Situation Calculus

- Situation Calculus is a framework for representing and reasoning about actions and their effects over time.

- States are represented by terms in the logic, as are actions.

- States encode how to get to the state that it denotes.

Overview

Planning

Situation Calculus

STIPS

- STRIPS (Stanfordese Terminology for Representing Information about Physical Systems) is a framework for representing actions and their effects.

- Uses logic to reason inside of a world.

- Derived primitives can describe what happens when a world.

- Does not use first-order logic about how actions change the world.

- Can encode actions and their effects in logic.

- How do we represent actions in logic?

- How do we represent that doing an action in a time point moves us to a different time point?

- How do we represent that doing an action in a time point modifies a predicate or an object?
Axiomatizing using the Situation Calculus

- **Static relations** are defined without reference to the situation
- **Dynamical relations** are defined with clauses with one free variable
  - in situation \( \text{do}(A,S) \)
  - in terms of what holds in situation \( S \)
- **Primitive relations** are axiomatized by specifying what is true in the situation parameter
- **Derived relations** are defined using clauses with a free variable in the situation argument
- **Static relations** are defined without reference to the situation

Using the Situation Terms

- Add state variable to primitive & derived predicates
- Example Atoms
  - \( \text{at}(\text{rob}, \text{o109}, \text{init}) \)
  - \( \text{at}(\text{rob}, \text{o103}, \text{do}(\text{move}(\text{rob}, \text{o109}, \text{o103}), \text{init})) \)
  - \( \text{at}(\text{k1}, \text{mail}, \text{do}(\text{move}(\text{rob}, \text{o109}, \text{o103}), \text{init})) \)

Example States

- \( \text{do}(\text{move}(\text{rob}, \text{o109}, \text{o103}, \text{init})) \)
- \( \text{do}(\text{move}(\text{rob}, \text{o103}, \text{mail}, \text{do}(\text{move}(\text{rob}, \text{o109}, \text{o103}), \text{init}))) \)
- \( \text{do}(\text{pick}(\text{rob}, \text{k1}, \text{do}(\text{move}(\text{rob}, \text{o103}, \text{mail}, \text{do}(\text{move}(\text{rob}, \text{o109}, \text{o103}), \text{init})))))) \)

Example States

- \( \text{do}(\text{move}(\text{rob}, \text{o109}, \text{o103}, \text{init})) \)
- \( \text{do}(\text{move}(\text{rob}, \text{o103}, \text{mail}, \text{do}(\text{move}(\text{rob}, \text{o109}, \text{o103}), \text{init}))) \)
- \( \text{do}(\text{pick}(\text{rob}, \text{k1}, \text{do}(\text{move}(\text{rob}, \text{o103}, \text{mail}, \text{do}(\text{move}(\text{rob}, \text{o109}, \text{o103}), \text{init})))))) \)

Example States
Axiomatizing Primitive Relations

Example: Unlocking the door makes the door unlocked
\[
\text{unlocked}(\text{Door}, \text{do}(\text{unlock}(\text{Ag}, \text{Door}), S)) \iff \text{poss}(\text{unlock}(\text{Ag}, \text{Door}), S)
\]

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

\[
\text{unlocked}(\text{Door}, \text{do}(\text{A}, S)) \iff \text{unlocked}(\text{Door}, S) \land \text{poss}(\text{A}, S)
\]

When are actions possible?

Need the equivalent of the precondition list of Strips

\[
\text{poss}(\text{putdown}(\text{Ag}, \text{Obj}), S) \iff \text{carrying}(\text{Ag}, \text{Obj}, S)
\]
\[
\text{poss}(\text{move}(\text{Ag}, \text{Pos}_1, \text{Pos}_2), S) \iff \text{autonomous}(\text{Ag}) \land \text{adjacent}(\text{Pos}_1, \text{Pos}_2, S) \land \text{sitting}\at(\text{Ag}, \text{Pos}_1, S)
\]

Need to do this for each action

Initial Situation

• Static Facts
  \[
  \text{between}(\text{door}_1, \text{o}_103, \text{lab}_2).
  \text{opens}(\text{k}_1, \text{door}_1).
  \text{autonomous}(\text{rob}).
  \ldots
  \]

• Derived Relations
  \[
  \text{adjacent}(\text{Pos}_1, \text{Pos}_2, S) \iff \neg \text{no orbetween}(\text{Pos}_1, \text{Pos}_2)
  \land \text{doorbetween}(\text{Door}, \text{Pos}_1, \text{Pos}_2) \land \text{unlocked}(\text{Door}, S)
  \]

• Primitive Relations of Initial Situation
  \[
  \text{sitting}\at(\text{rob}, \text{o}_109, \text{int}).
  \text{sitting}\at(\text{parcel}, \text{storage}, \text{init}).
  \text{sitting}\at(\text{r}_1, \text{mail}, \text{init}).
  \]

Facts about the robot:

• Autonomous
• Adjacent
• Sitting at

Initial Situation

 surviving and maintained
 surviving and maintained

Primitive Relations of Initial Situation

Derived Relations

...
Dealing with the Quantifiers

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

sitting/3(Obj,Pos,A) ← poss(A,S) ∧ sitting/3(Obj,Pos,S) ∧ ¬∃Pos1 A = move(Obj,Pos,Pos1) ∧ ¬∃Ag A = pickup(Ag,Obj)

More General Frame Axioms

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

sitting/3(Obj,Pos,A) ← sitting/3(Obj,Pos,S) ∧ poss(A,S) ∧ A ≠ putdown(Ag,Obj)

Example: Axiomatizing 'Carrying'

Picking up an object causes it to be carried

carrying/3(Ag,Obj,do(pickup(Ag,Obj),S)) ← poss(pickup(Ag,Obj),S)

carrying/3(Ag,Obj,do(A,S)) ← carrying/3(Ag,Obj,S) ∧ poss(A,S) ∧ A ≠ putdown(Ag,Obj)
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

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

• This has an answer

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

• What strategy should you use to find a solution?

Situation Semantics and Theorem Proving

• What kinds of queries can we make?

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

Overview

• Situation Calculus $\Rightarrow$ Planning

• Resolution Planning

• Situation Semantics and Theorem Proving

• What kinds of queries can we make?

• Planning

• Situation Calculus

• Overview
Situation Semantics Forward Planner (Alternate)

- Similar to previous one but
  - Create a name for each new state (s1, s2, s3, etc)
  - Assert all of the derived facts about it into the KB
  - Works easier with "holds" notation

```
Put init in frontier
While frontier is not empty
  Take out top world (e.g., si) from frontier
  For each possible action A (determined from poss relation)
    Create a new world sj
    Append sj to frontier
    for each Fact s.t. holds(Fact,do(si,A))
      add holds(Fact,sj) to KB
```

How do the two versions compare?

Situation Semantics Forward Planner

- Similar to STRIPS Forward Planner
- Put init in frontier
- Loop
  - Take out top world (e.g., do(move(rob,01,02),...)) from frontier
  - Figure out all possible actions using poss relation
  - Append resulting world (e.g., do(A,do(move(rob,01,02),...))) to frontier

```
Planning as Resolution
- Could use a top-down depth-first theorem prover
- You can simulate actions in the external space
- Could use a top-down depth-first theorem prover
- Could also use a top-down breadth-first theorem prover
- Figure out all possible actions using poss relation
- You can simulate actions in the external space
```

Situation Semantics Forward Planner (Alternate)

- Similar to previous one but
  - Create a name for each new state (s1, s2, s3, etc)
  - Assert all of the derived facts about it into the KB
  - Works easier with "holds" notation

```
Put init in frontier
While frontier is not empty
  Take out top world (e.g., si) from frontier
  For each possible action A (determined from poss relation)
    Create a new world sj
    Append sj to frontier
    for each Fact s.t. holds(Fact,do(si,A))
      add holds(Fact,sj) to KB
```