Planning
Classical Planning, Strips, Situation Calculus, POP
Chapter 10, Russell & Norvig

Planning

• How to devise plan of actions to reach the goal
• Strategy 1: search based problem solving
• Strategy 2: hybrid logical agent

• Methods
• Situation Calculus
• State Space Search (STRIPS – restricted FOL)
• Plan based search
Planning

- Search based – all the planning is done ahead of time
- Things may have changed between the time of planning and execution of the plan
- Need to interleave planning with executing – specially in case of stochastic env.

Planning

- Stochastic environments
- Multi-agent environments
- Partially observable environments

- Need to plan in order to deal with contingencies
- Lack of knowledge on the world which is incomplete
- Sometimes plans need to be hierarchical

- Instead of the world states – plan in belief states
- And still use the state based strategies
Example: vacuum world

• Deterministic, fully observable
• Actions L, R, S (left, right, suck)
• start in #5. Solution?

• Graph search to get to the goal state 8
• Graph corresponding to actions edges not drawn here

Example: vacuum world

• Sensorless – don’t know where you are, not sure if there is dirt
• start in \{1, 2, 3, 4, 5, 6, 7, 8\}
  Solution?

• Cannot do anything – state is unknown
• Strategy represent belief state
### Example: Vacuum world

- **Deterministic world and partially observable**
  - Partially observable: location, dirt at current location – we have local sensing
  - Outcomes of actions are deterministic
  - Percept: \([L, \text{Clean}]\), i.e., start in \#5 or \#7
    - **Solution?**
  - Planning in belief space
  - each belief state corresponds to a set of worlds stages

```latex
\begin{tabular}{ll}
1 & 2 \\
\end{tabular}
```

### Example: Vacuum world

- **Non-deterministic and/or partially observable**
  - Nondeterministic: Suck may dirty a clean carpet
  - Partially observable: location, dirt at current location.
  - Percept: \([L, \text{Clean}]\), i.e., start in \#5 or \#7
    - **Solution?**

```latex
\begin{tabular}{ll}
1 & 2 \\
3 & 4 \\
5 & 6 \\
7 & 8 \\
\end{tabular}
```
Representations in Planning

Planning opens up the black-boxes by using logic to represent:
- Actions
- States
- Goals

Problem solving Logic representation

Planning

Planning language

- What is a good language?
  - Expressive enough to describe a wide variety of problems.
  - Restrictive enough to allow efficient algorithms to operate on it.
  - Planning algorithm should be able to take advantage of the logical structure of the problem.

- STRIPS Stanford Research Institute Problem Solver (Fikes and Nilson 1971)
General language features

• Representation of states
  – Decompose the world in logical conditions and represent a state as a conjunction of positive literals.
    • Propositional literals: \( \text{Poor} \land \text{Unknown} \)
    • FO-literals (grounded and function-free): \( \text{At}(\text{Plane1}, \text{Melbourne}) \land \text{At}(\text{Plane2}, \text{Sydney}) \)
      • no negation allowed, symbols must be grounded
  – Closed world assumption

• Representation of goals
  – Partially specified state and represented as a conjunction of positive ground literals
  – A goal is satisfied if the state contains all literals in goal.

General language features

• Representations of actions
  – Action = PRECOND + EFFECT
    Action(Fly(p, from, to),
    PRECOND: \( \text{At}(p, \text{from}) \land \text{Plane}(p) \land \text{Airport}(\text{from}) \land \text{Airport}(\text{to}) \)
    EFFECT: \( \neg\text{AT}(p, \text{from}) \land \text{At}(p, \text{to}) \))
    = action schema (p, from, to need to be instantiated)
    • Action name and parameter list
    • Precondition (conj. of function-free literals)
    • Effect (conj of function-free literals and P is True and not P is false)

Add-list vs delete-list in Effect
Add fluents which appear as positive literals in effect
Delete fluents which appear as negative literals in effect
Language semantics?

• How do actions affect states?
  – An action is applicable in any state that satisfies the precondition.
  – Action schema applicability involves a substitution $\theta$ for the variables in the PRECOND.
  – Instantiate free variables
  – E.g. initial state

$$\text{At}(P_1, JFK) \land \text{At}(P_2, SFO) \land \text{Plane}(P_1) \land \text{Plane}(P_2) \land \text{Airport}(JFK)$$

$$\land \text{Airport}(SFO)$$

Satisfies : $\text{At}(p, \text{from}) \land \text{Plane}(p) \land \text{Airport}(\text{from}) \land \text{Airport}(\text{to})$

With $\theta = \{p/P_1, \text{from}/JFK, \text{to}/SFO\}$

Thus the action is applicable.

Language semantics?

• The result of executing action $a$ in state $s$ is the state $s'$
  – $s'$ is same as $s$ except
    • Any positive literal $P$ in the effect of $a$ is added to $s'$
    • Any negative literal $\lnot P$ is removed from $s'$

$\text{EFFECT: } \lnot \text{AT}(p, \text{from}) \land \text{At}(p, \text{to})$: $

\text{At}(P_1, SFO) \land \text{At}(P_2, SFO) \land \text{Plane}(P_1) \land \text{Plane}(P_2) \land \text{Airport}(JFK) \land \text{Airport}(SFO)$$

– STRIPS assumption: (avoids representational frame problem)
  every literal NOT in the effect remains unchanged

• Closed world assumption – any atom not mentioned is false
Expressiveness and extensions

- STRIPS is simplified
  - Important limit: function-free literals
    - Allows for propositional representation
    - Function symbols lead to infinitely many states and actions

- Extension: Action Description language (ADL)

  \[
  \text{Action}(\text{Fly}(p: \text{Plane}, \text{from}: \text{Airport}, \text{to}: \text{Airport}), \\
  \text{PRECOND}: \text{At}(p, \text{from}) \land (\text{from} \neq \text{to}) \\
  \text{EFFECT}: \neg \text{At}(p, \text{from}) \land \text{At}(p, \text{to}))
  \]

  Standardization: Planning domain definition language (PDDL)

Blocks world

The blocks world is a micro-world that consists of a table, a set of blocks and a robot hand.

Some domain constraints:
- Only one block can be on another block
- Any number of blocks can be on the table
- The hand can only hold one block

Typical representation:
- ontable(a)
- ontable(c)
- on(b,a)
- handempty
- clear(b)
- clear(c)
State Representation

Conjunction of propositions:
BLOCK(A), BLOCK(B), BLOCK(C),
ON(A,TABLE), ON(B,TABLE), ON(C,A),
CLEAR(B), CLEAR(C), HANDEMPNY

Goal Representation

Conjunction of propositions:
ON(A,TABLE), ON(B,A), ON(C,B)

The goal G is achieved in a state S if all the propositions in G are also in S
Action Representation

Unstack(x,y)
- **Precondition**: conjunction of propositions
  - \( P = \text{HANDEMP\text{TY}}, \text{BLOCK}(x), \text{BLOCK}(y), \text{CLEAR}(x), \text{ON}(x,y) \)
- **Effect**: list of literals
  - \( E = \neg \text{HANDEMP\text{TY}}, \neg \text{CLEAR}(x), \text{HOLDING}(x), \neg \text{ON}(x,y), \text{CLEAR}(y) \)

Example

Unstack(C,A)
- **Precondition**: conjunction of propositions
  - \( P = \text{HANDEMP\text{TY}}, \text{BLOCK}(C), \text{BLOCK}(A), \text{CLEAR}(C), \text{ON}(C,A) \)
- **Effect**: list of literals
  - \( E = \neg \text{HANDEMP\text{TY}}, \neg \text{CLEAR}(C), \text{HOLDING}(C), \neg \text{ON}(C,A), \text{CLEAR}(A) \)
Example

Unstack(C,A)
- P = HANDEMPTY, BLOCK(C), BLOCK(A), CLEAR(C), ON(C,A)
- E = ¬HANDEMPTY, ¬CLEAR(C), HOLDING(C), ¬ON(C,A), CLEAR(A)

Action Representation

Action(Unstack(x,y))
- P: HANDEMPTY, BLOCK(x), BLOCK(y), CLEAR(x), ON(x,y)
- E: ¬HANDEMPTY, ¬CLEAR(x), HOLDING(x), ¬ON(x,y), CLEAR(y)

Action(Stack(x,y))
- P: HOLDING(x), BLOCK(x), BLOCK(y), CLEAR(y)
- E: ON(x,y), ¬CLEAR(y), ¬HOLDING(x), CLEAR(x), HANDEMPTY

Action(Pickup(x))
- P: HANDEMPTY, BLOCK(x), CLEAR(x), ON(x,TABLE)
- E: ¬HANDEMPTY, ¬CLEAR(x), HOLDING(x), ¬ON(x,TABLE)

Action(PutDown(x))
- P: HOLDING(x)
- E: ON(x,TABLE), ¬HOLDING(x), CLEAR(x), HANDEMPTY
Example: air cargo transport

\[\text{Init}(\text{At}(C_1, \text{SFO}) \land \text{At}(C_2, \text{JFK}) \land \text{At}(P_1, \text{SFO}) \land \text{At}(P_2, \text{JFK}) \land \text{Cargo}(C_1) \land \text{Cargo}(C_2) \land \text{Plane}(P_1) \land \text{Plane}(P_2) \land \text{Airport}(\text{JFK}) \land \text{Airport}(\text{SFO}))\]
\[\text{Goal}(\text{At}(C_1, \text{JFK}) \land \text{At}(C_2, \text{SFO}))\]
\[\text{Action}(\text{Load}(c, p, a))\]
\[\text{PRECOND}: \text{At}(c, a) \land \text{At}(p, a) \land \text{Cargo}(c) \land \text{Plane}(p) \land \text{Airport}(a)\]
\[\text{EFFECT}: \neg \text{At}(c, a) \land \text{In}(c, p)\]
\[\text{Action}(\text{Unload}(c, p, a))\]
\[\text{PRECOND}: \text{In}(c, p) \land \text{At}(p, a) \land \text{Cargo}(c) \land \text{Plane}(p) \land \text{Airport}(a)\]
\[\text{EFFECT}: \text{At}(c, a) \land \neg \text{In}(c, p)\]
\[\text{Action}(\text{Fly}(p, \text{from}, \text{to}))\]
\[\text{PRECOND}: \text{At}(p, \text{from}) \land \text{Plane}(p) \land \text{Airport}(\text{from}) \land \text{Airport}(\text{to})\]
\[\text{EFFECT}: \neg \text{At}(p, \text{from}) \land \text{At}(p, \text{to})\]

\[\text{PLAN}:\]
\[
[\text{Load}(C_1, P_1, \text{SFO}), \ \text{Fly}(P_1, \text{SFO}, \text{JFK}), \ \text{Load}(C_2, P_2, \text{JFK}), \ \text{Fly}(P_2, \text{JFK}, \text{SFO})]\
\]

Example: Spare tire problem

\[\text{Init}(\text{At}(\text{Flat}, \text{Axle}) \land \text{At}(\text{Spare}, \text{trunk}))\]
\[\text{Goal}(\text{At}(\text{Spare}, \text{Axle}))\]
\[\text{Action}(\text{Remove}(\text{Spare}, \text{Trunk}))\]
\[\text{PRECOND}: \text{At}(\text{Spare}, \text{Trunk})\]
\[\text{EFFECT}: \neg \text{At}(\text{Spare}, \text{Trunk}) \land \text{At}(\text{Spare}, \text{Ground})\]
\[\text{Action}(\text{Remove}(\text{Flat}, \text{Axle}))\]
\[\text{PRECOND}: \text{At}(\text{Flat}, \text{Axle})\]
\[\text{EFFECT}: \neg \text{At}(\text{Flat}, \text{Axle}) \land \text{At}(\text{Flat}, \text{Ground})\]
\[\text{Action}(\text{PutOn}(\text{Spare}, \text{Axle}))\]
\[\text{PRECOND}: \text{At}(\text{Spare}, \text{Groundp}) \land \neg \text{At}(\text{Flat}, \text{Axle})\]
\[\text{EFFECT}: \text{At}(\text{Spare}, \text{Axle}) \land \neg \text{At}(\text{Spare}, \text{Ground})\]
\[\text{Action}(\text{LeaveOvernight})\]
\[\text{PRECOND}:\]
\[\text{EFFECT}: \neg \text{At}(\text{Spare}, \text{Ground}) \land \neg \text{At}(\text{Spare}, \text{Axle}) \land \neg \text{At}(\text{Spare}, \text{trunk}) \land \neg \text{At}(\text{Flat}, \text{Ground}) \land \neg \text{At}(\text{Flat}, \text{Axle})\]

This example goes beyond STRIPS: negative literal in pre-condition (ADL description)
Planning with state-space search

- The relationship between planning and state space search has been established
- 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 and regression
Progression algorithm

• Formulation as state-space search problem:
  – Initial state = initial state of the planning problem
    • Literals not appearing are false
  – 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

Progression Example

• Buy(isbn) effect Own(isbn)

• Goal state Own(1237121736813)
• How to reach the goal state
Regression algorithm

• How to determine predecessors?
  – What are the states from which applying a given action leads to the goal?
    Goal state = \( \text{After(C1, B)} \land \text{After(C2, B)} \land \ldots \land \text{After(C20, B)} \)
    Relevant action for first conjunct: Unload(C1,p,B)
    Works only if pre-conditions are satisfied.
    Previous state = \( \text{In(C1, p)} \land \text{At(p, B)} \land \text{After(C2, B)} \land \ldots \land \text{After(C20, B)} \)
    Subgoal At(C1,B) should not be present in this state.

• Actions must not undo desired literals (consistent)
• Main advantage: only relevant actions are considered.
  – Often much lower branching factor than forward search.

Regression algorithm

• General process for predecessor construction
  – Give a goal description G
  – Let A be an action that is relevant and consistent
  – The predecessors is as follows:
    • Any positive effects of A that appear in G are deleted.
    • Each precondition literal of A is added , unless it already appears.

• Any standard search algorithm can be added to perform the search.
• Termination when predecessor satisfied by initial state.
  – In FO case, satisfaction might require a substitution.
Heuristics for state-space search

- Neither progression or regression are very efficient without a good heuristic.
  - How many actions are needed to achieve the goal?
  - Exact solution is NP hard, find a good estimate
- Two approaches to find admissible heuristic:
  - The optimal solution to the relaxed problem.
    - Remove all preconditions from actions
  - The subgoal independence assumption:
    The cost of solving a conjunction of subgoals is approximated by the sum of the costs of solving the subproblems independently.

Situation Calculus Planning

- Formulate planning problem in FOL
- Use theorem prover to find proof (aka plan)

- Instead of using propositional logic
- Use first order logic – enable to use power of universal quantification
- Represent better the notion of time through notion of situation
Representing change

• Representing change in the world in logic can be tricky.
• One way is just to change the KB
  – Add and delete sentences from the KB to reflect changes
  – How do we remember the past, or reason about changes?
• **Situation calculus** is another way
• A **situation** is a snapshot of the world at some instant in time
• When the agent performs an action A in situation S1, the result is a new situation S2.

Stitution Calculus

• Wumpus world
Situation Calculus

Initial state

\[ On(A, Table, s_0) \]
\[ On(B, Table, s_0) \]
\[ On(C, Table, s_0) \]
\[ Clear(A, s_0) \]
\[ Clear(B, s_0) \]
\[ Clear(C, s_0) \]
\[ Clear(Table, s_0) \]

Goal

Find a state (situation) \( s \), such that

\[ On(A, B, s) \]
\[ On(B, C, s) \]
\[ On(C, Table, s) \]

Note: It is not necessary that the goal describes all relations

\[ Clear(A, s) \]
Assume a simpler goal \( \text{On}(A, B, s) \)

**Initial state**

\[
\begin{align*}
\text{On}(A, \text{Table}, s_0) \\
\text{On}(B, \text{Table}, s_0) \\
\text{On}(C, \text{Table}, s_0) \\
\text{Clear}(A, s_0) \\
\text{Clear}(B, s_0) \\
\text{Clear}(C, s_0) \\
\text{Clear}(\text{Table}, s_0)
\end{align*}
\]

\[
\begin{array}{c}
\text{A} \\
\text{B} \\
\text{C}
\end{array}
\]

3 possible goal configurations

\[
\begin{array}{c}
\text{C} \\
\text{A} \\
\text{B}
\end{array}
\]

Goal \( \text{On}(A, B, s) \)

- Knowledge base
- Two types of axioms
  - Effect axioms – describe changes in situations from actions
  - Frame axioms – things preserved from previous situations
**Effect Axioms**

**Effect axioms:**
Moving x from y to z.  \( MOVE (x, y, z) \)

Effect of move changes on **On** relations
\( On(x, y, s) \land Clear(x, s) \land Clear(z, s) \rightarrow On(x, z, DO(MOVE(x, y, z))) \)
\( On(x, y, s) \land Clear(x, s) \land Clear(z, s) \rightarrow \neg On(x, y, DO(MOVE(x, y, z))) \)

Effect of move changes on **Clear** relations
\( On(x, y, s) \land Clear(x, s) \land Clear(z, s) \rightarrow Clear(y, DO(MOVE(x, y, z))) \)
\( On(x, y, s) \land Clear(x, s) \land Clear(z, s) \land (z \neq \text{Table}) \rightarrow \neg Clear(z, DO(MOVE(x, y, z), s)) \)

**Frame axioms**

- Represent things that remain unchanged

**On relations:**
\( On(u, v, s) \land (u \neq x) \land (v \neq y) \rightarrow On(u, v, DO(MOVE(x, y, z), s)) \)

**Clear relations:**
\( Clear(u, s) \land (u \neq z) \rightarrow Clear(u, DO(MOVE(x, y, z), s)) \)
**Blocks World**

Initial state ($s_0$)

\[ s_0 = \]

\[
\begin{align*}
&\text{On}(A, \text{Table}, s_0) \quad \text{Clear}(A, s_0) \quad \text{Clear}(\text{Table}, s_0) \\
&\text{On}(B, \text{Table}, s_0) \quad \text{Clear}(B, s_0) \\
&\text{On}(C, \text{Table}, s_0) \quad \text{Clear}(C, s_0)
\end{align*}
\]

**Action:**  MOVE ($B, \text{Table}, C$)

\[
\begin{align*}
&s_1 = \text{DO}(\text{MOVE}(B, \text{Table}, C), s_0) \\
&\text{On}(A, \text{Table}, s_1) \\
&\text{On}(B, C, s_1) \quad \text{Clear}(A, s_1) \quad \text{Clear}(\text{Table}, s_1) \\
&\neg\text{On}(B, \text{Table}, s_1) \quad \text{Clear}(B, s_1) \\
&\text{On}(C, \text{Table}, s_1) \quad \neg\text{Clear}(C, s_1)
\end{align*}
\]

**Blocks World**

Initial state ($s_0$)

\[ s_0 = \]

\[
\begin{align*}
&\text{On}(A, \text{Table}, s_0) \quad \text{Clear}(A, s_0) \quad \text{Clear}(\text{Table}, s_0) \\
&\text{On}(B, \text{Table}, s_0) \quad \text{Clear}(B, s_0) \\
&\text{On}(C, \text{Table}, s_0) \quad \text{Clear}(C, s_0)
\end{align*}
\]

**Action:**  MOVE ($A, \text{Table}, B$)

\[
\begin{align*}
&s_1 = \text{DO}(\text{MOVE}(A, \text{Table}, B), s_0) \\
&\text{On}(B, C, s_1) \quad \text{Clear}(A, s_1) \quad \text{Clear}(\text{Table}, s_1) \\
&\neg\text{On}(B, \text{Table}, s_1) \quad \text{Clear}(B, s_1) \\
&\text{On}(C, \text{Table}, s_1) \quad \neg\text{Clear}(C, s_1)
\end{align*}
\]

**Blocks World**

Initial state ($s_0$)

\[ s_0 = \]

\[
\begin{align*}
&\text{On}(A, \text{Table}, s_0) \quad \text{Clear}(A, s_0) \quad \text{Clear}(\text{Table}, s_0) \\
&\text{On}(B, \text{Table}, s_0) \quad \text{Clear}(B, s_0) \\
&\text{On}(C, \text{Table}, s_0) \quad \text{Clear}(C, s_0)
\end{align*}
\]

**Action:**  MOVE ($A, \text{Table}, B$)

\[
\begin{align*}
&s_1 = \text{DO}(\text{MOVE}(A, \text{Table}, B), s_0) \\
&\text{On}(B, C, s_1) \quad \text{Clear}(A, s_1) \quad \text{Clear}(\text{Table}, s_1) \\
&\neg\text{On}(B, \text{Table}, s_1) \quad \text{Clear}(B, s_1) \\
&\text{On}(C, \text{Table}, s_1) \quad \neg\text{Clear}(C, s_1)
\end{align*}
\]

**Blocks World**

Initial state ($s_0$)

\[ s_0 = \]

\[
\begin{align*}
&\text{On}(A, \text{Table}, s_0) \quad \text{Clear}(A, s_0) \quad \text{Clear}(\text{Table}, s_0) \\
&\text{On}(B, \text{Table}, s_0) \quad \text{Clear}(B, s_0) \\
&\text{On}(C, \text{Table}, s_0) \quad \text{Clear}(C, s_0)
\end{align*}
\]

**Action:**  MOVE ($A, \text{Table}, B$)

\[
\begin{align*}
&s_1 = \text{DO}(\text{MOVE}(A, \text{Table}, B), s_0) \\
&\text{On}(B, C, s_1) \quad \text{Clear}(A, s_1) \quad \text{Clear}(\text{Table}, s_1) \\
&\neg\text{On}(B, \text{Table}, s_1) \quad \text{Clear}(B, s_1) \\
&\text{On}(C, \text{Table}, s_1) \quad \neg\text{Clear}(C, s_1)
\end{align*}
\]
Situation calculus

• A situation is a snapshot of the world at an interval of time during which nothing changes
• Every true or false statement is made with respect to a particular situation.
  – Add situation variables to every predicate.
  – at(hunter,1,1) becomes at(hunter,1,1,s0): at(hunter,1,1) is true in situation (i.e., state) s0.
• Add a new function, result(a,s), that maps a situation s into a new situation as a result of performing action a. For example, result(forward, s) is a function that returns the successor state (situation) to s
• Example: The action agent-walks-to-location-y could be represented by
  – (∀x)(∀y)(∀s) (at(Agent,x,s) ^ ~onbox(s)) -> at(Agent,y,result(walk(y),s))

Situation calculus planning

• Initial state: a logical sentence about (situation) S₀
  At(Home, S₀) ^ ~Have(Milk, S₀) ^ ~ Have(Bananas, S₀) ^ ~Have(Drill, S₀)
• Goal state:
  (∃s) At(Home,s) ^ Have(Milk,s) ^ Have(Bananas,s) ^ Have(Drill,s)
• Operators are descriptions of actions:
  ∀(a,s) Have(Milk,Result(a,s)) <=> ((a=Buy(Milk) ^ At(Grocery,s)) v (Have(Milk, s) ^ a==Drop(Milk)))
• Result(a,s) names the situation resulting from executing action a in situation s.
• Action sequences are also useful: Result’(l,s) is the result of executing the list of actions (l) starting in s:
  (∀s) Result’([],s) = s
  (∀a,p,s) Result’([a|p]s) = Result’(p,Result(a,s))
Situation calculus planning II

• A solution is thus a plan that when applied to the initial state yields a situation satisfying the goal query:
  \[
  \text{At(Home,Result'(p,S_0))} \\
  \wedge \text{Have(Milk,Result'(p,S_0))} \\
  \wedge \text{Have(Bananas,Result'(p,S_0))} \\
  \wedge \text{Have(Drill,Result'(p,S_0))}
  \]

• Thus we would expect a plan (i.e., variable assignment through unification) such as:
  \[
  p = [\text{Go(Grocery)}, \text{Buy(Milk)}, \text{Buy(Bananas)}, \text{Go(HardwareStore)}, \text{Buy(Drill)}, \text{Go(Home)}]
  \]

SC planning: analysis

• This is fine in theory, but remember that problem solving (search) is exponential in the worst case
• Also, resolution theorem proving only finds a proof (plan), not necessarily a good plan
• Another important issue: the Frame Problem
The Frame Problem

- In SC, need not only axioms to describe what changes in each situation, but also need axioms to describe what stays the same (can do this using successor-state axioms)

- Qualification problem: difficulty in specifying all the conditions that must hold in order for an action to work

- Ramification problem: difficulty in specifying all of the effects that will hold after an action is taken

Partial-order planning

- Progression and regression planning are *totally ordered plan search* forms.
  - They cannot take advantage of problem decomposition.
    - Decisions must be made on how to sequence actions on all the subproblems

- Least commitment strategy:
  - Delay choice during search
Shoe example

Goal(RightShoeOn \land LeftShoeOn)
Init()
Action(RightShoe, \text{ PRECOND: RightSockOn}
  \text{ EFFECT: RightShoeOn})
Action(RightSock, \text{ PRECOND: }
  \text{ EFFECT: RightSockOn})
Action(LeftShoe, \text{ PRECOND: LeftSockOn}
  \text{ EFFECT: LeftShoeOn})
Action(LeftSock, \text{ PRECOND: }
  \text{ EFFECT: LeftSockOn})

Planner: combine two action sequences (1)leftsock, leftshoe
(2)rightsock, rightshoe

Partial-order planning (POP)

- Any planning algorithm that can place two actions into a plan without which comes first is a PO plan.
POP as a search problem

- States are (mostly unfinished) plans.
  - The empty plan contains only start and finish actions.
- Each plan has 4 components:
  - A set of actions (steps of the plan)
  - A set of ordering constraints: A < B (A before B)
    - Cycles represent contradictions.
  - A set of causal links
    - The plan may not be extended by adding a new action C that conflicts with the causal link. (if the effect of C is ¬p and if C could come after A and before B) $A \rightarrow p \rightarrow B$
  - A set of open preconditions.
    - If precondition is not achieved by action in the plan.

Example of final plan

- Actions={Rightsock, Rightshoe, Leftsock, Leftshoe, Start, Finish}
- Orderings={Rightsock < Rightshoe; Leftsock < Leftshoe}
- Links={Rightsock->Rightsockon -> Rightshoe, Leftsock->Leftsockon-> Leftshoe, Rightshoe->Rightshoeon->Finish, …}
- Open preconditions={}
POP as a search problem

- A plan is *consistent* iff there are no cycles in the ordering constraints and no conflicts with the causal links.
- A consistent plan with no open preconditions is a *solution*.
- A partial order plan is executed by repeatedly choosing *any* of the possible next actions.
  - This flexibility is a benefit in non-cooperative environments.

Solving POP

- Assume propositional planning problems:
  - The initial plan contains *Start* and *Finish*, the ordering constraint *Start < Finish*, no causal links, all the preconditions in *Finish* are open.
  - Successor function:
    - picks one open precondition $p$ on an action $B$ and
    - generates a successor plan for every possible consistent way of choosing action $A$ that achieves $p$.
  - Test goal
Enforcing consistency

• When generating successor plan:
  – The causal link $A \rightarrow p \rightarrow B$ and the ordering constraint $A < B$ is added to the plan.
    • If $A$ is new also add start $< A$ and $A < B$ to the plan
  – Resolve conflicts between new causal link and all existing actions
  – Resolve conflicts between action $A$ (if new) and all existing causal links.

Process summary

• Operators on partial plans
  – Add link from existing plan to open precondition.
  – Add a step to fulfill an open condition.
  – Order one step w.r.t another to remove possible conflicts
• Gradually move from incomplete/vague plans to complete/correct plans
• Backtrack if an open condition is unachievable or if a conflict is irresolvable.
Example: Spare tire problem

\begin{align*}
\text{Init}( & \text{At(Flat, Axle) } \land \text{At(Spare, trunk)}) \\
\text{Goal}( & \text{At(Spare, Axle)}) \\
\text{Action}( & \text{Remove(Spare, Trunk)} \\
\text{PRECOND: } & \text{At(Spare, Trunk)} \\
\text{EFFECT: } & \lnot \text{At(Spare, Trunk) } \land \text{At(Spare, Ground))} \\
\text{Action}( & \text{Remove(Flat, Axle)} \\
\text{PRECOND: } & \text{At(Flat, Axle)} \\
\text{EFFECT: } & \lnot \text{At(Flat, Axle) } \land \text{At(Flat, Ground))} \\
\text{Action}( & \text{PutOn(Spare, Axle)} \\
\text{PRECOND: } & \text{At(Spare, Ground) } \land \lnot \text{At(Flat, Axle)} \\
\text{EFFECT: } & \text{At(Spare, Axle) } \land \lnot \text{At(Spare, Ground}}) \\
\text{Action}( & \text{LeaveOvernight} \\
\text{PRECOND: } & \\
\text{EFFECT: } & \lnot \text{At(Spare, Ground) } \land \lnot \text{At(Spare, Axle) } \land \lnot \text{At(Spare, trunk) } \land \lnot \text{At(Flat, Ground)} \land \lnot \text{At(Flat, Axle)})
\end{align*}

POP Solving the problem

- Remove LeaveOverNight and causal links
- Add RemoveFlatAxle and finish
Solving the problem

- Initial plan: Start with EFFECTS and Finish with PRECOND.
- Idea: search through the space of plans as opposed state space
- Start – empty plan
- Flaw – nothing achieves the goal
- Insert – PutOn action
- Flaw – nothing achieves the precondition – keep on adding

Solving the problem

- Initial plan: Start with EFFECTS and Finish with PRECOND.
- Pick an open precondition: \textit{At(Spare, Axle)}
- Only \textit{PutOn(Spare, Axle)} is applicable
- Add causal link: \textit{PutOn(Spare, Axle) \rightarrow At(Spare, Axle) \rightarrow Finish}
- Add constraint : \textit{PutOn(Spare, Axle) < Finish}
Solving the problem

- Pick an open precondition: $\text{At}(\text{Spare}, \text{Ground})$
- Only $\text{Remove}(\text{Spare}, \text{Trunk})$ is applicable
- Add causal link: $\text{Remove}(\text{Spare}, \text{Trunk}) \rightarrow \text{PutOn}(\text{Spare}, \text{Axle})$
- Add constraint: $\text{Remove}(\text{Spare}, \text{Trunk}) < \text{PutOn}(\text{Spare}, \text{Axle})$

Solving the problem

- Pick an open precondition: $\neg \text{At}(\text{Flat}, \text{Axle})$
- $\text{LeaveOverNight}$ is applicable
- conflict: $\text{LeaveOverNight}$ also has the effect $\neg \text{At}(\text{Spare}, \text{Ground})$
- $\text{Remove}(\text{Spare}, \text{Trunk}) \rightarrow \text{PutOn}(\text{Spare}, \text{Axle})$
- To resolve, add constraint: $\text{LeaveOverNight} < \text{Remove}(\text{Spare}, \text{Trunk})$
Solving the problem

• Pick an open precondition: \textit{At(Spare, Trunk)}
• Only \textit{Start} is applicable
• Add causal link: \textit{Start} \rightarrow \textit{Remove(Spare,Trunk)}
• Conflict: of causal link with effect \textit{At(Spare,Trunk)} in \textit{LeaveOverNight}
  – No re-ordering solution possible.
• backtrack

Solving the problem

• Remove \textit{LeaveOverNight} and causal links
• Add \textit{RemoveFlatAxle} and finish
Planning graphs

- Used to achieve better heuristic estimates.
  - A solution can also directly extracted using GRAPHPLAN.

- Uses special data structure that can be used to give better heuristics

- Consists of a sequence of levels that correspond to time steps in the plan.
  - Level 0 is the initial state.
  - Each level consists of a set of literals and a set of actions.
    - *Literals* = all those that *could* be true at that time step, depending upon the actions executed at the preceding time step.
    - *Actions* = all those actions that *could* have their preconditions satisfied at that time step, depending on which of the literals actually hold.

Planning graphs

- “Could”?
  - Records only a restricted subset of possible negative interactions among actions.

- They work only for propositional problems.

- Example:
  
  ```
  Init(Have(Cake))
  Goal(Have(Cake) ∧ Eaten(Cake))
  Action(Eat(Cake), PRECOND: Have(Cake)
  EFFECT: ¬Have(Cake) ∧ Eaten(Cake))
  Action(Bake(Cake), PRECOND: ¬Have(Cake)
  EFFECT: Have(Cake))
  ```
Cake example

- Start at level S0 and determine action level A0 and next level S1.
  - A0 -> all actions whose preconditions are satisfied in the previous level.
  - Connect precond and effect of actions S0 -> S1
  - Inaction is represented by persistence actions.
- Level A0 contains the actions that could occur
  - Conflicts between actions are represented by mutex links

Cake example

- Level S1 contains all literals that could result from picking any subset of actions in A0
  - Conflicts between literals that can not occur together (as a consequence of the selection action) are represented by mutex links.
  - S1 defines multiple states and the mutex links are the constraints that define this set of states.
- Continue until two consecutive levels are identical: leveled off
  - Or contain the same amount of literals (explanation follows later)
**Cake example**

- A mutex relation holds between **two actions** when:
  - *Inconsistent effects*: one action negates the effect of another.
  - *Interference*: one of the effects of one action is the negation of a precondition of the other.
  - *Competing needs*: one of the preconditions of one action is mutually exclusive with the precondition of the other.

- A mutex relation holds between **two literals** when (**inconsistent support**):
  - If one is the negation of the other OR
  - if each possible action pair that could achieve the literals is mutex.

**PG and heuristic estimation**

- PG’s provide information about the problem
  - A literal that does not appear in the final level of the graph cannot be achieved by any plan.
    - Useful for backward search (cost = inf).
  - Level of appearance can be used as cost estimate of achieving any goal literals = *level cost*.
  - Small problem: several actions can occur
    - Restrict to one action using serial PG (add mutex links between every pair of actions, except persistence actions).
    - Cost of a conjunction of goals? Max-level, sum-level and set-level heuristics.

PG is a relaxed problem.
The GRAPHPLAN Algorithm

- How to extract a solution directly from the PG

```
function GRAPHPLAN(problem) return solution or failure
graph ← INITIAL-PLANNING-GRAPH(problem)
goals ← GOALS[problem]
loop do
  if goals all non-mutex in last level of graph then do
    solution ← EXTRACT-SOLUTION(graph, goals, LENGTH(graph))
    if solution ≠ failure then return solution
  else if NO-SOLUTION-POSSIBLE(graph) then return failure
  graph ← EXPAND-GRAPH(graph, problem)
```

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)

This example goes beyond STRIPS: negative literal in pre-condition (ADL description)
```
Initially the plan consists of 5 literals from the initial state and the CWA literals (S0).

- Add actions whose preconditions are satisfied by EXPAND-GRAPH (A0)
- Also add persistence actions and mutex relations.
- Add the effects at level S1
- Repeat until goal is in level Si

EXPAND-GRAPH also looks for mutex relations
- Inconsistent effects
  - E.g. Remove(Spare, Trunk) and LeaveOverNight due to At(Spare, Ground) and not At(Spare, Ground)
- Interference
  - E.g. Remove(Flat, Axle) and LeaveOverNight At(Flat, Axle) as PRECOND and not At(Flat, Axle) as EFFECT
- Competing needs
  - E.g. PutOn(Spare, Axle) and Remove(Flat, Axle) due to At(Flat, Axle) and not At(Flat, Axle)
- Inconsistent support
  - E.g. in S2, At(Spare, Axle) and At(Flat, Axle)
• In S2, the goal literals exist and are not mutex with any other
  – Solution might exist and EXTRACT-SOLUTION will try to find it
• EXTRACT-SOLUTION can use Boolean CSP to solve the problem or a search process:
  – Initial state = last level of PG and goal goals of planning problem
  – Actions = select any set of non-conflicting actions that cover the goals in the state
  – Goal = reach level S0 such that all goals are satisfied
  – Cost = 1 for each action.

• Termination? YES
• PG are monotonically increasing or decreasing:
  – Literals increase monotonically
  – Actions increase monotonically
  – Mutexes decrease monotonically
• Because of these properties and because there is a finite number of actions and literals, every PG will eventually level off!
Planning with propositional logic

- Planning can be done by proving theorem in situation calculus.
- Here: test the satisfiability of a logical sentence:

\[ \text{initial state} \land \text{all possible action descriptions} \land \text{goal} \]

- Sentence contains propositions for every action occurrence.
  - A model will assign true to the actions that are part of the correct plan and false to the others
  - An assignment that corresponds to an incorrect plan will not be a model because of inconsistency with the assertion that the goal is true.
  - If the planning is unsolvable the sentence will be unsatisfiable.

Analysis of planning approach

- Planning is an area of great interest within AI
  - Search for solution
  - Constructively prove a existence of solution
- Biggest problem is the combinatorial explosion in states.
- Efficient methods are under research
  - E.g. divide-and-conquer