Planning and Search

Classical Planning

Outline

- ♢ Search vs. planning
- ♢ STRIPS operators
- ♢ PDDL
- ♢ Forward (progression) state-space search
- ♢ Backward (regression) relevant-states search
- ♢ Partial-order planning

Search vs. planning contd.

Planning systems do the following:

1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

	Search	**Planning**
States	data structures	Logical sentences
Actions	code	Preconditions/outcomes
Goal	code	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

Classical planning

Assumptions are:

(1) Environment is deterministic

(2) Environment is observable

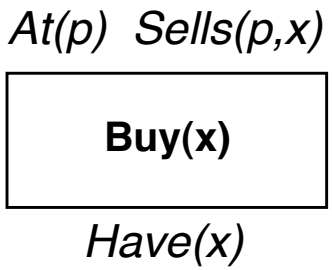
(3) Environment is static (it only in response to the agent's actions)

STRIPS operators

STRIPS panning language (Fikes and Nilsson, 1971)

Tidily arranged actions descriptions, restricted language

ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p, x)$
EFFECT: $Have(x)$



[Note: this abstracts away many important details!]

Restricted language $\Rightarrow$ efficient algorithm
Precondition: conjunction of positive literals
Effect: conjunction of literals

PDDL

Planning Domain Definition Language

A bit more relaxed that STRIPS

Preconditions and goals can contain negative literals

ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p, x)$
EFFECT: $Have(x)$

is called an action schema

Planning domain

States are sets of fluents (ground, functionless atoms). Fluents which are not mentioned are false. (Closed world assumption.)

$a \in \textsc{Actions}(s)$ iff $s \models \textsc{Precond}(a)$

$\textsc{Result}(s, a) = (s - \textsc{Del}(a)) \cup \textsc{Add}(a)$

where $\textsc{Del}(a)$ is the list of literals which appear negatively in the effect of a, and $\textsc{Add}(a)$ is the list of positive literals in the effect of a.

Example (slightly modified)

ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p, x), Have(Money)$
EFFECT: $Have(x), \neg Have(Money)$

$\text{DEL}(Buy(Jaguar)) = \{Have(Money)\}$

$\text{ADD}(Buy(Jaguar)) = \{Have(Jaguar)\}$

If $s = \{At(JDealer), Sells(JDealer, Jaguar), Blue(Sky), Have(Money)\}$,

$Buy(Jaguar) \in \text{ACTIONS}(s)$

$\text{RESULT}(s, Buy(Jaguar) = (s - \{Have(Money)\}) \cup \{Have(Jaguar)\}$

$= \{At(JDealer), Sells(JDealer, Jaguar), Blue(Sky), Have(Jaguar)\}$

Planning problem

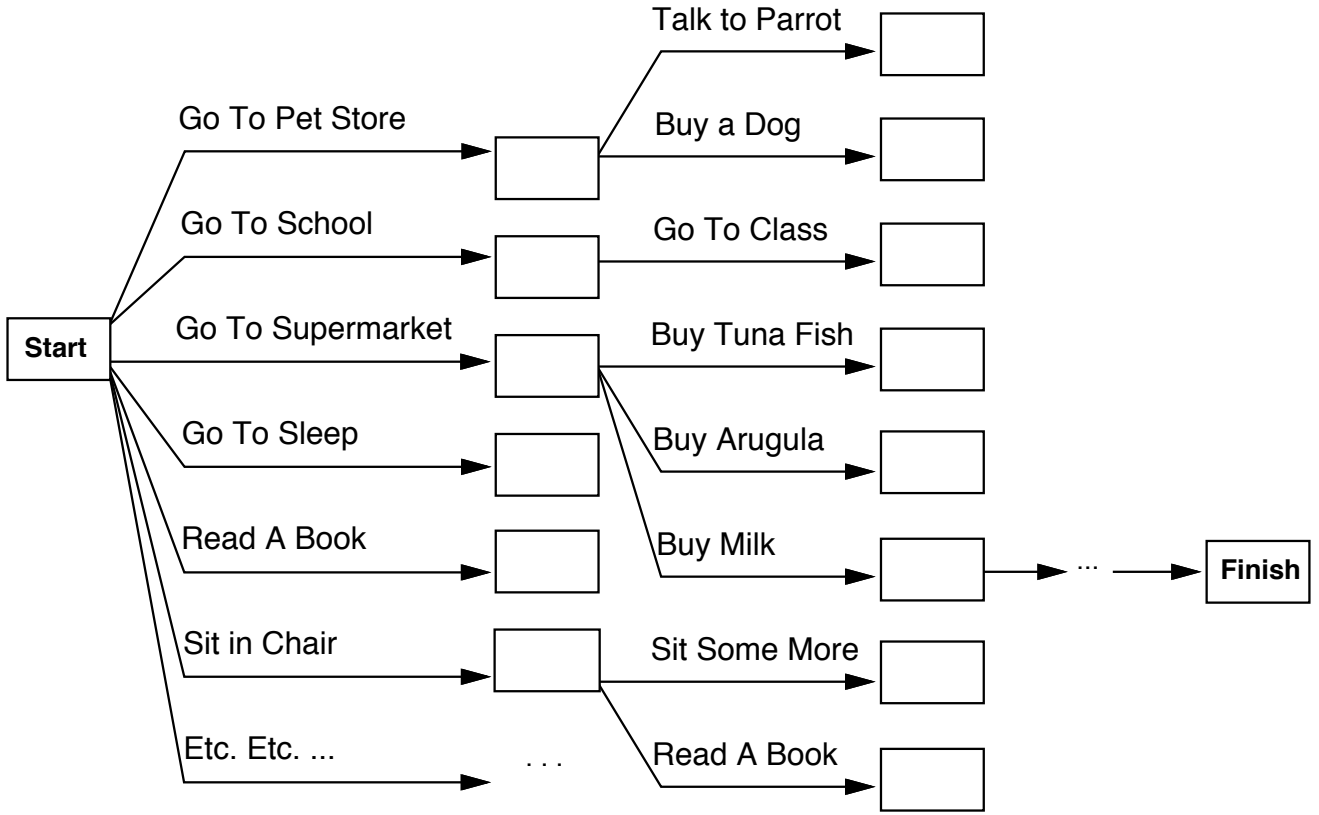
Planning problem = planning domain + initial state + goal

Goal is a conjunction of literals: $Have(Jaguar) \wedge \neg At(Jail)$

Can solve planning problem using search

Forward (progression) planning

Searching for a solution starting from the initial state looks hopeless



Forward planning 2

However, it turns out we can automatically derive good heuristics (and remember how much better A^* is compared to uninformed search)

Two basic approaches:

1) add more edges to the graph (make more actions possible), and use solutions to the resulting problem as a heuristic

Examples: remove (some) preconditions, ignore delete lists...

ACTION $Slide(t, s_1, s_2))$
PRECOND: $On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$
EFFECT: $On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2)$

removing $Blank(s_2)$ will enable tiles to move to occupied places: Manhattan distance heuristic

2) abstract the problem (make the search space smaller).

Backward (regression) planning

Also called relevant-states search

Start at the goal state(s) and do regression (go back):

Given a goal description g and a ground action a, the regression from g over a gives a state description g':

$$g' = (g - \textsc{Add}(a)) \cup \{\textsc{Precond}(a)\}$$

For example, if the goal is $Have(Jaguar) \wedge \neg At(Jail)$,

$$g' = (\{Have(Jaguar), \neg At(Jail)\} - \{Have(Jaguar)\}) \cup$$

$$\{At(p), Sells(p, Jaguar), Have(Money)\} =$$

$$\{\neg At(Jail), At(p), Sells(p, Jaguar), Have(Money)\}$$

note that g' is partially uninstantiated (p is a free variable)

Backward (regression) planning 2

Which actions to regress over?

Relevant actions: have an effect which is in the set of goal elements and no effect which negates an element of the goal.

For example, $Buy(Jaguar)$ is a relevant action. $Steal(Jaguar)$ may also result in $Have(Jaguar)$ but if it has an additional effect of $At(Jail)$, it is not a relevant action.

Search backwards from g, remembering the actions and checking whether we reached an expression applicable to the initial state.

A lot fewer actions/relevant states than forward search, but uses sets of states (g, g') - hard to come up with good heuristics.

Totally vs partially ordered plans

So far we produced a linear sequence of actions (totally ordered plan)

Often it does not matter in which order $some\ of\ the\ actions$ are executed

For problems with independent subproblems often easier to find a **partially ordered plan**: a plan which is a set of actions and a set of constraints $Before(a_i, a_j)$

Partially ordered plans are created by a search through a space of plans (rather than the state space)

Next lecture

More classical planning