Where are we?

The first two blocks of the course dealt with . . .

- Basic notions of agency
- Intelligent problem-solving
- Heuristic search, constraints
- Logic & logical reasoning
- Reasoning about actions and time

In the remainder of the course we will talk about . . .

- Planning
- Uncertainty
Planning is the task of coming up with a sequence of actions that will achieve a goal.

We are only considering classical planning in which environments are:

- fully observable (accessible),
- deterministic,
- finite,
- static (up to agents’ actions),
- discrete (in actions, states, objects and events).

Some of these assumptions will be relaxed in the “uncertainty” part of this course.
Why planning?

So far we have dealt with two types of agents:

1. Search-based problem-solving agents
2. Logical planning agents

Do these techniques work for solving planning problems?
Why planning?

- Consider a search-based problem-solving agent in a robot shopping world
- Task: Go to the supermarket and get milk, bananas and a cordless drill
- What would a search-based agent do?
Problems with search

- No goal-directedness.
- No problem decomposition into sub-goals that build on each other
  - May undo past achievements
  - May go to the store 3 times!
- Simple goal test doesn’t allow for the identification of milestones
- How do we find a good heuristic function?
  - How do we model the way humans perceive complex goals and the quality of a plan?

17/02/2017 Michael Herrmann Inf2D 15
How about logic & deductive inference?

- Generally a good idea, allows for “opening up” representations of states, actions, goals and plans
- If $Goal = \text{Have}(\text{Bananas}) \land \text{Have}(\text{Milk})$ this allows achievement of sub-goals (if independent)
- Current state can be described by properties in a compact way (e.g. $\text{Have}(\text{Drill})$ stands for hundreds of states)
- Allows for compact description of actions, for example

$$Object(x) \Rightarrow \text{Can}(a, \text{Grab}(x))$$

- Allows for representing a plan hierarchically, e.g.

$$\text{GoTo}(\text{Supermarket}) = \text{Leave}(\text{House}) \land \text{ReachLocationOf}(\text{Supermarket}) \land \text{Enter}(\text{Supermarket})$$
then decompose further into sub-plans
How about logic & deductive inference?

Problems:

- In its general form either awkward (propositional logic) or tractability problems (first-order logic), high complexity
- If \( p \) is a sequence that achieves the goal, then so is \( [a, a^{-1}|p] \)!

Solutions: We need

1. To reduce complexity to allow scaling up.
2. To allow reasoning to be guided by plan ‘quality’/efficiency.

Do 1 today; 2 later.
Need a language expressive enough to cover interesting problems, restrictive enough to allow efficient algorithms.

**Planning Domain Definition Language (PDDL)**

PDDL will allow you to express:

1. states
2. actions: a description of transitions between states
3. and goals: a (partial) description of a state.
States represented as conjunctions of propositional or function-free first order positive literals:

- $Happy \land Sunshine$
- $At(Plane_1, Melbourne) \land At(Plane_2, Sydney)$

These aren’t states:

- $At(x, y)$ (no variables allowed),
- $Love(Father(Fred), Fred)$ (no function symbols allowed)
- $\neg Happy$ (no negation allowed).

Closed-world assumption! (i.e. non-derivable sentences are assumed to be false)

A goal is a partial description of a state, and you can use negation, variables etc. to express that description.

- $\neg Sunshine, At(x, EDI), Love(Father(Fred), Fred)$, . . .
Actions in PDDL

Action($Fly(p, from, to)$),

Precond: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$

Effect: $\neg At(p, from) \land At(p, to))$

- Actually action schemata, as they may contain variables
- Action name and parameter list serves to identify action
- Precondition: defines states in which action is executable:
  - Conjunction of positive and negative literals, where all variables must occur in action name.
- Effect: defines how literals in the input state get changed (anything not mentioned stays the same).
  - Conjunction of positive and negative literals, with all its variables also in the preconditions.
  - Often positive and negative effects are divided into add list and delete list
The Semantics of PDDL

States and their Descriptions

\[ s \models \text{At}(P_1, \text{EDI}) \iff \text{At}(P_1, \text{EDI}) \in s \]
\[ s \models \neg \text{At}(P_1, \text{EDI}) \iff \text{At}(P_1, \text{EDI}) \notin s \]
\[ s \models \varphi(x) \iff \text{there is a ground term } d \text{ such that } s \models \varphi[x/d]. \]
\[ s \models \varphi \land \psi \iff s \models \varphi \text{ and } s \models \psi \]
Applicable Actions

- Any action is applicable in any state that satisfies the precondition with an appropriate substitution for parameters.

Example: State

\[ \text{At}(P_1, Melbourne) \land \text{At}(P_2, Sydney) \land \text{Plane}(P_1) \land \text{Plane}(P_2) \]
\[ \land \text{Airport}(Sydney) \land \text{Airport}(Melbourne) \land \text{Airport}(Heathrow) \]
satisfies precondition

\[ \text{At}(p, from) \land \text{Plane}(p) \land \text{Airport}(from) \land \text{Airport}(to) \]

with substitution (among others)

\[ \{ p/P_2, \ from/Sydney, \ to/Heathrow \} \]
The semantics of PDDL: The Result of an Action

- **Result** of executing action \( a \) in state \( s \) is state \( s' \) with any positive literal \( P \) in \( a \)'s **Effects** added to the state and every negative literal \( \neg P \) removed from it (under the given substitution).

- In our example \( s' \) would be

\[
\text{At}(P_1, \text{Melbourne}) \land \text{At}(P_2, \text{Heathrow}) \land \text{Plane}(P_1) \land \text{Plane}(P_2) \\
\land \text{Airport}(\text{Sydney}) \land \text{Airport}(\text{Melbourne}) \land \text{Airport}(\text{Heathrow})
\]

- “PDDL assumption”: every literal not mentioned in the effect remains unchanged (cf. frame problem)

- **Solution** = action sequence that leads from the initial state to a state that satisfies the goal.
Blocks world example

- Given: A set of cube-shaped blocks sitting on a table
- Can be stacked, but only one on top of the other
- Robot arm can move around blocks (one at a time)
- Goal: to stack blocks in a certain way
- Formalisation in PDDL:
  - \( \text{On}(b, x) \) to denote that block \( b \) is on \( x \) (block/table)
  - \( \text{Move}(b, x, y) \) to indicate action of moving \( b \) from \( x \) to \( y \)
  - Precondition for this action: nothing must be stacked on \( y \): \( \text{Clear}(y) \) etc.  
    (\( x \) was considered irrelevant in the previous Move function)
Blocks world example

- Action schema:
  
  \[
  \text{Action}(\text{Move}(b, x, y), \\
  \quad \text{Precond} : \text{On}(b, x) \land \text{Clear}(b) \land \text{Clear}(y) \\
  \quad \text{Effect} : \text{On}(b, y) \land \text{Clear}(x) \land \neg \text{On}(b, x) \land \neg \text{Clear}(y))
  \]

- Problem: When \( x = \text{Table} \) or \( y = \text{Table} \) we infer that the table is clear when we have moved a block from it (not true) and require that table is clear to move something on it (not true)

- Solution: Introduce another action
  
  \[
  \text{Action}(\text{MoveToTable}(b, x), \\
  \quad \text{Precond} : \text{On}(b, x) \land \text{Clear}(b) \\
  \quad \text{Effect} : \text{On}(b, \text{Table}) \land \text{Clear}(x) \land \neg \text{On}(b, x))
  \]
Interpret $Clear(b)$ as “there is space on $b$ to hold a block” (thus $Clear(Table)$ is always true)

But without further modification, planner can still use $Move(b, x, Table)$:
- Needlessly increases search space
  (not a big problem here, but can be)

So part of solution is to also add $Block(b) \land Block(y)$ to precondition of $Move$
Defined the planning problem
Discussed problems with search/logic
Introduced PDDL: a special representation language for planning
Blocks world example as a famous application domain
Next time: Algorithms for planning!
State-Space Search and Partial-Order Planning