Deductive Planning

Deductive inference can be used to solve planning problems.

Introduce a situation variable to store the partial plans:

\[ s_{i+1} = \text{put}(A, B, s_i), \ldots \]
\[ s_2 = \text{puttable}(A, s_1) \]
\[ s = \text{put}(A, B, \text{puttable}(A, \text{on}(A, C), \text{clear}(A), \ldots)) \]

Situation calculus: Introduced by McCarthy (1963) and used for plan construction by resolution by Green (1969)

In general: extensions of FOL (action languages)

Proof logically, that a set of goals follows from an initial state given operator definitions (axioms)

Perform the proof in a constructive way (plan is constructed as a byproduct of the proof)
Situation Calculus

A1  \textit{on}(a, \textit{table}, s_1) \text{ (literal of the initial state)}

A2  \text{(axiom for put-operator)}

\[ \forall S [\text{on}(a, \text{table}, S) \rightarrow \text{on}(a, b, \text{put}(a, b, S))] \equiv \neg \text{on}(a, \text{table}, S) \lor \text{on}(a, b, \text{put}(a, b, S)) \text{ (clausal form)} \]

Proof the goal predicate \textit{on}(a, b, S_F)

1. \neg \text{on}(a, b, S_F) \text{ (Negation of the theorem)}

2. \neg \text{on}(a, \text{table}, S) \lor \text{on}(a, b, \text{put}(a, b, S)) \text{ (A2)}

3. \neg \text{on}(a, \text{table}, S) \text{ (Resolve 1, 2) } \leftrightarrow \text{answer(put}(a, b, S))

4. \text{on}(a, \text{table}, s_1) \text{ (A1)}

5. \text{contradiction} \text{ (Resolve 3, 4) } \leftrightarrow \text{answer(put}(a, b, s_1))

\(s_2 = \text{on}(a, \text{table}, s_1)\) with \textit{on}(a, b, s_2) exists and \(s_2\) can be reached by putting \(a\) on \(b\) in situation \(s_1\).
Frame Problem Revisited

- No closed world assumption $\iff$ full expressive power of FOL

- Problem: additionally to axioms describing the effects of actions, frame axioms become necessary

- Frame axioms are necessary to allow proofing conjunctions of goal literals.

- Example for a frame axiom:

  $\forall S [on(Y, Z, S) \rightarrow on(Y, Z, put(X, Y, S))]$

  $on(Y, Z, put(X, Y, S)) \leftarrow on(Y, Z, S)$

  After a block $X$ was put on a block $Y$, it still holds that $Y$ is lying on a block $Z$, if this did hold before the action was performed.
Blocksworld in Prolog

Effect Axioms:

\[
\begin{align*}
on(X, Y, \text{put}(X, Y, S)) & \leftarrow \quad \text{clear}(X, S) \land \text{clear}(Y, S) \\
clear(Z, \text{put}(X, Y, S)) & \leftarrow \quad \text{on}(X, Z, S) \land \text{clear}(X, S) \land \text{clear}(Y, S) \\
clear(Y, \text{puttable}(X, S)) & \leftarrow \quad \text{on}(X, Y, S) \land \text{clear}(X, S) \\
\text{ontable}(X, \text{puttable}(X, S)) & \leftarrow \quad \text{clear}(X, S)
\end{align*}
\]

Frame Axioms:

\[
\begin{align*}
clear(X, \text{put}(X, Y, S)) & \leftarrow \quad \text{clear}(X, S) \land \text{clear}(Y, S) \\
clear(Z, \text{put}(X, Y, S)) & \leftarrow \quad \text{clear}(X,S) \land \text{clear}(Y, S) \land \text{clear}(Z, S) \\
\text{ontable}(Y, \text{put}(X, Y, S)) & \leftarrow \quad \text{clear}(X, S) \land \text{clear}(Y, S) \land \text{ontable}(Y, S) \\
\text{ontable}(Z, \text{put}(X, Y, S)) & \leftarrow \quad \text{clear}(X, S) \land \text{clear}(Y, S) \land \text{ontable}(Z, S) \\
on(Y, Z, \text{put}(X, Y, S)) & \leftarrow \quad \text{clear}(X, S) \land \text{clear}(Y, S) \land \text{on}(Y, Z, S) \\
on(W, Z, \text{put}(X, Y, S)) & \leftarrow \quad \text{clear}(X, S) \land \text{clear}(Y, S) \land \text{on}(W, Z, S)
\end{align*}
\]
Frame Axioms cont.:

\[
\begin{align*}
\text{clear}(Z, \text{puttable}(X, S)) & \leftarrow \text{clear}(X, S) \land \text{clear}(Z, S) \\
\text{ontable}(Z, \text{puttable}(X, S)) & \leftarrow \text{clear}(X, S) \land \text{ontable}(Z, S) \\
\text{on}(Y, Z, \text{puttable}(X, S)) & \leftarrow \text{clear}(X, S) \land \text{on}(Y, Z, S) \\
\text{clear}(Z, \text{puttable}(X, S)) & \leftarrow \text{on}(Y, X, S) \land \text{clear}(Y, S) \land \text{clear}(Z, S) \\
\text{ontable}(Z, \text{puttable}(X, S)) & \leftarrow \text{on}(Y, X, S) \land \text{clear}(Y, S) \land \text{ontable}(Z, S) \\
\text{on}(W, Z, \text{puttable}(X, S)) & \leftarrow \text{on}(Y, X, S) \land \text{clear}(Y, S) \land \text{on}(W, Z, S)
\end{align*}
\]

Facts (Initial State):

\[
\begin{align*}
\text{on}(d, c, s_1) \\
\text{clear}(d, s_1) \\
\text{ontable}(a, s_1) \\
\text{on}(c, a, s_1) \\
\text{clear}(b, s_1) \\
\text{ontable}(b, s_1)
\end{align*}
\]

Theorem (Goal):

\[
\text{on}(a, b, S) \land \text{on}(b, c, S)
\]
AI Planning: Current Approaches

Domain-independent planning in deterministic domains

- 70ies: STRIPS and deductive planning
- 80ies: partial order planning (e.g. UCPOP)
- 90ies: extensions of STRIPS language (PDDL) and new, efficient algorithms – Graphplan, SATPlan
Graphplan: Basic Ideas

- Make search for a plan more efficient by first constructing a planning graph from which the valid plan can be extracted

- A planning graph is a directed, levelled graph, that is, nodes can be partitioned into disjoint sets $L_1, L_2, \ldots, L_n$ such that the edges connect only nodes in adjacent levels

- Two kinds of nodes
  - Starting with level 0, nodes at even levels represent propositions true at time $t_i$
  - Nodes at odd levels represent possible actions at time $t$
Four kinds of edges

- Precondition edges: from propositions to actions
- Add edges: from actions to propositions
- Del edges: from actions to propositions
- No-op edges: from propositions to propositions
Rocket Example

move(\texttt{?r} \texttt{?f} \texttt{?t})
PRE: (at \texttt{?r} \texttt{?f}), (\texttt{?f} \neq \texttt{?t}), (has-fuel \texttt{?r})
ADD: (at \texttt{?r} \texttt{?t})
DEL: (at \texttt{?r} \texttt{?f}), (has-fuel \texttt{?r})

unload(\texttt{?r} \texttt{?p} \texttt{?c})
PRE: (cargo \texttt{?c}) (at \texttt{?r} \texttt{?p}) (in \texttt{?c} \texttt{?r})
ADD: (at \texttt{?c} \texttt{?p})
DEL: (in \texttt{?c} \texttt{?p})

load(\texttt{?r} \texttt{?p} \texttt{?c})
PRE: (cargo \texttt{?c}) (at \texttt{?r} \texttt{?p}) (at \texttt{?c} \texttt{?p})
ADD: (in \texttt{?c} \texttt{?r})
DEL: (at \texttt{?c} \texttt{?p})

Example Problem:
Initial State: \{(at \texttt{R} \texttt{L}), (at \texttt{A} \texttt{L}), (at \texttt{B} \texttt{L}), (has-fuel \texttt{R}), (cargo \texttt{A}), (cargo \texttt{B})\} 
Goal: \{(at \texttt{A} \texttt{P}), (at \texttt{B} \texttt{P})\}
Construct Planning Graph

- Start with initial state
- Introduce all propositions as nodes on level $S_0$

Construct the next levels:

- Find all actions whose preconditions are contained in level $S_i$ and introduce them as nodes in level $A_i$
- Introduce all propositions of the ADD and DEL list in level $S_{i+1}$
- Copy all propositions from level $S_i$ to level $S_{i+1}$
Rocket Planning Graph
Remarks on Planning Graphs

- can be constructed in polynomial time since planning is NP-hard (PSPACE complete), plan extraction is of exponential effort!

- based on propositional logic may be hard to transform a problem given in PDDL into propositional form! (all possible instantiation of actions, which instances are legal?)

- represent a partially-ordered plan (independent actions are given on the same level and can be performed in arbitrary order) actions are independent, if one does not delete a precondition or an add-effect of the other

- to facilitate plan extraction, mutual exclusive nodes can be identified
Mutex Relations

Two actions at one level are mutex, if
- either one deletes a precondition or add-effect of the other (interference)
- their preconditions are mutex

Two propositions are mutex, if
actions adding them are mutex.

Mutex Relations of the Rocket Problem:
- (move L P R) and (load A L R) are mutex because the precondition for “load” is deleted by “move”

Mutex Relations are a heuristic:
- Not all mutex relations can be found by this rules
- More complex mutex relations (between triples etc.) are not checked (too expensive)
Extraction of a Valid Plan

by backwards-search (goal regression)

- Level-by-level to exploit the mutex constraints
- Given a set of goals at level $i$, find a set of actions (including no-ops) at the preceding level which have these goals as add-effects
- The preconditions of these actions form the subgoals for the next regression step
- if a goal set at a level is unsolvable, backtrack and select different actions
- continue until success or prove that the original goals are not solvable at this level ($\rightarrow$ expand planning graph to a next level containing all goals)
Termination

Termination on unsolvable problems is tricky, but the Graphplan algorithm is sound and complete for STRIPS-style operators where only a finite set of propositions can be generated.
Planning in Real-World Domains

- **Incomplete Information**
  - Conformant planning: Create plans that work for all cases
  - Conditional planning: sense world during execution and decide which branch of the plan to follow

- **Incorrect Information**
  - Execution monitoring: check for unsatisfied preconditions
  - Re-planning

- Continuous planning: create new goals during acting in real time

- Multiagent planning
Including Knowledge

Using knowledge about the structure of the domain

- Hierarchical Planning (decomposition rules) cf. problem solving with AND-OR trees
- Domain axioms
- Domain specific search strategies

larger plans become feasible (necessary for many real world problems, e.g. Mars Mobile)

Alternative to knowledge engineering: Learning of planning strategies!
Further Topics

- Interleaving plan construction and plan execution
- Plan revision
- Planning with temporal/resource constraints
- Non-deterministic planning
- ...

CogSysI Lecture 7: Different Approaches To AI Planning – p. 216