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 by terms in the logic (functions, constants, variables).

Planning

⇒ Situation Calculus

Review

Strategy 1: Strips
- Uses logic to reason inside of a world
+ Use theorem proving to prove preconditions of an action are true
- Does not use logic to reason about how actions change world
+ Instead, create a new KB for the new world
- Tried domain is partitioned into operations that can be executed in parallel

Strategy 2: Situation Calculus
- Action instances are objects in the domain
+ Primitive relations are true of just some worlds
- Static relations are true of all worlds
+ World-changing actions may change the truth of static relations

Problem: every KB is at one time a domain and a world.

Axiomatizing using Situation Calculus

- **Primitive relations**: The situation argument is not used.
- **Static relations**: Defined without reference to the situation.
- **Derived relations**: Defined using clauses with a variable in the situation argument.
- **Derived relations**: Defined with reference to the situation.
- **Derived relations**: Specified what is true in the initial state using axioms.
- **Derived relations**: Defined in terms of what holds in the situation.

Using the Situation Terms

- Add state variable to primitive & derived predicates.
- Example atoms:
  - `at(rob, o109, init)`
  - `at(rob, o103, do(move(rob, o109, o103), init))`
  - `at(k1, mail, do(move(rob, o109, o103), init))`

Example States

- `do(move(rob, o109, o103), init)`
- `do(move(rob, o103, mail), do(move(rob, o109, o103), init))`
- `do(pickup(rob, k1), do(move(rob, o103, mail), do(move(rob, o109, o103), init)))`

Example States
Axiomatizing Primitive Relations

- Example: Unlocking the door makes the door unlocked
  
  \[ \text{unlocked}(\text{Door}, \text{do}(\text{unlock}(\text{Ag}, \text{Door}), S)) \]
  
  Same as:
  
  \[ \text{unlocked}(\text{Door}, \text{NewS}) \]
  \[ \text{poss}(\text{unlock}(\text{Ag}, \text{Door}), \text{PrevS}) \]

Frame Axiom:

- Captures under what circumstances a predicate remains true
  
  \[ \text{unlocked}(\text{Door}, \text{do}(\text{A}, S)) \]
  
  \[ \text{unlocked}(\text{Door}, S) \land \text{poss}(\text{A}, S) \]

When are actions possible?

- Need the equivalent of the precondition list of Strips
  
  \[ \text{poss}(\text{putdown}(\text{Ag}, \text{Obj}), S) \]
  
  \[ \text{poss}(\text{move}(\text{Ag}, \text{Pos1}, \text{Pos2}), S) \]
  
  \[ \text{poss}(\text{carrying}(\text{Ag}, \text{Obj}), S) \]
  
  \[ \text{poss}(\text{robot}(\text{Ag}), S) \]
  
  \[ \text{poss}(\text{adjacent}(\text{Pos1}, \text{Pos2}, S)) \]
  
  Need to do this for each action

Initial Situation

- Static Facts
  
  \[ \text{between}(\text{door1}, \text{o103}, \text{lab2}). \]
  
  \[ \text{opens}(\text{k1}, \text{door1}). \]
  
  \[ \text{robot}(\text{rob}). \]
  
  \[ \ldots \]

- Derived Relations
  
  \[ \text{adjacent}(\text{Pos1}, \text{Pos2}, S) \]
  
  \[ \text{nodoorbetween}(\text{Pos1}, \text{Pos2}) \]
  
  \[ \text{doorbetween}(\text{Door}, \text{Pos1}, \text{Pos2}) \]
  
  \[ \text{unlocked}(\text{Door}, S) \]

- Primitive Relations of Initial Situation
  
  \[ \text{sittingat}(\text{rob}, \text{o109}, \text{int}). \]
  
  \[ \text{sittingat}(\text{parcel}, \text{storage}, \text{init}). \]
  
  \[ \text{sittingat}(\text{k1}, \text{mail}, \text{init}). \]

- Initial Facts
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

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

Frame Axiom: The object is being carried if it was being carried before unless the action was to put down the object

Frame Axiom: The object is being carried if it was being carried before unless the action was to put down the object
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

\[
? \text{carrying(rob,} k_1, \text{S)} \land \text{at(rob,} o_103, \text{S)}
\]

This has an answer

\[
S = \text{do(move(rob,} mail, o_103), \text{do(pickup(rob,} k_1), \text{do(move(rob,} o_103,} mail), \\
\text{do(move(rob,} o_109,} o_103), \text{init)})}
\]

What strategy should you use to find a solution?

Situation Semantics and Theorem Proving

- What kinds of queries can we make?
  - \(? \text{poss(putdown(rob,} key),} \text{init)}\)
  - \(? \text{poss(Action)} \text{init}\)
  - \(? \text{poss(Action,} \text{do(move(rob,} o_109,} o_103), \text{init)}\)
  - \(? \text{carrying(rob,} k_1,} \text{init}\)
  - \(? \text{carrying(rob,} k_1, \text{do(pickup(rob,} k_1),}
    \text{do(move(rob,} o_103,} mail),
    \text{do(move(rob,} o_109,} o_103), \text{init)})\)

Situation Semantics and Theorem Proving

- What kinds of queries can we make?

Situation Calculus ⇒ Planning

Overview
How do the two versions compare?

- 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

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

Situation Semantics Forward Planner (Alternate)

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 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?

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

Similar to STRIPS Forward Planner

- Put init in frontier
- Loop

Situation Semantics Forward Planner

- 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

Planning as Resolution

- Could use a top-down depth-first search strategy
- Could use a top-down breadth-first search strategy
- Could use a top-down depth-first search strategy
- Could use a top-down breadth-first search strategy
- Could use a top-down depth-first search strategy

Similar to STRIPS Forward Planner

- 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
- Assert all of the derived facts about it into the KB
- Create a name for each new state (s1, s2, s3, etc)