CogSysI Lecture 6: Basic AI Planning

Intelligent Agents

WS 2006/2007

Part II: Problem Solving and Planning

II.6 Basic AI Planning
AI Planning

AI planning deals with the formalization, implementation and evaluation of algorithms for construction plans.

Plan: sequence of actions for transforming a given state into a state which fulfills a predefined set of goals.

Basic approach to state based planning: Strips

Basic approach to deductive planning: Situation Calculus

Alternative to planning: reinforcement learning
Strips


a linear (and therefore incomplete) approach

relies on the closed-world assumption (CWA)

today: extensions of the Strips language and non-linear algorithms

classical examples: moving boxes between rooms (“Strips World”), blocksworld
Components of a Planning Formalism

- A language to represent states, goals, operators
- Typically, different subsets of FOL for states, goals and operators.
  States as conjunction of positive ground literals, goals might be more complex (allowing quantified variables, disjunction, etc.)
- In contrast to problem solving, operators are typically represented as schemes
- Operator semantics: achieving a state change by applying an action (state space model)

- An algorithm for constructing a plan (a search algorithm; in contrast to problem solving, typically backwards).

- Crucial difference to problem solving: no domain specific knowledge (heuristics)
Strips States

State description as conjunction of positive ground literals, with domain objects (constants)

\[ D = \{A, B, C, D\} \]

and predicate symbols

\[ P = \{\text{ontable}^1, \text{clear}^1, \text{on}^2\} \]

\{ontable(A), ontable(C), ontable(D), clear(A), clear(B), clear(D), on(B, C)\}

Only conjunction \(\rightarrow\) write set of literals.

CWA: All relations not given explicitly are assumed to be false.

A state description corresponds to a state in a state-space model.
Typically, only “relevant” aspects of a state are represented.
Strips Goals

- Goals: typically partial state descriptions, e.g., $G = \{\text{on}(B,C), \text{ontable}(C)\}$
- A state $s$ is called goal state iff $G \subseteq s$
Strips Operator Schemes

A Strips operator is described by **precondition** (PRE) and effect.

Effects are represented as **ADD** and **DEL** lists (more precisely **sets**)

In the most simple case, PRE, ADD, DEL are conjunctions of literals.
Variables are assumed to be existentially quantified, literals in the precondition are positive.

**Operator:** put(?x, ?y)
**PRE:** {ontable(?x), clear(?x), clear(?y)}
**ADD:** {on(?x, ?y)}
**DEL:** {ontable(?x), clear(?y)}
Operator Application

- The precondition of an operator is matched against the current state.

- Convention: Variables with different names are instantiated with different objects.

- If $\text{PRE}_\sigma \subseteq s$, the preconditions “hold” and the operator can be applied.

- Substitution $\sigma$ is performed over the complete operator scheme, that is, the variables occurring in ADD and DEL are instantiated accordingly.

- Complication: free variables in ADD/DEL (instantiate again by matching with the state)

- Strips is propositional (no terms), that is, variables are always replaced by constant symbols.

- Instantiated operators are called actions.
Instantiated Operator

Operator: \( \text{put}(A, B) \)
PRE: \{ontable}(A), \text{clear}(A), \text{clear}(B)\}
ADD: \{on}(A, B)\}
DEL: \{ontable}(A), \text{clear}(B)\}

\[ \sigma = \{x \leftarrow A, y \leftarrow B\} \]
Operator Application cont.

With $o$ we denote a fully instantiated operator, with $\text{PRE}(o)$, $\text{ADD}(o)$, $\text{DEL}(o)$, the according components of $o$.

Operator application is defined as:

$$\text{Res}(o, s) = (s \setminus \text{DEL}(o)) \cup \text{ADD}(o); \text{ if } \text{PRE}(o) \subseteq s$$

Predicates which only occur in $\text{PRE}$, but never in $\text{ADD}/\text{DEL}$ are called **statics**, they describe “constraining attributes”

Set-theoretical definition of operator application: allows for syntactic calculation of operator effects.
State-Space Model

Arc from state $s_i$ to $s_j$ iff $s_j$ can be reached from $s_i$ by performing a single action.
Operator Application Example

\[
\{\text{ontable}(A), \text{clear}(A), \text{clear}(B)\} \subseteq \{\text{ontable}(A), \text{ontable}(C), \text{ontable}(D), \text{clear}(A), \text{clear}(B), \text{clear}(D), \text{on}(B, C)\} \\
\implies \text{applicable}
\]

\[
\{\text{ontable}(A), \text{ontable}(C), \text{ontable}(D), \text{clear}(A), \text{clear}(B), \text{clear}(D), \text{on}(B, C)\} \\
\setminus \{\text{ontable}(A), \text{clear}(B)\} \\
= \{\text{ontable}(C), \text{ontable}(D), \text{clear}(A), \text{clear}(D), \text{on}(B, C)\}
\]

\[
\{\text{ontable}(C), \text{ontable}(D), \text{clear}(D), \text{on}(B, C)\} \cup \{\text{on}(A, B)\} \\
= \{\text{ontable}(C), \text{ontable}(D), \text{clear}(A), \text{clear}(D), \text{on}(A, B), \text{on}(B, C)\}
\]

\[
s_{i+1} = \text{Res}(o, s_i)
\]

CogSysI Lecture 6: Basic AI Planning – p. 182
Planning Domain and Problem

**Planning Domain**: Operator Schemes (for extended formalisms additionally types, axioms, functions, ...)  
General description of a domain, such as blocksworld.

**Planning Problem**: a domain, an initial state and a planning goal  
The problem, not the domain, constitutes a set of domain objects!

Standardized, extended Strips language for state-based planners: **PDDL** (Planning Domain Definition Language), see, e.g.,  
http://www.dur.ac.uk/d.p.long/competition.html

PDDL has a Lisp-based syntax. Classically state-based planning is realized in Lisp. Today most planners are in C.
Blocksworld Domain

Operators:

**put(?x, ?y)**

**PRE:**  \{ontable(?x), clear(?x), clear(?y)\}

**ADD:**  \{on(?x, ?y)\}

**DEL:**  \{ontable(?x), clear(?y)\}

**puttable(?x)**

**PRE:**  \{clear(?x), on(?x, ?y)\}

**ADD:**  \{ontable(?x), clear(?y)\}

**DEL:**  \{on(?x, ?y)\}
Blocksworld Problem

Blocksworld Domain +:

Goal: \{on(A, B), on(B, C)\}

Initial State: \{on(D, C), on(C, A), clear(D),
clear(B), ontable(A), ontable(B)\}
PDDL

Equality constraints and conditioned effects

(define (domain blocksworld-adl)
 (:requirements :strips :equality :conditional-effects)
 (:predicates (on ?x ?y)
   (clear ?x)) ; clear(Table) is static
 (:action puton
  :parameters (?x ?y ?z)
  :precondition (and (on ?x ?z) (clear ?x) (clear ?y)
    (not (= ?y ?z)) (not (= ?x ?z))
    (not (= ?x ?y)) (not (= ?x Table)))
  :effect
    (and (on ?x ?y) (not (on ?x ?z))
      (when (not (eq ?z Table)) (clear ?z))
      (when (not (eq ?y Table)) (not (clear ?y))))))
)
Planning Algorithms

- Typically: variants of depth-first search
- Planning domains are usually too complex for applying breadth-first search
- Breadth-first based strategies: used in universal/conformant planning, based on techniques from symbolic model-checking

- **Forward planning:** Start with the initial state, built a search tree by transforming a given state by action application, check for cycles, backtrack if you reach a dead-end, terminate if a goal state is reached or if all paths are generated (for finite domains)

- **Backward planning:** Start with the top-level goals (a partial description of the goal state), build a search tree by transforming a given state by *backwards* action application, ..., terminate if the a state is reached which subsumes the initial state or ...
Backward Planning

backward operator application:

\[ \text{Res}^{-1}(o, s) = (s \setminus \text{ADD}(o)) \cup \text{PRE}(o); \text{ if } \text{ADD}(o) \subseteq s \]

- Also called regression planning
- Advantage: typically smaller search trees (cf. discussion in problem solving)
- Problem: inconsistent states can be produced can e.g. be detected by including axioms (domain knowledge!)
- Graphplan strategy: build a Planning Graph by forwards search (polynomial effort) and extract the plan from the graph backwards (exponential effort, as usual for planning)
Backward Planning cont.

inconsistent:

- to fulfill clear(y), y = B
  puttable(x)
- deletes on(x, B)

will become inconsistent for z = B

CogSysI Lecture 6: Basic AI Planning – p. 189
Evaluating Planners

- **Termination** (critical case: no solution exists)

- **Soundness**: every plan returned is a legal sequence of actions to achieve the goal
  implies consistency: each intermediate state appearing in the plan is a legal state of the domain

- **Completeness**: the planner finds a solution, if one exists.

- **Optimality**: the returned plans are optimal (shortest) solutions (typically not considered)

- **Expressiveness of the planning language**

- **Efficiency**

- **AIPS planning competition**: every two years current systems compete on hard benchmark problems (logistics, freecell, lift-control, ...)
Planning Approaches

- **Total vs. partial order** planners: returned plan is a totally ordered sequence of actions vs. some independent actions are given in parallel; *Graphplan* planners are partial order.

- **Linear vs. non-linear** planners: linear planners consider on goal at a time and work with a goal-stack, non-linear planners allow *interleaving of goals* (see below); all modern planners are non-linear.
Planning Approaches cont.

- **State-space vs. plan-space** planners: the first partial order planners (NOAH by Sacerdoti, UCPOP by Weld) were plan-space planners, they work with a least commitment strategy
  Plan-space: search in the space of partial plans, start with a plan which only contains initial state and top-level goals

- **Hierarchical** planning (see AND-OR Trees in problem solving); not to confuse with partial order planning!

- A successful forward-planning system, which estimates heuristic values for the distance of a state from the goal from the problem definition is **HSP**.
Incompleteness of Linear P.

The Sussman Anomaly

Initial State

Goal: on(A, B) and on(B, C)

on(B, C) on(A, B)
Linear planning corresponds to dealing with goals organized in a stack:

\[\text{on}(A, B), \text{on}(B, C)\]

try to satisfy goal \text{on}(A, B)

solve sub-goals \[\text{clear}(A), \text{clear}(B)\]

all sub-goals hold after \text{puttable}(C)

apply \text{put}(A, B)

goal \text{on}(A, B) is reached

try to satisfy goal \text{on}(B, C).

\(^{a}\)We ignore the additional subgoal \text{ontable}(A) \text{rsp. on}(A, z) \text{here.}
Interleaving of Goals

Non-linear planning allows that a sequence of planning steps dealing with one goal is interrupted to deal with another goal.

For the Sussman Anomaly, that means that after block $C$ is put on the table pursuing goal $on(A, B)$, the planner switches to the goal $on(B, C)$.

Non-linear planning corresponds to dealing with goals organized in a set.

The correct sequence of goals might not be found immediately but involve backtracking.
Interleaving of Goals cont.

\{on(A, B), on(B, C)\}
try to satisfy goal \(on(A, B)\)
\{clear(A), clear(B), on(A, B), on(B, C)\}
clear(A) and clear(B) hold after \(puttable(C)\)

try to satisfy goal \(on(B, C)\)
apply \(put(B, C)\)

try to satisfy goal \(on(A, B)\)
apply \(put(A, B)\).
Rocket Domain

(Veloso)

Objects: \( n \) boxes, Positions (Earth, Moon), one Rocket

Operators: load/load a box, move the Rocket (oneway: only from earth to moon, no way back!)

Linear planning: to reach the goal, that Box1 is on the Moon, load it, shoot the Rocket, unload is, now no other Box can be transported!
The Running Gag of CogSysI

Question: How many AI people does it take to change a lightbulb?  
Answer: At least 67.

6th part of the solution: The Planning Group (4)

- One to define STRIPS-style operators for lightbulb changing
- One to show that linear planning is not adequate
- One to show that nonlinear planning is adequate
- One to show that people don’t plan; they simply react to lightbulbs

(“Artificial Intelligence”, Rich & Knight)