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 new states of the world
  - action objects model activities
  - uses inference methods developed for FOL to do the reasoning

Situation calculus

The language is based on the First-order logic plus:

- **Special variables**: $s, a$ – objects of type situation and action
- **Action functions**: return actions.
  - E.g. $Move(A, \text{TABLE}, B)$ represents a move action
  - $Move(x, y, z)$ represents an action schema
- **Two special function symbols of type situation**
  - $s_0$ – initial situation
  - $DO(a, s)$ – denotes the situation obtained after performing an action $a$ in situation $s$
- **Situation-dependent functions and relations** (also called *fluents*)
  - **Relation**: $On(x, y, s)$ – object $x$ is on object $y$ in situation $s$;
  - **Function**: $Above(x, s)$ – object that is above $x$ in situation $s$. 
Situation calculus. Blocks world example.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial state</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>On(A, Table, s₀)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>On(B, Table, s₀)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>On(C, Table, s₀)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clear(A, s₀)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clear(B, s₀)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clear(C, s₀)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clear(Table, s₀)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find a state (situation) s, such that</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>On(A, B, s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>On(B, C, s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>On(C, Table, s)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Knowledge base: Axioms.

Knowledge base 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
Blocks world example. Effect axioms.

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

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

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

Blocks world example. Frame axioms.

- **Frame axioms.**
  - Represent things that remain unchanged after an action.

  **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:**
  \( Clear(u, s) \land (u \neq z) \rightarrow Clear(u, DO(MOVE(x, y, z), s)) \)
Planning in situation calculus

Planning problem:
- find a sequence of actions that lead to a 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)

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

Action: MOVE (B,Table,C)

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

Initial state ($s_0$)

- $A$
- $B$
- $C$

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

$s_1 = DO(MOVE (B, Table, C), s_0)$

- $On(A, Table, s_1)$
- $Clear (A, s_1)$
- $Clear (Table, s_1)$
- $On(B, C, s_1)$
- $Clear (B, s_1)$
- $On(C, Table, s_1)$
- $Clear (C, s_1)$

$s_2 = DO(MOVE (A, Table, B), s_1)$

- $On(A, B, s_2)$
- $On(A, Table, s_2)$
- $Clear (B, s_2)$
- $Clear (A, s_2)$
- $Clear (Table, s_2)$

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.

Problems:

- 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 may be too cumbersome and inefficient to represent and solve the planning problems.

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

\[
On(u, v, DO(MOVE(x, y, z), s)) \iff (\neg((u = x) \land (v = y)) \land On(u, v, s)) \lor \\
(\neg((u = x) \land (v = z)) \land On(x, y, s) \land Clear(x, s) \land Clear(z, s))
\]

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
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

• 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 as 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:** \( \text{Move}(x,y,z) \)

• **Preconditions:** \( \text{On}(x,y), \text{Clear}(x), \text{Clear}(z) \)

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

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

---

[Diagram showing the effects of moving an object from one location to another, with states and actions represented.]
STRIPS planning

**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:
\[ On(A,B) \land On(B,C) \]

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 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

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

\[
\begin{array}{c|c}
\text{Clear}(A) & \text{Clear}(B) \\
\text{Clear}(Table) & \text{Clear}(Table) \\
\end{array}
\]

\[
\text{Move}(B, Table, C)
\]

\[
\text{Add: On}(B, C) \\
\text{Delete: On}(B, Table)
\]

\[
\text{Move}(A, Table, B)
\]

Forward search (goal progression)

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

**Search tree:**

- Initial state: \( A \ B \ C \)
- Goal: \( A \ B \ C \)
**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  B  C
```

- `Move (A, Table, B)`
- `Move (B, Table, C)`
- `Move (A, Table, C)`
- `Move (A, Table, B)`

Goal

```
A  B  C
```

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 $a$

New goal ($G'$)

```
On (A, Table)
Clear (B)
Clear (A)
On (B, C)
On (C, Table)
```

Goal ($G$)

```
On (A, B)
On (B, C)
On (C, Table)
```

precondition
add

Mapped 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)
```
Sussman’s anomaly

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

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

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

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

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

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