CSE 473

Chapters 10 & 11
Situation Calculus and Planning

Overview

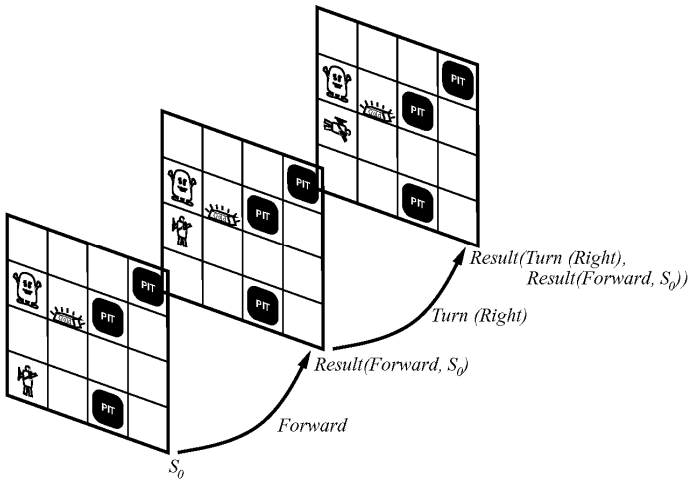
- FOL Planning in Situation Calculus
  Section 10.3 only in Chap. 10
- The Planning Problem (Chap. 11)
- STRIPS Formalism
- Examples
- Progression vs. Regression Planning

Situation Calculus

- Situations: Logical description of world at some point in time
  - Result(a,s) returns next "situation" (state)
- Fluents: Functions and predicates that change over time
  - Holding(G_1, S_4) (where S_4 is a situation)
- Atemporal: Static functions and predicates
  - Gold(G_1)

Situation Calculus

- Result([], s) = s
- Result([a|seq], s) = Result(seq, Result(a, s))



Projection and Planning

- Projection task: Deduce outcome of sequence of actions
- Planning task: Find sequence of actions that achieves desired effect

- Example: Initial Knowledge Base:
  - $At(Agent, [1,1], S_0) \wedge At(G_1, [1,2], S_0) \wedge \neg Holding(G_1, S_0)$
  - $Gold(G_1) \wedge Adjacent([1,1], [1,2]) \wedge Adjacent([1,2], [1,1])$

Projection and Planning

- Projection / prediction:
  - $At(G_1, [1,1], Result([Go([1,1],[1,2]), Grab(G_1), Go([1,2],[1,1])], S_0))$

- Planning Problem:
  - $\exists$ seq $At(G_1, [1,1], Result(seq, S_0))$

Actions in Situation Calculus

- Possibility axioms:
  - At(Agent, x, s) ∧ Adjacent(x,y) ⇒ Poss(Go(x,y),s)
  - Gold(g) ∧ At(Agent,x,s) ∧ At(g, x, s) ⇒ Poss(Grab(g),s)
- Effect axioms:
  - Poss(Go(x,y),s) ⇒ At(Agent, y, Result(Go(x,y),S))
  - Poss(Grab(g),s) ⇒ Holding(g, Result(Grab(g),S))
  - Poss(Release(g),s) ⇒ ¬ Holding(g, Result(Release(g),S))
- Can prove now:
  - At(Agent, [1,2], Result(Go([1,1],[1,2]), S_0))
  - Can't show: At(G_1, [1,2], Result(Go([1,1],[1,2]), S_0))

Frame Problem

- How to handle the things that are NOT changed by an action?
- A actions, E effects per action, F fluents
- Representational frame problem: Size of knowledge base should depend on number of actions and effects, not fluents: O(AE)

Representational Frame Problem

- Naïve solution O(AF):
  - $At(o,x,s) \wedge (o \neq Agent) \wedge \neg Holding(o,s) \Rightarrow At(o,x,Result(Go(y,z),s))$
- Successor-state axioms O(AE):
  - Action possible $\Rightarrow$ (fluent true in result state Action's effect made it true OR It was true before and action didn't change it)
  - $Poss(a,s) \Rightarrow (At(Agent,y,Result(a,s)) \quad a = Go(x,y) \vee (At(Agent,y,s) \wedge a \neq Go(y,z)))$

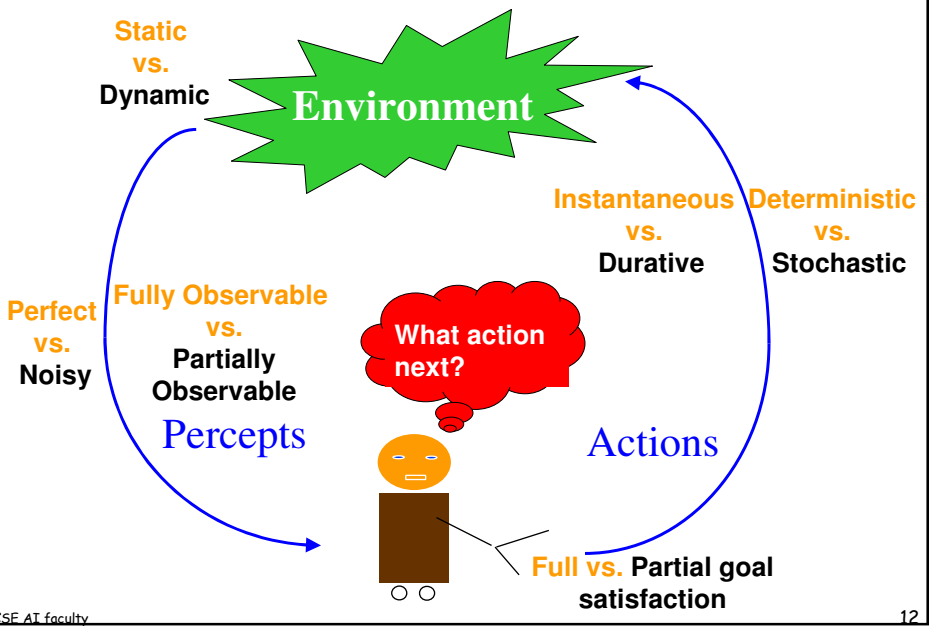
GOLOG

- Robot programming language based on Situation Calculus
- Cognitive robotics
- Extensions can handle concurrent actions, stochastic environments, and sensing
- Was used in museum tour-guide robots
- Still too inefficient due to generality

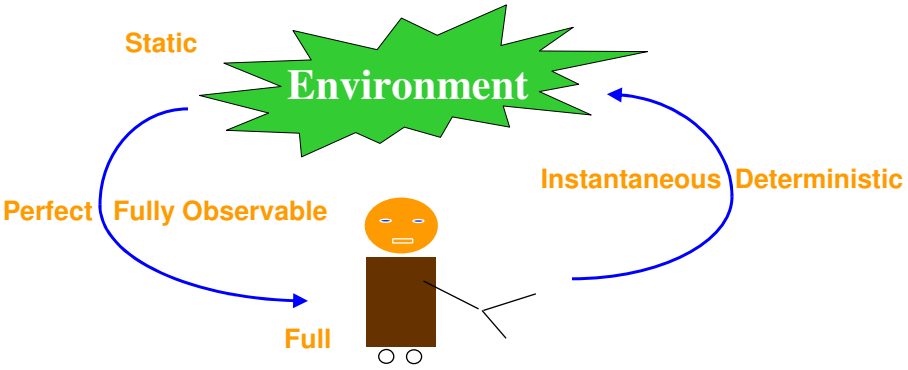
The Planning Problem

- Given
  - a logical description of the **initial situation**,
  - a logical description of the **goal conditions**, and
  - a logical description of a set of **possible actions**,
- Find
  - a **sequence of actions** for going from the initial situation to a goal situation
- Practical applications
  - design and manufacturing
  - military operations
  - games
  - space exploration

General Planning Problem



Classical Planning (this lecture)



I = initial state G = goal state (Prec) A_i (Effects)

[I] A_i → A_j → A_k → A_m [G]

How to Represent Actions?

- *Simplifying assumptions*
  - Atomic time
  - Agent is omniscient (no sensing necessary)
  - Agent is sole cause of change
  - Actions have deterministic effects
- *STRIPS representation*
  - World = set of true propositions (conjunction of literals)
  - Closed Word Assumption (CWA): literals not appearing are assumed false
  - Actions:
    - Precondition: (conjunction of positive literals, ground, no functions)
    - Effects (conjunction of literals, ground, no function)
  - Goal = conjunction of literals

(STRIPS = Stanford Research Institute Problem Solver)

Example: Air cargo transport

Init(At(C1, SFO) ∧ At(C2,JFK) ∧ At(P1,SFO) ∧ At(P2,JFK) ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO))

Goal(At(C1,JFK) ∧ At(C2,SFO))

Action(Load(c,p,a)
PRECOND: *At(c,a) ∧At(p,a) ∧Cargo(c) ∧Plane(p) ∧Airport(a)*
EFFECT: *¬At(c,a) ∧In(c,p))*
Action(Unload(c,p,a)
PRECOND: *In(c,p) ∧At(p,a) ∧Cargo(c) ∧Plane(p) ∧Airport(a)*
EFFECT: *At(c,a) ∧ ¬In(c,p))*
Action(Fly(p,from,to)
PRECOND: *At(p,from) ∧Plane(p) ∧Airport(from) ∧Airport(to)*
EFFECT: *¬ At(p,from) ∧ At(p,to))*

Example Plan:
[Load(C1,P1,SFO), Fly(P1,SFO,JFK), Load(C2,P2,JFK), Fly(P2,JFK,SFO)]

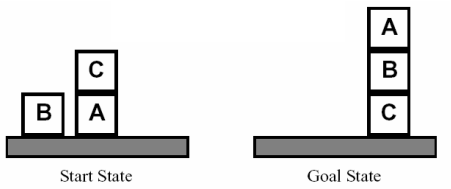
Example: Spare tire problem

Init(At(Flat, Axle) ∧ At(Spare,Trunk))

Goal(At(Spare,Axle))

Action(Remove(Spare,Trunk)
PRECOND: *At(Spare,Trunk)*
EFFECT: *¬At(Spare,Trunk) ∧ At(Spare,Ground))*
Action(Remove(Flat,Axle)
PRECOND: *At(Flat,Axle)*
EFFECT: *¬At(Flat,Axle) ∧ At(Flat,Ground))*
Action(PutOn(Spare,Axle)
PRECOND: *At(Spare,Groundp) ∧¬At(Flat,Axle)*
EFFECT: *At(Spare,Axle) ∧ ¬At(Spare,Ground))*
Action(LeaveOvernight
PRECOND:
EFFECT: *¬ At(Spare,Ground) ∧ ¬ At(Spare,Axle) ∧ ¬ At(Spare,trunk) ∧ ¬ At(Flat,Ground) ∧ ¬ At(Flat,Axle))*

Example: Blocks world



Init(On(A, Table) ∧ On(B, Table) ∧ On(C,A) ∧ Block(A) ∧ Block(B) ∧ Block(C) ∧ Clear(B) ∧ Clear(C))

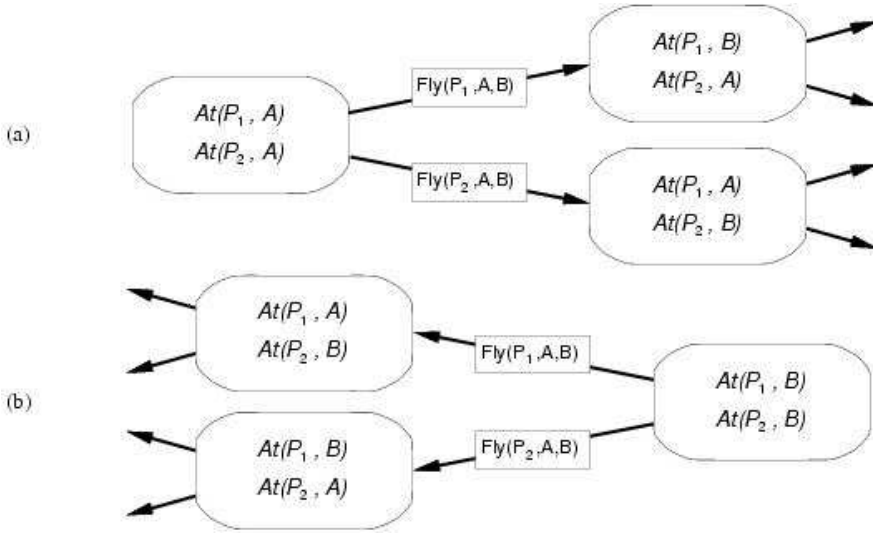
Goal(On(A,B) ∧ On(B,C))

Action(Move(b,x,y)
PRECOND: *On(b,x) ∧ Clear(b) ∧ Clear(y) ∧ Block(b) ∧ (b≠ x) ∧ (b≠ y) ∧ (x≠ y)*
EFFECT: *On(b,y) ∧ Clear(x) ∧ ¬ On(b,x) ∧ ¬ Clear(y))*
Action(Move To Table(b,x)
PRECOND: *On(b,x) ∧ Clear(b) ∧ Block(b) ∧ (b≠ x)*
EFFECT: *On(b, Table) ∧ Clear(x) ∧ ¬ On(b,x))*

Planning with state-space search

- Both forward and backward search possible
- Progression planners
  - Forward state-space search
  - Consider the effect of all possible actions in a given state
- Regression planners
  - Backward state-space search
  - To achieve a goal, what must have been true in the previous state?

Progression vs. Regression Planning



Progression Planning

- Formulation as state-space search problem:
  - Initial state = initial state of the planning problem
    - Literals not appearing are false (CWA)
  - Actions = those whose preconditions are satisfied
    - Add positive effects, delete negative
  - Goal test = does the state satisfy the goal
  - Step cost = each action costs 1
- No functions … any graph search that is complete is a complete planning algorithm.
  - E.g. A*
- Inefficient:
  - (1) irrelevant action problem
  - (2) good heuristic required for efficient search

Next Time

- Regression Planning
- Partial-order Planning
- GraphPlan