Planning: STRIPS

STRIPS - Essence

A state in STRIPS consists of a set of formulae describing the current world state.
Planning is search through the space of world states, using actions as operators to generate the search space.
Actions connect ‘before’ and ‘after’ world states
Before and after states are described using only ground literals (conjunction thereof) and some general axioms.

STRIPS - Actions 1

- Actions are described by preconditions and effects (conjunctions of literals).
- Effects are split into an ADD-list and a DELETE-list:
  - the ADD-list contains every new formula to be added to the current state as result of the action
  - the DELETE-list contains all formulae to be deleted from the current state as result of the action

Note: ADD- and DELETE-lists explicitly specify what becomes true and what becomes false after an action. Thus, the state is carried along, just with necessary changes made. This circumvents the Frame Problem.
Actions in STRIPS are described as schemata with variables which are instantiated during planning (→ unification).

**Example:**

\[ \text{move (x, y, z)} \]

- **precondition:** \( \text{on (x, y)} \land \text{clear (x)} \land \text{clear (z)} \)
- **delete-list:** \( \text{clear (z)}, \text{on (x, y)} \)
- **add-list:** \( \text{on (x, z)}, \text{clear (y)}, \text{clear (Table)} \)

*in case z=Table*

---

Actions are described by **preconditions** and **effects** (conjunctions of positive ground literals).

**Effects** are split into an **ADD-list** and a **DELETE-list**.

An action can be applied in a state, if its **preconditions** are satisfied in that state.

The **resulting new state** is determined through the effect, described as **ADD-** and **DELETE-lists**.

**Planning** find action-operator which achieves goal-state

**initial world state:** \( \text{on (A, B)} \lor \text{clear (A)} \lor \text{clear (C)} \)

**goal state:** \( \text{on (A, C)} \)

---

---

**Example:**

\[ \text{move (x, z)} \]

- **precondition:** \( \text{on (x, y)} \land \text{clear (x)} \land \text{clear (z)} \)
- **delete-list:** \( \text{clear (z)}, \text{on (x, y)} \)
- **add-list:** \( \text{on (x, z)}, \text{clear (y)}, \text{clear (Table)} \)

*in case z=Table*
STRIPS – Example

move (x,y,z): has on(x,z) in delete-list
 instantiate with x=A, z=C
 precondition: on (A,y) \land clear (A) \land clear (C)
 delete-list: clear (C), on (A, y)
 add-list: on (A,C), clear (y), clear (Table)

Planning
match action-description with current world state:
initial world state: on (A,B) \land clear (A) \land clear (C)
 instantiate y=B; done
 goal state: on (A,C)

STRIPS – Planning as Search

Planning is similar to theorem proving in Prolog.
Plan = sequence of action-operators
defines a path from a start-state to a goal-state.
Planning can be performed
• forward (from start state to goal state)
  \rightarrow progression planning
• backward (from goal to start)
  \rightarrow regression planning
Planning recursively - for each achieved sub-goal

STRIPS – Planning as Search

Forward planning:
Start with the initial state (current world state).
Select actions whose preconditions match with the current state description (before-action state) through unification.
Apply actions to the current world state. Generate possible follow-states (after-action states) according to the add and delete lists of the action description.
Repeat for every generated new world state.
Stop when a state is generated which includes the goal formula.

means-ends analysis
Reduce the difference between the start state and the goal state
Choose action-operators which generate literals of the goal-state description. Recursively generate action sequence (=plan) by trying to fulfill preconditions of chosen actions, until they are finally grounded in the start-state, i.e. match with literals of the start-state.
### STRIPS – Planning as Search

**Backward planning:**
- **Start** with a given **goal state description**.
- **Check unsatisfied (pre)conditions** of the goal description.
- **Select and apply actions** which have those conditions as effects.
- **Proceed** in the same way **backwards**, trying to fulfill preconditions of each new action by **recursively choosing actions** which achieve those preconditions.
- **Stop** when the **precondition is part of the initial state**, and thus a sequence of actions is found leading from the initial state to the goal state.

### STRIPS – Problems

**Sussman Anomaly**
In the process of achieving a new sub-goal during the planning process, STRIPS might reverse an effect it had already achieved.

**Solution Approach:**
Don't do that! Perform backward search. Just look for conditions to achieve, and do not delete anything which is in an achieved goal or sub-goal (goals are conjunctions of literals; deleting a part of this, would make the complete goal formula false).

### Planning - Problems

**Frame-Problem**
- specify everything which remains stable computationally very expensive
  - Approach: successor-state axioms; STRIPS

**Qualification-Problem**
- specify all preconditions to an action
difficult to include every possible precondition
  - Approach: non-monotonic reasoning with defaults

**Ramification-Problem**
- considering side-effects of actions
  - conflict between effects and frames
  - Approach: integrate TMS

### Resource Constraints in Planning

- **Resources**
  - physical quantities, e.g. money, fluids etc.
  - time
  - **Integrate Measures** into Action Description and Planning
    - representation of physical quantities and reasoning / calculation, e.g. for buy-action: effect: cash := cash – price (x)
    - time system / time logic, e.g. go-to-action: effect: time := time + 30 (Minutes)
- **Backtracking on Constraint Violation**
**ADL - Action Definition Language**

**ADL**
Can be seen as extension of the STRIPS language.
Contains typing of parameters (sorts).
Allows explicit expression of negation.
Allows equality of terms in precondition formula.

**Example:**
Fly \( \left( p: \text{plane}; \text{from: airport}; \text{to: airport}; c: \text{cargo} \right) \)
precondition: \( \text{at}(p, \text{to}) \land \text{at}(c, \text{to}) \land \text{in}(c, p) \land \text{to} \neq \text{from} \)
effect: \( \text{at}(p, \text{to}) \land \text{at}(c, \text{to}) \land \neg \text{at}(p, \text{from}) \land \neg \text{at}(c, \text{from}) \)

---

**STRIPS – Plan Schemata**

- Concrete plans can also be seen as instantiations of **Plan Schemata**.
- Shakey was able to generalize generated concrete plans into such plan schemata.
  \( \rightarrow \text{plan generalization, learning} \)

---

**Plan Learning / Plan Abstraction**

Shakey generated plan schemata, so-called **MACROPs** (Macro-Operators), from concrete plans it had constructed earlier (i.e. “it learns plans”).

The learning process is based on describing the plan in a **triangle-table** (preconditions and effects of sequential actions in rows, actions in columns). Then substitute **constants with variables**, in a kind of inverse variable binding process with unification. (substitute the same constant with one variable)

Thus, an **abstract plan schema** is generated from the concrete plan.

---

**ABSTRIPS - Triangle Table**

- **Generate Plan with STRIPS**
- **Set up Triangle Table:**
  
  - left of action: precondition
  
  - below action: effects (add-list) - record only literals needed by subsequent actions or as part of the goal clause.

- **Generalization (abstraction)**
  
  - Substitute constants with variables.
Generate Plan Schema
- Generalization (abstraction) of concrete plan.
- Substitute constants with variables:
  - $F_i$ remains $F_i$
  - $A \rightarrow x$
  - $B \rightarrow y$
  - $C \rightarrow z$

Planning – Other Approaches

Partial Order Planning
Start with rough plan, and use refinement operators to work out details.

Hierarchical Task Networks / Plan Decomposition
Plan schemata are organized in a hierarchy. Links between nodes at different levels in this hierarchy denote a decomposition of a complex plan/action.

Situation Calculus
Planning as Theorem Proving. Describe situations (world states) using a situation parameter in action descriptions.
Additional References