Planning

Planning problem:
• find a sequence of actions that achieves some goal
• an instance of a search problem
• the state description is typically very complex and relies on a logic-based representation

Methods for modeling and solving a planning problem:
• State space search
• Situation calculus based on FOL
• STRIPS – state-space search algorithm
• Partial-order planning algorithms
Situation calculus

Provides a framework for representing change, actions and for reasoning about them

- **Situation calculus**
  - based on the first-order logic,
  - a situation variable models possible states of the world
  - properties and relations among objects depend on different world states (situations)
  - action objects model activities
- **Inference:**
  - inference methods developed for FOL to do the reasoning

Situation calculus. Blocks world example.

\[
\begin{array}{ccc}
A & B & C \\
\hline
\text{Initial state} \\
\text{On}(A, \text{Table}, s_0) \\
\text{On}(B, \text{Table}, s_0) \\
\text{On}(C, \text{Table}, s_0) \\
\text{Clear}(A, s_0) \\
\text{Clear}(B, s_0) \\
\text{Clear}(C, s_0) \\
\text{Clear}(\text{Table}, s_0) \\
\end{array}
\]

\[
\begin{array}{c}
A \\
B \\
C \\
\hline
\text{Goal} \\
\text{On}(A, B, s) \\
\text{On}(B, C, s) \\
\text{On}(C, \text{Table}, s) \\
\end{array}
\]

Find a state (situation) \( s \), such that
Knowledge base: Axioms

Knowledge base is needed to support the reasoning:
• Must represent changes in the world due to actions.

Two types of axioms:
• **Effect axioms**
  – changes in situations that result from actions
• **Frame axioms**
  – things preserved from the previous situation

Planning in situation calculus

**Planning problem:**
• find a sequence of actions that lead to the goal

**Planning in situation calculus is converted to the theorem proving problem**

**Goal state:**
\[ \exists s \ On(A, B, s) \land On(B, C, s) \land On(C, Table, s) \]

• Possible inference approaches:
  – **Inference rule approach**
  – **Conversion to SAT**
• **Plan** (solution) is a byproduct of theorem proving
• **Example:** blocks world
Planning in the blocks world.

Initial state (s0)

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

Action: \( MOVE(B, Table, C) \)

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

Planning in the blocks world.

Initial state (s0) \( \rightarrow \) s1 \( \rightarrow \) s2

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

Action: \( MOVE(A, Table, B) \)

\[ s_2 = DO(MOVE(A, Table, B), s_1) = DO(MOVE(A, Table, B), DO(MOVE(B, Table, C), s_0)) \]
\[ \begin{align*}
  & On(A, B, s_2) \quad \neg On(A, Table, s_2) \quad \neg Clear(B, s_2) \\
  & On(B, C, s_2) \quad \neg On(B, Table, s_2) \quad \neg Clear(C, s_2) \\
  & On(C, Table, s_2) \quad Clear(A, s_2) \quad Clear(Table, s_2)
\end{align*} \]
Planning in the blocks world.

Initial state (s0) s1 s2

\[ s_1 = \text{DO}(\text{MOVE} (B, \text{Table}, C), s_0) \]
\[ \text{On}(A, \text{Table}, s_1) \]
\[ \text{On}(B, C, s_1) \]
\[ \text{Clear}(A, s_1) \]
\[ \text{Clear}(B, s_1) \]
\[ \neg\text{On}(B, \text{Table}, s_1) \]
\[ \text{Clear}(C, s_1) \]
\[ \text{On}(C, \text{Table}, s_1) \]

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

Initial state (s0) s1 s2

\[ s_2 = \text{DO}(\text{MOVE} (A, \text{Table}, B), s_1) \]
\[ = \text{DO}(\text{MOVE} (A, \text{Table}, B), \text{DO}(\text{MOVE} (B, \text{Table}, C), s_0)) \]

\[ \text{On}(A, B, s_2) \]
\[ \text{On}(B, C, s_2) \]
\[ \text{On}(C, \text{Table}, s_2) \]

DO functions capture the plan

Satisfies the goal
Situation calculus: problems

**Frame problem** refers to:
- The need to represent a large number of frame axioms

**Solution:** combine positive and negative effects in one rule

\[ \text{On}(u, v, \text{DO}(\text{MOVE} (x, y, z), s)) \leftrightarrow (\neg ((u = x) \land (v = y)) \land \text{On}(u, v, s)) \lor \]

\[ \lor (((u = x) \land (v = z)) \land \text{On}(x, y, s) \land \text{Clear} (x, s) \land \text{Clear} (z, s)) \]

**Inferential frame problem:**
- We still need to derive properties that remain unchanged

**Other problems:**
- **Qualification problem** – enumeration of all possibilities under which an action holds
- **Ramification problem** – enumeration of all inferences that follow from some facts

Planning in situation calculus.

**Planning problem:**
- Find a sequence of actions that lead to a goal
- Is a special type of a search problem
- Planning in situation calculus is converted to theorem proving.

**Limitations:**
- Large search space
- Large number of axioms to be defined for one action
- Proof may not lead to the best (shortest) plan.
Planning problems

Properties of many (real-world) planning problems:
- The description of the state of the world is very complex
- Many possible actions to apply in any step
- Actions are typically local
  - they affect only a small portion of a state description
- Goals are defined as conditions referring only to a small portion of state
- Plans consists of a large number of actions

The state space search and situation calculus frameworks:
- too cumbersome and inefficient to represent and solve the planning problems

Solutions

- **Complex state description and local action effects:**
  - avoid the enumeration and inference of every state component, focus on changes only

- **Many possible actions:**
  - Apply actions that make progress towards the goal
  - Understand what the effect of actions is and reason with the consequences of these action

- **Sequences of actions in the plan can be too long:**
  - Many goals consists of independent or nearly independent sub-goals
  - Allow goal decomposition & divide and conquer strategies
STRIPS planner

Defines a **restricted representation language** when compared to the situation calculus

**Advantage:** leads to more efficient planning algorithms.
- State-space search with structured representations of states, actions and goals
- Action representation avoids the frame problem

**STRIPS planning problem:**
- much like a standard search problem

---

**STRIPS planner**

- **States:**
  - conjunction of literals, e.g. \(\text{On}(A,B), \text{On}(B,\text{Table}), \text{Clear}(A)\)
  - represent facts that are true at a specific point in time
- **Actions (operators):**
  - **Action:** \(\text{Move}(x,y,z)\)
  - **Preconditions:** conjunctions of literals with variables
    \(\text{On}(x,y), \text{Clear}(x), \text{Clear}(z)\)
  - **Effects.** Two lists:
    - **Add list:** \(\text{On}(x,z), \text{Clear}(y)\)
    - **Delete list:** \(\text{On}(x,y), \text{Clear}(z)\)
    - Everything else remains untouched (is preserved)
STRIPS planning

**Operator:** Move (x,y,z)

- **Preconditions:** \( \text{On}(x,y), \text{Clear}(x), \text{Clear}(z) \)
- **Add list:** \( \text{On}(x,z), \text{Clear}(y) \)
- **Delete list:** \( \text{On}(x,y), \text{Clear}(z) \)

---

**Initial state:**
- Conjunction of literals that are true

**Goals in STRIPS:**
- A goal is a partially specified state
- Is defined by a conjunction of ground literals
  - No variables allowed in the description of the goal

Example:
\[
\text{On}(A,B) \land \text{On}(B,C)
\]
Search in STRIPS

Objective:
Find a sequence of operators (a plan) from the initial state to the state satisfying the goal

Two approaches to build a plan:
• **Forward state space search (goal progression)**
  – Start from what is known in the initial state and apply operators in the order they are applied
• **Backward state space search (goal regression)**
  – Start from the description of the goal and identify actions that help us to reach the goal

---

**Forward search (goal progression)**

• **Idea:** Given a state \( s \)
  – Unify the preconditions of some operator \( a \) with \( s \)
  – Add and delete sentences from the add and delete list of an operator \( a \) from \( s \) to get a new state

\[
\begin{array}{c}
\text{On}(B, \text{Table}) \\
\text{Clear}(C)
\end{array}
\]  

\[
\begin{array}{c}
\text{On}(A, \text{Table}) \\
\text{On}(C, \text{Table}) \\
\text{Clear}(A) \\
\text{Clear}(B) \\
\text{Clear}(\text{Table})
\end{array}
\]  

\[
\begin{array}{c}
\text{On}(B, \text{Table}) \\
\text{Clear}(C)
\end{array}
\]  

\[
\begin{array}{c}
\text{On}(A, \text{Table}) \\
\text{On}(C, \text{Table}) \\
\text{Clear}(A) \\
\text{Clear}(B) \\
\text{Clear}(\text{Table})
\end{array}
\]  

\[
\begin{array}{c}
\text{On}(B, \text{C})
\end{array}
\]  

\[
\begin{array}{c}
\text{On}(A, \text{Table}) \\
\text{On}(C, \text{Table}) \\
\text{Clear}(A) \\
\text{Clear}(B) \\
\text{Clear}(\text{Table})
\end{array}
\]  

\[
\begin{array}{c}
\text{On}(B, \text{Table}) \\
\text{Clear}(C)
\end{array}
\]  

\[
\begin{array}{c}
\text{On}(A, \text{Table}) \\
\text{On}(C, \text{Table}) \\
\text{Clear}(A) \\
\text{Clear}(B) \\
\text{Clear}(\text{Table})
\end{array}
\]  

Move \((B, \text{Table}, C)\)
Forward search (goal progression)

- Use operators to generate new states to search
- Check new states whether they satisfy the goal

Search tree:

Initial state

\[ A \rightarrow B \rightarrow C \rightarrow \text{goal} \]

Heuristics?
Backward search (goal regression)

**Idea:** Given a goal \( G \)
- Unify the add list of some operator \( a \) with a subset of \( G \)
- If the delete list of \( a \) does not remove elements of \( G \), then the goal regresses to a new goal \( G' \) that is obtained from \( G \) by:
  - deleting add list of \( a \)
  - adding preconditions of \( a \)

<table>
<thead>
<tr>
<th>New goal (( G' ))</th>
<th>Goal (( G ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{On}(A,\text{Table}) )</td>
<td>( \text{On}(A,B) )</td>
</tr>
<tr>
<td>( \text{Clear}(B) )</td>
<td>( \text{On}(B,C) )</td>
</tr>
<tr>
<td>( \text{Clear}(A) )</td>
<td>( \text{On}(C,\text{Table}) )</td>
</tr>
<tr>
<td>( \text{On}(B,C) )</td>
<td></td>
</tr>
<tr>
<td>( \text{On}(C,\text{Table}) )</td>
<td></td>
</tr>
</tbody>
</table>

Maped from \( G \)

---

Backward search (goal regression)

- Use operators to generate new goals
- Check whether the initial state satisfies the goal

**Search tree:**
State-space search

- **Forward and backward state-space planning approaches:**
  - Work with strictly linear sequences of actions

- **Disadvantages:**
  - They cannot take advantage of the *problem decompositions* in which the goal we want to reach consists of a set of independent or nearly independent sub-goals
  - Action sequences cannot be *built from the middle*
  - No mechanism to represent *least commitment* in terms of the action ordering

---

Divide and conquer

- **Divide and conquer strategy:**
  - divide the problem to a set of smaller sub-problems,
  - solve each sub-problem independently
  - combine the results to form the solution

In planning we would like to satisfy a set of goals
- **Divide and conquer in planning:**
  - Divide the planning goals along individual goals
  - Solve (find a plan for) each of them independently
  - Combine the plan solutions in the resulting plan

- Is it always safe to use divide and conquer?
  - No. There can be interacting goals.
Sussman’s anomaly.

- An example from the blocks world in which the divide and conquer fails due to interacting goals

```
Initial state: C A B
Goal: A B C
On(A, B)
On(B, C)
```

---

1. Assume we want to satisfy $On(A, B)$ first

```
Initial state: C A B
But now we cannot satisfy $On(B, C)$ without undoing $On(A, B)$
```

---
**Sussman’s anomaly**

1. Assume we want to satisfy $On(A, B)$ first

   ![Initial state diagram](image)

   But now we cannot satisfy $On(B, C)$ without undoing $On(A, B)$

2. Assume we want to satisfy $On(B, C)$ first.

   ![Initial state diagram](image)

   But now we cannot satisfy $On(A, B)$ without undoing $On(B, C)$

---

**State space vs. plan space search**

- An alternative to planning algorithms that search states (configurations of world)
- **Plan**: Defines a sequence of operators to be performed
- **Partial plan**:  
  - plan that is not complete  
    - Some plan steps are missing  
      - Some orderings of operators are not finalized  
      - Only relative order is given
- **Benefits of working with partial plans**:  
  - We do not have to build the sequence from the initial state or the goal  
  - We do not have to commit to a specific action sequence  
  - We can work on sub-goals individually (divide and conquer)
State-space vs. plan-space search

State-space search

STRIPS operator

\[ S_0 \xrightarrow{\text{operator}} S_1 \]

State (set of formulas)

\[ S_2 \]

Plan-space search

Plan transformation operators

Start

Finish

Incomplete (partial) plan

Plan transformation operators

Examples of:

- Add an operator to a plan so that it satisfies some open condition
- Add link (+ instantiate)
- Order (reorder) operators
Partial-order planners (POP)

- also called Non-linear planners
- Use STRIPS operators

Graphical representation of an operator Move(x,y,z)

<table>
<thead>
<tr>
<th>Preconditions</th>
<th>Add list</th>
</tr>
</thead>
<tbody>
<tr>
<td>On(x,y)</td>
<td>Clear(y)</td>
</tr>
<tr>
<td>Clear(x)</td>
<td>Clear(z)</td>
</tr>
</tbody>
</table>

Delete list is not shown !!!

Illustration of a POP on the Sussman’s anomaly case

Partial order planning. Start and finish.

Start

<table>
<thead>
<tr>
<th>Clear(C)</th>
<th>On(B, F1)</th>
<th>Clear(B)</th>
<th>On(A, F1)</th>
<th>Clear(F1)</th>
<th>On(A, B)</th>
</tr>
</thead>
</table>

Finish

Goal

A
B
C
Partial order planning. Start and finish.

**Open conditions:** conditions yet to be satisfied

---

Partial order planning. Add operator.

We want to satisfy **an open condition**

Always select an operator that helps to satisfy one of the open conditions
Partial order planning. Add link.

Start

On(C,A) Clear(Fl) On(A,Fl) Clear(B) On(B,Fl) Clear(C)

Move(A,y,B)

Clear(A) On(A,y) Clear(B)

On(A,B)

Clear(y) On(A,B)

Finish

On(A,B) On(B,C)

Goal

CS 1571 Intro to AI

M. Hauskrecht

Partial order planning. Add link.

Start

On(C,A) Clear(Fl) On(A,Fl) Clear(B) On(B,Fl) Clear(C)

Move(A,y,B)

Clear(A) On(A,y) Clear(B)

On(A,B)

Clear(y) On(A,B)

Finish

On(A,B) On(B,C)

Goal

Add link

Satisfies an open condition

CS 1571 Intro to AI

M. Hauskrecht
Partial order planning. Add link.

Satisfies an open condition
instantiates $y/Fl$

Partial order planning. Add operator.
Partial order planning. Add links.

Start

On(C,A)
Clear(Fl)

On(A,B)
Clear(Fl)

Clear(C)
On(A,Fl)

Move(A,Fl,B)
Clear(A)

Clear(Fl)
On(A,B)

Move(B,Fl,C)
Clear(B)

On(B,C)
Clear(Fl)

On(A,Fl)
Clear(B)

On(B,Fi)
Clear(C)

On(B,C)
Finish

Goal

Start

On(A,B)
Clear(Fl)

Clear(C)
On(A,Fl)

Move(A,Fl,B)
Clear(A)

Clear(Fl)
On(A,B)

Move(B,Fl,C)
Clear(B)

On(B,C)
Clear(Fl)

On(A,Fl)
Clear(B)

On(B,Fi)
Clear(C)

On(B,C)

Start

CS 1571 Intro to AI

M. Hauskrecht

Partial order planning. Interactions.

Start

On(C,A)
Clear(Fl)

On(A,B)
Clear(Fl)

Clear(C)
On(A,Fl)

Move(A,Fl,B)
Clear(A)

Clear(Fl)
On(A,B)

Move(B,Fl,C)
Clear(B)

On(B,C)
Clear(Fl)

On(A,Fl)
Clear(B)

On(B,Fi)
Clear(C)

On(B,C)

Start

CS 1571 Intro to AI

M. Hauskrecht

**Deletes Clear(B)**

**A is stacked on B**
Partial order planning. Order operators.

- Move(A,Fl,B)
  - On(A,Fl)
  - Clear(B)
  - Clear(Fl)

- Move(B,Fl,C)
  - On(B,Fl)
  - Clear(B)
  - Clear(C)

Move(B,Fl,C) comes before Move(A,Fl,B)

Partial order planning. Add operator

- Move(u,A,v)
  - On(u,A)
  - Clear(u)
  - Clear(v)

- Move(u,A,v)
  - On(u,A)
  - Clear(u)
  - Clear(v)
Partial order planning. Add links.

Partial order planning. Threats.
Partial order planning. Order operators.

- On(A,B) On(B,C)
- Start
- Clear(Fl)
- Clear(C)
- Move(A,Fl,B)
- On(A,Fl) Clear(A) Clear(B)
- On(A,B) Clear(Fl)
- Move(C,A,Fl)
- On(C,A) Clear(Fl) Clear(C)
- On(A,Fl) Clear(B)
- Move(B,Fl,C)
- On(B,Fl) Clear(B) Clear(C)
- On(B,C) Clear(Fl)
- Move(B,Fl)
- On(B,Fl)
- Finish
- Goal

Move(B,Fl,C) comes before Move(A,Fl,B)

POP planning. Directions.

- On(A,B) On(B,C)
- Start
- Clear(Fl)
- Clear(C)
- Move(A,Fl,B)
- On(A,Fl) Clear(A) Clear(B)
- On(A,B) Clear(Fl)
- Move(C,A,Fl)
- On(C,A) Clear(Fl) Clear(C)
- On(A,Fl) Clear(B)
- Move(B,Fl,C)
- On(B,Fl) Clear(B) Clear(C)
- On(B,C) Clear(Fl)
- Move(B,Fl)
- On(B,Fl)
- Finish
- Goal
Consistent POP plan.

Partial order planning. Result plan.

Plan: a topological sort of a graph
Partial order planning.

- **Remember** we search the space of partial plans

- POP: is sound and complete

Hierarchical planners

**Extension of STRIPS planners.**
- Example planner: ABSTRIPS.

**Idea:**
- Assign a **criticality level** to each conjunct in preconditions list of the operator
- Planning process refines the plan gradually based on criticality threshold, starting from the highest criticality value:
  - Develop the plan ignoring preconditions of criticality less than the criticality threshold value (assume that preconditions for lower criticality levels are true)
  - Lower the threshold value by one and repeat previous step
Towers of Hanoi

Start position

Goal position

Hierarchical planning

Assume:

- the largest disk – criticality level 2
- the medium disk – criticality level 1
- the smallest disk – criticality level 0
Planning with incomplete information

Some conditions relevant for planning can be:
- true, false or unknown

**Example:**
- Robot and the block is in Room 1
- **Goal:** get the block to Room 4
- **Problem:** The door between Room 1 and 4 can be closed

Initially we do not know whether the door is opened or closed:
- **Different plans:**
  - If not closed: pick the block, go to room 4, drop the block
  - If closed: pick the block, go to room 2, then room 3 then room 4 and drop the block
Conditional planners

- Are capable to create conditional plans that cover all possible situations (contingencies) – also called contingency planners
- Plan choices are applied when the missing information becomes available
- Missing information can be sought actively through actions
  - Sensing actions

Example:

**CheckDoor(d):** checks the door d

**Preconditions:** Door(d,x,y) – one way door between x and y

& At(Robot,x)

**Effect:** (Closed(d) v ¬Closed(d)) - one will become true
Conditional plans

Sensing actions and conditions incorporated within the plan:

- Pick(B) $\rightarrow$ CheckDoor(D) $\rightarrow$ Closed door?
- Go (R1,R4) $\rightarrow$ Drop(B)
- Go (R1,R2) $\rightarrow$ Go (R2,R3) $\rightarrow$ Go(R3,R4)

Room4 Room3
Room1 Room2

Room4 Room3
Room1 Room2