock, there is no action that locks a door
- So, if door was unlocked in previous state, it will continue to be true

unlocked(Door, do(A, S)) ← unlocked(Door, S) ∧ poss(A, S)

When are actions possible?

• Need the equivalent of the precondition list of Strips
• poss(A, S) is true if action A is possible in state S

Need to do this for each action

- Possession (A, S)
- Adjacency (A, S)
- Motion (A, S)

Initial Situation

• Static Facts
  - between(door1, o103, lab2).
  - opens(k1, door1).
  - robot(rob).
  ...

• Derived Relations
  - adjacent(pos1, pos2, S) ← nodoorbetween(pos1, pos2).
  - adjacent(pos1, pos2, S) ← doorbetween(Door, pos1, pos2) ∧ unlocked(Door, S).

• Primitive Relations of Initial Situation
  - sitting_at(rob, o109, in).
  - sitting_at(parcel, storage, init).
  - sitting_at(k1, mail, init).

Dotted Relations

...
Dealing with the Quantifiers

This is how Prolog's negation as failure works (no delaying).

• The only actions that undo sitting at for object Obj is when Obj is an agent and moves somewhere or when someone is picking up Obj.

Example: Axiomatizing 'Carrying'

• Picking up an object causes it to be carried

\[ \text{carrying} (\text{Ag}, \text{Obj}, \text{do} (\text{pickup} (\text{Ag}, \text{Obj}), S)) \leftarrow \text{poss} (\text{pickup} (\text{Ag}, \text{Obj}), S) \]

• Frame Axiom: The object is being carried if it was being carried before an action was performed or when object Obj is when Op.

\[ \text{Frame Axiom: } \neg \exists \text{ Op} \neg \exists \text{ Op} \]

[More General Frame Axioms]
Resolution Planning

If you want a plan to achieve Rob holding the key \( k_1 \) and being at \( o_{103} \), you can issue the query

\[
?\text{carrying}(\text{rob}, k_1, S) \land \text{at}(\text{rob}, o_{103}, S)
\]

This has an answer

\[
S = \text{do}(\text{move}(\text{rob}, \text{mail}, o_{103}), \text{do}(\text{pickup}(\text{rob}, k_1), \text{do}(\text{move}(\text{rob}, o_{103}, \text{mail}), \text{do}(\text{move}(\text{rob}, o_{109}, o_{103}), \text{init})))
\]

What strategy should you use to find a solution?

Situation Semantics and Theorem Proving

What kinds of queries can we make?

- \(? \text{poss}(\text{putdown}(\text{rob}, \text{key}), \text{init})\)
- \(? \text{poss}(\text{Action}, \text{init})\)
- \(? \text{poss}(\text{Action}, \text{do}(\text{move}(\text{rob}, o_{109}, o_{103}), \text{init}))\)
- \(? \text{carrying}(\text{rob}, k_1, \text{init})\)
- \(? \text{carrying}(\text{rob}, k_1, \text{do}(\text{pickup}(\text{rob}, k_1), \text{do}(\text{move}(\text{rob}, o_{103}, \text{mail}), \text{do}(\text{move}(\text{rob}, o_{109}, o_{103}), \text{init}))))\)

Overview

Situation Semantics and Theorem Proving
Situation Semantics Forward Planner (Alternate)

How do the two versions compare?

• Create a name for each new state (s1, s2, etc)
• Assert all of the derived facts about it into the KB
• Works easier with holds notation

Similiar to previous one but

Put init in frontier

While frontier is not empty

Take out top world (e.g., s1) from frontier

For each possible action A (determined from poss relation)

Create a new world s2 for each new state (e.g., s1, s2, etc)

Append each new world to frontier

for each Fact s.t. holds(Fact,do(s1,A))

add holds(Fact,s2) to KB

Situation Semantics Forward Planner

• Similar to STRIPS Forward Planner
• Put init in frontier
• Loop

Planning as Resolution

• Could use a top-down depth-first theorem prover
• Could do a top-down breadth-first theorem prover or another strategy

Similar to STRIPS Forward Planner

and work back towards the initial state

Search is called backward-chaining or you start from the goal

Search strategies that are guaranteed to find a solution

Could use a top-down depth-first theorem prover or another

Lastly, every search strategy has a base axiom so that the search space is

Could use a top-down depth-first theorem prover