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 via the add/delete lists

• 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
  - Action instances are also objects in the domain
    + ‘do’ function maps action and world to a new world

Overview

⇒ Situation Calculus
• Planning
**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, if it is possible to do A in S
- Worlds and actions are objects in the domain
  - So are denoted by terms in the logic (functions, constants, variables)

**Example States**

- `init`
  
  - `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)))`
Using the Situation Terms

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

Axiomatizing using Situation Calculus

- You specify what is true in the initial state using axioms with init as the situation parameter
- Static relations are defined without reference to the situation
- Derived relations are defined using clauses with a variable in the situation argument
- Primitive relations are axiomatized by specifying what is true in situation \(\text{do}(A,S)\) in terms of what holds in situation \(S\)
When are actions possible?

- Need the equivalent of the precondition list of Strips
- \( \text{poss}(A,S) \) is true if action \( A \) is possible in state \( S \)
  
  \[
  \text{poss}(\text{putdown}(Ag,Obj),S) \leftarrow \text{carrying}(Ag,Obj,S)
  \]

  \[
  \text{poss}(\text{move}(Ag,Pos1,Pos2),S) \leftarrow \text{robot}(Ag) \land \text{adjacent}(Pos1,Pos2,S) \land \text{sitting.at}(Ag,Pos1,S)
  \]

- Need to do this for each action
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 at 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)

Example: Axiomatizing ‘Carrying’

• Picking up an object causes it to be carried
  carrying(Ag,Obj,do(pickup(Ag,Obj),S)) ←
  poss(pickup(Ag,Obj),S)

• Frame Axiom: The object is being carried if it was being carried
  before unless the action was to put down the object
  carrying(Ag,Obj,do(A,S)) ←
  carrying(Ag,Obj,S) ∧
  poss(A,S) ∧
  A ≠ putdown(Ag,Obj)
More General Frame Axioms

• 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

\[
sitting\text{-}at(Obj,Pos,do(A,S)) \leftarrow
\text{poss}(A,S) \land
sitting\text{-}at(Obj,Pos,S) \land
\forall \ Pos1 \ A \neq move(Obj,Pos,Pos1) \land
\forall \ Ag \ A \neq pickup(Ag,Obj)
\]

• Last two lines are not datalog
  - Is there a way that we can use datalog theorem prover anyways?

• Last line is equivalent to: \(\neg \exists \ Ag \ A = pickup(Ag,Obj)\)

Dealing with the Quantifier

\[
sitting\text{-}at(Obj,Pos,do(A,S)) \leftarrow
\text{poss}(A,S) \land
sitting\text{-}at(Obj,Pos,S) \land
\sim \exists \ Pos1 \ A = move(Obj,Pos,Pos1) \land
\sim \exists \ Ag \ A = pickup(Ag,Obj)
\]

• This is how Prolog’s negation as failure works (no delaying)
  - It fails if it finds any variable instantiation that succeeds
Overview

- Situation Calculus
  ⇒ Planning

Situation Semantics and Theorem Proving

- What kinds of queries can we make?
  - ? poss(putdown(rob,key),init)
  - ? poss(Action,init)
  - ? poss(Action,do(move(rob,o109,o103),init))
  - ? carrying(rob,k1,init)
  - ? carrying(rob,k1,do(pickup(rob,k1),
   do(move(rob,o103,mail),
   do(move(rob,o109,o103),init)))))
Resolution Planning

- If you want a plan to achieve Rob holding the key \( k1 \) and being at \( o103 \), you can issue the query
  \( \text{?carrying(rob,k1,S) \land at(rob,o103,S)} \)

- This has an answer
  \[ S = \text{do(move(rob,mail,o103),} \]
  \[ \text{do(pickup(rob,k1),} \]
  \[ \text{do(move(rob,o103,mail),} \]
  \[ \text{do(move(rob,o109,o103),init)}))} \]

- What strategy should you use to find a solution?

Planning as Resolution

- Could use a top-down depth-first theorem prover
  - You can virtually always use a frame axiom so that the search space is largely unconstrained by the goal
- Could do a top-down breadth-first theorem prover or another search strategy that is guaranteed to find a solution
- Strategy is called backward-chaining, as you start from the goal and work back towards the initial state
Situation Semantics Forward Planner (Alternate)

• 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
    + This will tell you the neighboring worlds
  - For each possible action A
    + Append resulting world (e.g., `do(A,do(move(rob,01,02),...))`) to frontier

• How do the two versions compare?