Planning II

STRIPS framework

• Defines a restricted version of the FOL 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 (planning) 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 of STRIPS planning](attachment:image.png)
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:
\[\text{On}(A,B) \land \text{On}(B,C)\]

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 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 (can be repeated)

```
| A | B | C |
```

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 (can be repeated)

```
| A | B | C |
```
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 (can be repeated)
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)
```

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 $a$

```
Goal (G)

precondition add
On (A, B)
On (B, C)
On (C, Table)

Mapped from G
```

```
A B C
```

```
A
B
C
```

```
A
B
C
```
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 $a$

\[
\begin{array}{c|c|c}
\text{Goal (G)} & \text{New goal (G')} \\
\hline
\text{On}(A,B) & \text{On}(B,C) \\
\text{On}(B,C) & \text{On}(C,\text{Table}) \\
\end{array}
\]
Backward search (goal regression)

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

Search tree:

![Search Tree Diagram](https://example.com/search_tree.png)

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 divide and conquer fails due to interacting goals
1. Assume we want to satisfy $On(A, B)$ first

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

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

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
- State (set of formulas)
- Transition from $s_0$ to $s_1$ to $s_2$

**Plan-space search**

- Start
- Incomplete (partial) plan
- Plan transformation operators
- Transition to Finish

Diagram:

```
State-space search

STRIPS operator

State
(set of formulas)

Transition from $s_0$ to $s_1$ to $s_2$

Plan-space search

Start

Incomplete (partial) plan

Plan transformation operators

Transition to Finish
```
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 $\text{Move}(x,y,z)$

Delete list is not shown !!!

Illustration of a POP on the Sussman’s anomaly case
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.
Partial order planning. Add link.

Finish

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

Goal

Start

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

Finish

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

Goal

Start

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

Add link

Satisfies an open condition

Add link

Satisfies an open condition

instantiates y/FL
Partial order planning. Add operator.

Partial order planning. Add links.
Partial order planning. Interactions.

A
B
C

Finish
On(A,B)
Clear(Fl)
On(A,B)
Move(A,Fl,B)
Clear(A)
On(A,F1)
Clear(C)
On(C,A)
Clear(Fl)
On(B,C)
Clear(Fl)
On(B,C)
Move(B,Fl,C)
Clear(B)
On(B,Fl)
Clear(B)
On(B,F1)
Clear(C)
On(C,A)
Clear(Fl)
On(C,A)
Clear(C)
On(A,Fl)
Clear(B)
On(A,B)
Clear(Fl)
On(A,B)
Clear(Fl)
On(A,B)

Start

Goal
Delete Clear(B)
A was stacked on B

Partial order planning. Order operators.

A
B
C

Finish
On(A,B)
Clear(Fl)
On(A,B)
Move(A,Fl,B)
Clear(A)
On(A,F1)
Clear(C)
On(C,A)
Clear(Fl)
On(B,C)
Clear(Fl)
On(B,C)
Move(B,Fl,C)
Clear(B)
On(B,Fl)
Clear(B)
On(B,F1)
Clear(C)
On(C,A)
Clear(Fl)
On(C,A)
Clear(C)
On(A,Fl)
Clear(B)
On(A,B)
Clear(Fl)
On(A,B)
Clear(Fl)
On(A,B)

Start

Goal
Move(B,F1,C)
comes before
Move(A,F1,B)
Partial order planning. Add operator

Partial order planning. Add links.
Partial order planning. Threats.

On(A,B)

Clear(Fl)

Move(A,Fl,B)

On(A,Fl)

Clear( Fl)

Move(C,A,Fl)

On(C,A)

Clear(Fl)

Clear(C)

On(C,Fl)

C

Start

Finish

On(B,C)

Clear( Fl)

Move(B,Fl,C)

On(B,Fl)

Clear(C)

On(B,C)

Clear(Fl)

Delete Clear(C)

B moved on top of C

A

B

C

Start

Goal

On(A,Fl) Clear(B)

On(B,Fl)

C

B

A

Goal

Partial order planning. Order operators.

On(A,B)

Clear(Fl)

Move(A,Fl,B)

On(A,Fl)

Clear( Fl)

Move(C,A,Fl)

On(C,A)

Clear(Fl)

Clear(C)

On(C,Fl)

C

Start

Finish

On(B,C)

Clear( Fl)

Move(B,Fl,C)

On(B,Fl)

Clear(C)

On(B,C)

Clear(Fl)

Delete Clear(C)

B moved on top of C

A

B

C

Start

Goal
POP planning. Directions.

Consistent POP plan.
Partial order planning. Result plan.

Plan: a topological sort of a graph

Start
Move(C, A, Fl)
Move(A, Fl, B)
Move(B, Fl, C)
Finish

Partial order planning.

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

  Start
  Incomplete (partial) plan
  Finish

- 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
Planning with incomplete information

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**
### Sensing actions

**Example:**

- **CheckDoor(d):** checks the door d
- **Preconditions:** $\text{Door}(d,x,y)$ – one way door between x and y
  - & $\text{At}(\text{Robot},x)$
- **Effect:** $(\text{Closed}(d) \lor \neg \text{Closed}(d))$ - one will become true

### Conditional plans

Sensing actions and conditions incorporated within the plan:

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