Planning

Lecture 1

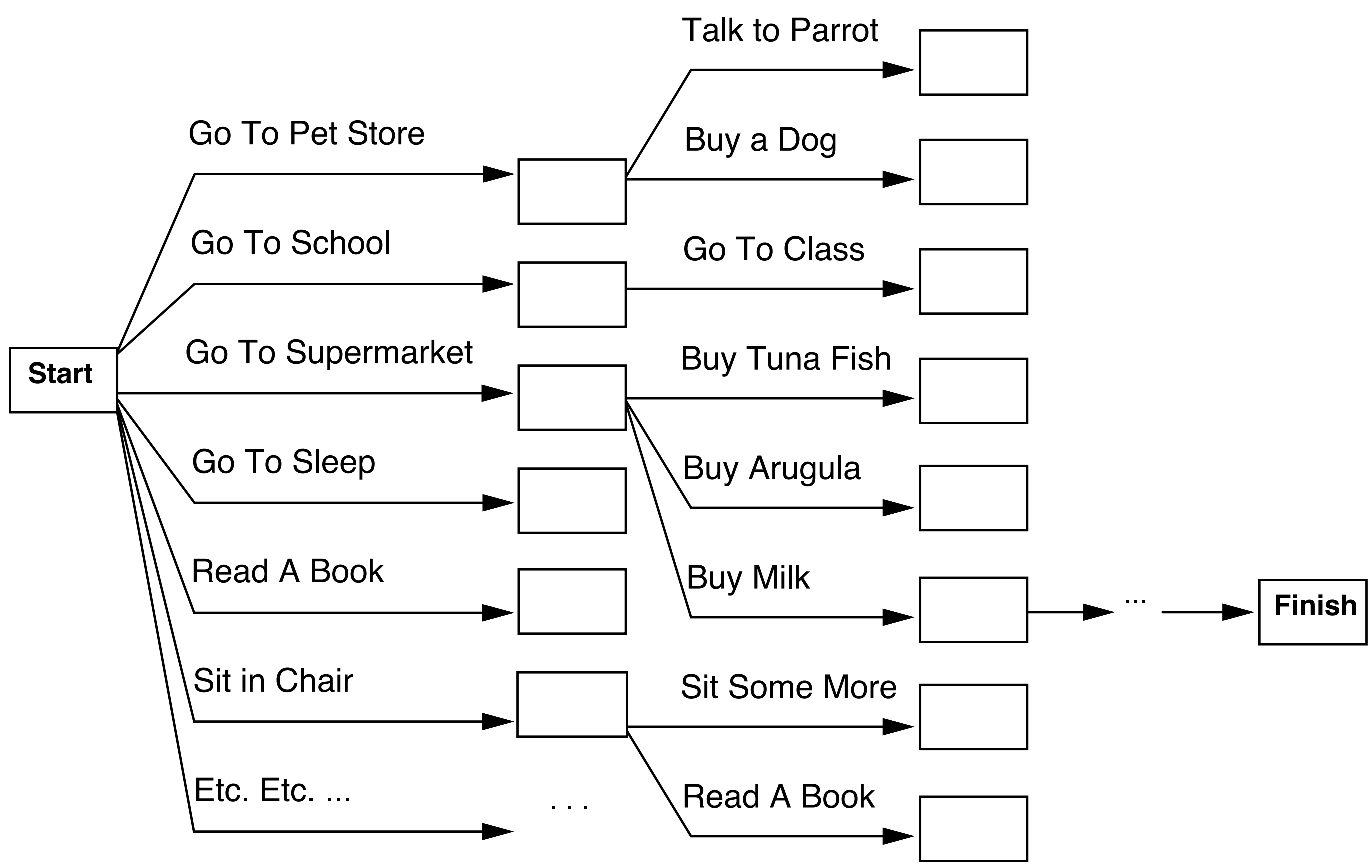
Outline

♢ The planning problem, STRIPS operators and planning specifications (RN 10.1)

♢ Planning in the state-space (RN 10.2)

Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*
Standard search algorithms seem to fail miserably:



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

Classical Planning

Environment:

- ♢ fully observable
- ♢ deterministic
- ♢ static
- ♢ discrete (and finite)

A plan is a **sequence** of actions

STRIPS

STRIPS: STanford Research Institute Problem Solver, language for action description.

Literals are expressions of the form $P(x_1, \ldots, x_n)$ or $\neg P(x_1, \ldots, x_n)$, where $0 \leq 0n$, and x_i are either variable or constant symbols

States are represented by **sets of instantiated literals**, which represent properties that must be satisfied in the state, with a boolean $\{true, false\}$ value

$\diamondsuit$ **closed-world assumption**: a state contains only positive literals (all the missing ones are assumed to be $false$)

$\diamondsuit$ **function free**: the domain is assumed to be finite

STRIPS: example

Init state:

$$At(Home) \wedge sells(SM, Milk) \wedge sells(SM, Bananas) \wedge sells(HWS, D$$

Goal:

$$At(Home) \wedge have(Milk) \wedge have(Bananas) \wedge have(Drill)$$

♢ "factorized" representation allows for domain independent heuristics

♢ "restricted" language aims at ad hoc efficient algorithms

Action Representation

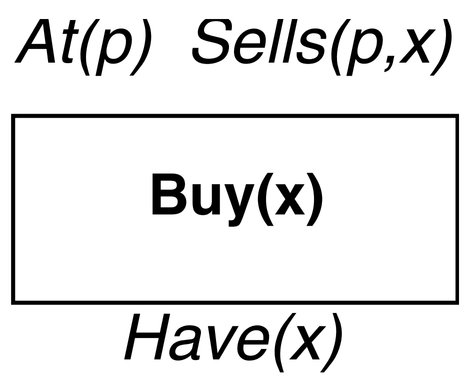
Action schemas:

Precondition: conjunction (set) of positive literals
Effect: conjunction (sets) of literals

Note 1: variables can be instantiated to a finite number of individuals.
Note 2: In STRIPS effects specify literals to be added or removed from the state (called ADD list and DELETE list).
Note 3: Can be generalized to both positive and negative literals also in preconditions

Action schema: example

$\text{ACTION: } Buy(x)$
$\text{PRECONDITION: } At(p), Sells(p, x)$
$\text{EFFECT: } Have(x)$



Applicable actions

An action a is **applicable** in a state s iff:

$$s \models Preconditions(a)$$

Note 1: Variables are instantiated to match the preconditions in s; all the variables in the effects must be instantiated by the precondition

Note 2: if there are no negative literals in the preconditions $\models$ is simply set containment, otherwise the negative literals in the precondition should not belong to s (not positive in s)

Computing the successor state

Computing the state resulting from action execution:

$$Result(s, a) = (s - DEL(a)) \cup ADD(a)$$

Variables are instantiated by matching the preconditions

STRIPS makes an implicit **persistence** assumption (to solve the so-called **frame** problem):

> **Everything not explicitely changed by effects persists, after the execution of an action.**

Semantics of STRIPS

- Operational (discussed earlier)
- Logic (Situation Calculus, after we do logic)
- State transition systems

$\Sigma = (S, A, \gamma)$

S is a finite set of states
A is a finite set of actions
$\gamma(s, a) = s'$ is a transition function, specifying the result of the execution of the action a in the state s.

Planning problem

1. A specification of **actions** (via action schemas)
2. The **initial state**: $I \subseteq S$
3. The **goal**: $G \subseteq S$

The specification of the actions is called **planning domain**.

Action schemas are a compact representation of the transition system (allowing variables and action parameters).

Planning problem: example

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

ACTION: $Go(x)$
PRECONDITION: $At(y)$
EFFECT: $At(x) \wedge \neg At(y)$

Init state:

$$At(Home) \wedge sells(SM, Milk) \wedge sells(SM, Bananas) \wedge$$
$$sells(HWS, Drill) \cdots$$

Goal:

$$At(Home) \wedge have(Milk) \wedge have(Bananas) \wedge have(Drill)$$

Plans

A **plan** is a sequence of actions that allows us to reach a state where the goal condition holds, starting from the initial state.

Example:

$Go(SM)$,
$Buy(Milk)$,
$Buy(Bananas)$,
$Go(HWS)$,
$Buy(Drill)$,
$Go(Home)$

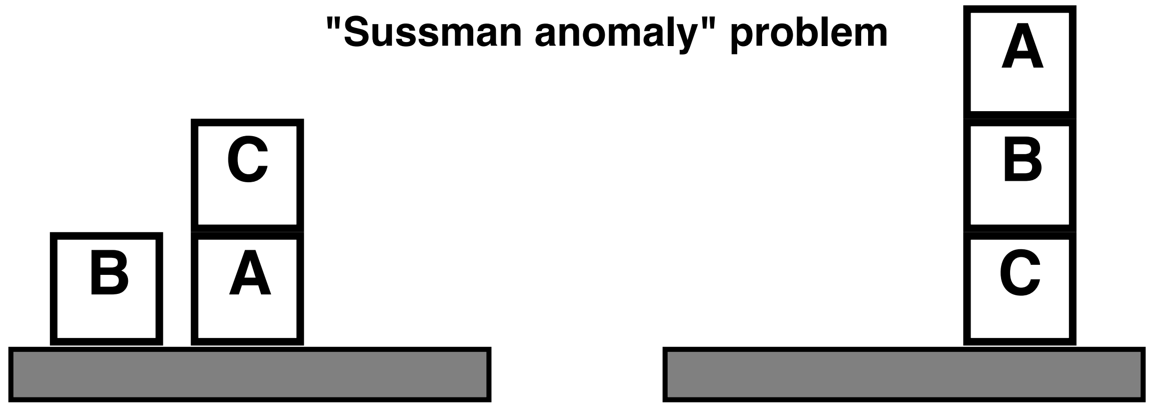
“Standard” language for planning problems

PDDL: Planning Domain Description Language

- de facto standard plan specification language
- generalizes STRIPS and other planning languages
- several extensions to express non deterministic actions, sensing, ...

Example: Blocks world

"Sussman anomaly" problem



Start State | Goal State

Clear(x) On(x,z) Clear(y)	*Clear(x) On(x,z)*
PutOn(x,y)	PutOnTable(x)
~On(x,z) ~Clear(y) Clear(z) On(x,y)	*~On(x,z) Clear(z) On(x,Table)*

\+ several inequality constraints

Actions in the blocks world

ACTION: $PutOn(b, y)$
PRECONDITION: $On(b, z), Clear(b), Clear(y)$
EFFECT: $On(b, y), Clear(z), \neg On(b, y), \neg Clear(y)$

ACTION: $PutOnTable(b)$
PRECONDITION: $On(b, z) \wedge Clear(b)$
EFFECT: $On(b, Table), Clear(z), \neg On(b, z)$

PDDL: blocks world domain

```
(define (domain blocks-world)
   (:requirements :equality)
   (:predicates (on ?B ?O) (clear ?O) (table ?O) (block ?B))
   (:action puton              ; put ?B1 onto ?B2 from ?O
        :parameters (?B1 ?B2 ?O)
        :effect (and (on ?B1 ?B2) (clear ?O)
                     (not (on ?B1 ?O)) (not (clear ?B2)))
        :precondition (and (on ?B1 ?O) (clear ?B1) (clear ?B2)
                           (not (= ?B1 ?O)) (not (= ?B2 ?B1)) (not (= ?B2 ?O))
                           (block ?B1) (block ?B2)))
   (:action putTable
        :parameters (?B1 ?T ?B2) ; put ?B1 onto table ?T from ?B2
        :effect (and (on ?B1 ?T) (clear ?B2) (not (on ?B1 ?B2)))
        :precondition (and (on ?B1 ?B2) (clear ?B1) (not (= ?B1 ?B2))
                           (block ?B2) (table ?T)))
)
```

PDDL: blocks world problem instance

```
(define (problem bw1)
  (:domain blocks-world)
  (:objects a b c t)
  (:init (block a) (block b) (block c) (table t)
         (on a b) (on b t) (on c t) (clear a) (clear c))
  (:goal (and (on c a) (on b c)))
```

Planning techniques

- Search in the state space
- Search in the plan space
- Hierarchical planning

Planning in the state space

The search problem

- states are characterized by sets of propositions (by conjunctive formula)
- operators are determined by action specifications
- initial state is given
- final state must satisfy the goal
- step cost $= 1$

Complexity of classical planning

PLANSAT: Find whether there exists a sequence of actions that leads to a goal state from the initial state

Bounded PLANSAT: Find whether there exists a sequence of actions of length k or less that leads to a goal state from the initial state (allows to determine optimal plans)

- both PLANSAT and Bounded PLANSAT are in PSPACE;
- disallow negative effects: PLANSAT and Bounded PLANSAT are NP-hard
- disallow negative effects and negative preconditions: PLANSAT in P

Search for a plan

Given the above complexity bounds: efficient search becomes the key issue.

- progression
- regression
- heuristics

Progression

Search **forward**: apply actions whose preconditions are satisfied until goal state is found or all states have been explored

$$Result(s, a) = (s - DEL(a)) \cup ADD(a)$$

Irrelevant actions cause the search space to blow-up.

Good, problem independent, heuristics needed

Regression

Search **backward**:

- ♢ STRIPS operators are easily **invertible**

- ♢ the goal specification represents **sets of states**

- ♢ focusses on the actions that make some of the goal conjuncts true

Regression: remarks

Choice of actions to be regressed:

- **relevant** actions (i.e. actions that add a goal conjunct)
- **consistent** actions (i.e. not undo any other goal conjunct)

♢ To produce a predecessor state from action A (for goal G):

- any positive effect of A that appears in G is deleted
- each precondition literal of A is added, unless it already appears

$$Result^{-1}(s, a) = (s - EFF(a)) \cup PREC(a)$$

State representation in regression

The goal represents a set of states!

A regression step may introduce variables, when the preconditions are not fully instantiated:

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

Heuristics search

Naive regression performs better than naive progression but they are both unsatisfactory.

Consider using $A*$: a good heuristics is based on a good estimate of the distance to the goal.

State space planners won the AIPS 2000 competition, thus renewing the interest for their practical applicability.

Specifically, FF (1998) revamped Forward Search.

Heuristics for state space search I

relaxing the problem: increasing the number of arcs in the state space

- **removing preconditions** $\rightarrow$ set cover
- **removing negated effects** (empty-delete-list)

Note 1: both may lead to NP-hard problems, but fast approximate solutions can be effective.

Note 2: care must be taken to guarantee admissibility in computing approximations.

Heuristics for state space search II

relaxing the problem: abstracting sets of states

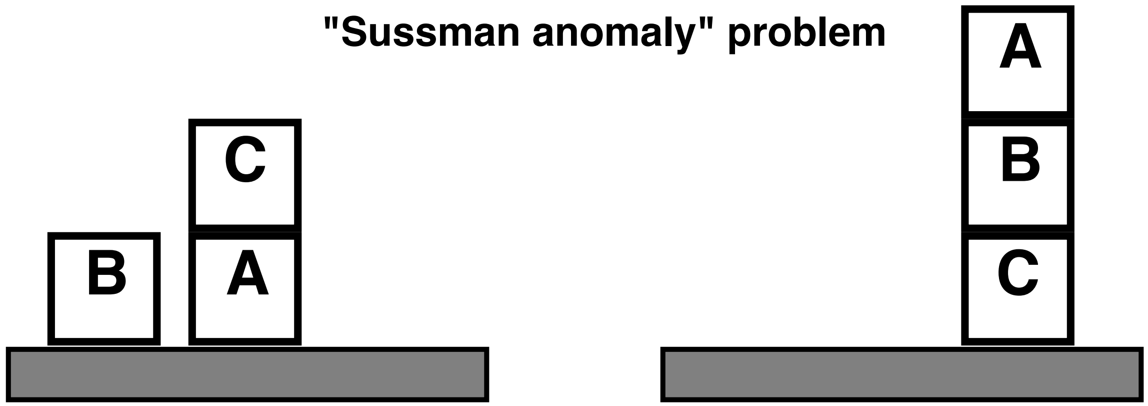
- **subgoal independence assumption**: the cost of solving a conjunction estimated based on the costs of solving each of the conjuncts (i.e. MAX or SUM)

Note: when pessimistic (solving one subgoal can help solving other subgoals) is not admissible, hence SUM is in general not admissible.

More about heuristics with **GRAPHPLAN**

Example: Blocks world

"Sussman anomaly" problem



Start State | Goal State

Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y)
Clear(z) On(x,y)

Clear(x) On(x,z)

PutOnTable(x)

~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

Actions in the blocks world

Action: $PutOn(b, y)$
Precondition: $On(b, z), Clear(b), Clear(y)$
Effect: $On(b, y), Clear(z), \neg On(b, y), \neg Clear(y)$

Action: $PutOnTable(b)$
Precondition: $On(b, z) \wedge Clear(b)$
Effect: $On(b, Table), Clear(z), \neg On(b, z)$

STRIPS planning

Basic idea: **goal stack**: work on one goal until completely solved before moving on to the next goal

Goal: $On(A, B) \wedge On(B, C)$

Start with Goal $On(A, B)$:
$PutOnTable(C)$
$PutOn(A, B)$ Achieves first goal
Move to Goal $On(B, C)$:
$PutOnTable(A)$
$PutOn(B, C)$ Achieves second goal but loses first

Again goal $On(A, B)$: $PutOn(A, B)$ Achieves, both goals

STRIPS plannning II

Start with Goal $On(B, C)$:
$PutOn(B, C)$ achieves first goal
Move to Goal $On(A, B)$:
$PutOnTable(B)$
$PutOnTable(C)$
$PutOn(A, B)$Achieves second goal, but looses first

Again goal $On(B, C)$: $PutOnTable(A), PutOn(B, C)$
Again goal $On(A, B)$: $PutOn(A, B)$ Achieves both goals

STRIPS plannning III

but, the shortest plan is:

$PutOnTable(C)$	goal $on(A, B)$
$PutOn(B, C)$	goal $on(B, C)$
$PutOn(A, B)$	goal $on(A, B)$

Linear vs Non Linear planning:
consider sets of goals to allow **interleaving**
PRODIGY (Veloso et al.)