

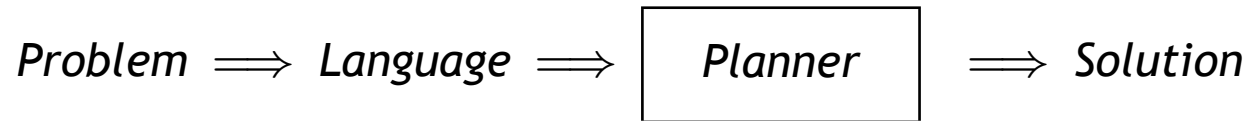
# Artificial Intelligence Planning

Héctor Geffner

ICREA and Universitat Pompeu Fabra  
Barcelona, Spain

# Planning: Motivation

- Planning is a form of **general problem solving**



- **Idea:** problems **described** at high-level and **solved** automatically
- **Goal:** facilitate modeling with small penalty in performance

# Planning and General Problem Solving: How general?

For which class of problems planner should work?

- **Classical planning** focuses on problems that map into deterministic state models:
  - state space  $S$
  - initial state  $s_0 \in S$
  - goal states  $S_G \subseteq S$
  - actions  $A(s)$  applicable in each  $s$
  - transition function  $s' = f(a, s)$ ,  $a \in A(s)$
- This class includes **sequential decision problems with deterministic actions and full information**
- Other forms of planning involve different models . . .

# Classical Planning

- Three components in modeling and problem solving:
  - **languages** for describing problems conveniently
  - **models** for making sense of classes of problems
  - **algorithms** for solving the models
- In **classical planning**, the mathematical model is fixed and corresponds to the (deterministic) **state model**
- The two key elements in classical planning are **languages** and **algorithms**
- Languages include **Strips**, **ADL**, **PDDL**, etc

# Planning Languages

- Planning languages provide a convenient, declarative way for describing **state models**
- They capture fragments of predicate logic + extensions for expressing
  - the set of **available actions**
  - the **conditions** under which they are applicable, and
  - the **effects** they produce
- In addition, they must accommodate information about the **initial situation** and the **goal situation**

# The Strips Language (Fikes and Nilsson 71)

- **Strips** is one of the oldest, simplest, and most used planning language, even if **not too flexible**
- A **problem** in Strips is a tuple  $\langle A, O, I, G \rangle$  where
  - $A$  stands for set of all **atoms** (boolean vars)
  - $O$  stands for set of all **operators** (ground actions)
  - $I \subseteq A$  stands for **initial situation**
  - $G \subseteq A$  stands for **goal situation**
- Operators  $o \in O$  **represented** by three lists
  - the **Add** list  $Add(o) \subseteq A$
  - the **Delete** list  $Del(o) \subseteq A$
  - the **Precondition** list  $Pre(o) \subseteq A$
- Intuitively,  $Pre(o)$  encodes the atoms that must be true for  $o$  to be applicable,  $Add(o)$  encodes the atoms that  $o$  makes true, and  $Del(o)$  encodes the atoms that  $o$  makes false

# From Language to Models: Semantics of Strips

Strips problem  $P = \langle A, O, I, G \rangle$  determines **state model**  $\mathcal{S}(P)$

- the states  $s \in \mathcal{S}$  are **collections of atoms**
- the initial state  $s_0$  is  $I$
- the goal states  $s \in \mathcal{S}_G$  are such that  $G \subseteq s$
- the actions in  $s$  are the  $op \in O$  s.t.  $Prec(op) \subseteq s$
- the state that results from doing  $op$  in  $s$  is  $s' = s - Del(op) + Add(op)$
- action costs  $c(op, s)$  are all 1

The (optimal) **solution** of planning problem  $P$  is the (optimal) solution of State Model  $\mathcal{S}(P)$

## Example: Blocks in Strips (PDDL Syntax)

```
(define (domain BLOCKS)
  (:requirements :strips) ...
  (:action pick_up
    :parameters (?x)
    :precondition (and (clear ?x) (ontable ?x) (handempty))
    :effect (and (not (ontable ?x)) (not (clear ?x)) (not (handempty)) (hol
  (:action put_down
    :parameters (?x)
    :precondition (holding ?x)
    :effect (and (not (holding ?x)) (clear ?x) (handempty) (ontable ?x)))
  (:action stack
    :parameters (?x ?y)
    :precondition (and (holding ?x) (clear ?y))
    :effect (and (not (holding ?x)) (not (clear ?y)) (clear ?x) (handempty)
      (on ?x ?y))) ...
(define (problem BLOCKS_6_1)
  (:domain BLOCKS)
  (:objects F D C E B A)
  (:init (CLEAR A) (CLEAR B) ... (ONTABLE B) ... (HANDEEMPTY))
  (:goal (AND (ON E F) (ON F C) (ON C B) (ON B A) (ON A D))))
```

## Example: Logistics in Strips/PDDL

```
(define (domain logistics)
  (:requirements :strips :typing :equality)
  (:types airport - location truck airplane - vehicle vehicle packet - thing thing)
  (:predicates (loc-at ?x - location ?y - city) (at ?x - thing ?y - location) (in ?x - location))
  (:action load
    :parameters (?x - packet ?y - vehicle)
    :vars (?z - location)
    :precondition (and (at ?x ?z) (at ?y ?z))
    :effect (and (not (at ?x ?z)) (in ?x ?y)))
  (:action unload ..)
  (:action drive
    :parameters (?x - truck ?y - location)
    :vars (?z - location ?c - city)
    :precondition (and (loc-at ?z ?c) (loc-at ?y ?c) (not (= ?z ?y)) (at ?x ?z))
    :effect (and (not (at ?x ?z)) (at ?x ?y)))
  ...
(define (problem log3_2)
  (:domain logistics)
  (:objects packet1 packet2 - packet truck1 truck2 truck3 - truck airplane1 - airplane)
  (:init (at packet1 office1) (at packet2 office3) ...))
  (:goal (and (at packet1 office2) (at packet2 office2))))
```

## Example: 15-Puzzle in PDDL

```
(define (domain tile)
  (:requirements :strips :typing :equality)
  (:types tile position)
  (:constants blank - tile)
  (:predicates (at ?t - tile ?x - position ?y - position)
    (inc ?p - position ?pp - position)
    (dec ?p - position ?pp - position))
  (:action move-up
    :parameters (?t - tile ?px - position ?py - position ?bx - position ?by - position)
    :precondition (and (= ?px ?bx) (dec ?by ?py) (not (= ?t blank)) ...)
    :effect (and (not (at blank ?bx ?by)) (not (at ?t ?px ?py)) (at blank ?px ?py) (
      ...
  (define (domain eight_tile) ..
  (:constants t1 t2 t3 t4 t5 t6 t7 t8 - tile      p1 p2 p3 - position)
  (:timeless (inc p1 p2) (inc p2 p3) (dec p3 p2) (dec p2 p1)))

(define (situation eight_standard)
  (:domain eight_tile)
  (:init (at blank p1 p1) (at t1 p2 p1) (at t2 p3 p1) (at t3 p1 p2) ..)
  (:goal (and (at t8 p1 p1) (at t7 p2 p1) (at t6 p3 p1) ..)
```

# Computation: how to solve Strips planning problems?

- **Key issue: exploit two roles of language:**
  - **specification:** concise model description
  - **computation:** reveal useful heuristic info
- **Two traditional approaches: search vs. decomposition**
  - explicit **search** of the state model  $S(P)$  direct but not effective til recently
  - **near decomposition** of the planning problem thought a better idea

# Approaches to Classical Strips Planning

- **Strips algorithm** (70's): Total ordering planning backward from Goal; work always on **top** subgoal in stack, delay rest
- **Partial Order (POCL) Planning** (80's): work on **any** subgoal, resolve threats; UCPOP 1992
- **Graphplan** (1995 -- . . . ): build graph containing all possible **parallel** plans up to certain length; then extract plan by searching the graph backward from Goal
- **SatPlan** (1996 -- . . . ): map planning problem given horizon into SAT problem; use state-of-the-art SAT solver
- **Heuristic Search Planning** (1996 -- . . . ): search state space  $\mathcal{S}(P)$  with heuristic function  $h$  extracted from problem  $P$
- **Model Checking Planning** (1998 -- . . . ): search state space  $\mathcal{S}(P)$  with 'symbolic' BrFS where sets of states represented by formulas implemented by BDDs

# State of the Art in Classical Planning

- **fast and significant progress since Graphplan (1995)**
- **empirical methodology**
  - standard PDDL language
  - planners and benchmarks available
  - AIPS/ICAPS planning competitions . . .
- **novel formulations and ideas**
  - Graphplan, SAT, HSP, Model checking . . .
- **large problems solved**

# Heuristic Search Planning

Simple, powerful, and explains success of recent approaches in relation to previous approaches

- Explicitly **searches** state space  $S(P)$  with heuristic  $h(s)$  that estimates cost from  $s$  to Goal
- Heuristic  $h$  extracted **automatically** from problem encoding

# Heuristic Functions

- Heuristics derived as **optimal** cost function of **relaxed** problems
- Simple relaxations used in Planning
  - ignore delete lists
  - reduce large goal sets to subsets
  - ignore certain atoms

# Additive Heuristic

- For all atoms  $p$ :

$$g(p; s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s, \text{ else} \\ \min_{a \in O(p)} [1 + g(\text{Prec}(a); s)] & \end{cases}$$

- For sets of atoms  $C$ , assume **independence**:

$$g(C; s) \stackrel{\text{def}}{=} \sum_{r \in C} g(r; s)$$

- Resulting heuristic function  $h_{add}(s)$ :

$$h_{add}(s) \stackrel{\text{def}}{=} g(\text{Goals}; s)$$

Heuristic not admissible but informative and fast

# Max Heuristic

- For all atoms  $p$ :

$$g(p; s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s, \text{ else} \\ \min_{a \in O(p)} [1 + g(\text{Prec}(a); s)] & \end{cases}$$

- For sets of atoms  $C$ , replace **sum** by **max**

$$g(C; s) \stackrel{\text{def}}{=} \max_{r \in C} g(r; s)$$

- Resulting heuristic function  $h_{max}(s)$ :

$$h_{max}(s) \stackrel{\text{def}}{=} g(\text{Goals}; s)$$

Heuristic admissible but not very informative . . .

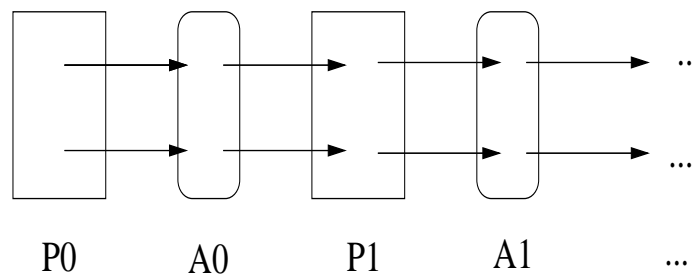
# Max Heuristic and Planning Graphs

- Build reachability graph  $P_0, A_0, P_1, A_1, \dots$

$$P_0 = \{p \in \text{Init}\}$$

$$A_i = \{a \in O \mid \text{Prec}(a) \subseteq P_i\}$$

$$P_{i+1} = P_i \cup \{p \in \text{Add}(a) \mid a \in A_i\}$$



-- Graph implicitly **represents** max heuristic:

$$h_{max}(s) = \min i \text{ such that } G \subseteq P_i$$

## More Informed $h$ in Graphplan

- Graphplan seeks optimal **parallel** plans, where actions can be done in parallel, as long they don't 'interfere' (actions interfere when one deletes a precondition or add of the other)
  - Graphplan computes heuristic  $h_G$  more informed than  $h_{max}$  by keeping track of **mutex pairs**; pairs that cannot be **simultaneously** achieved in  $i$  steps:
    - **action pair mutex** at  $i$  if actions interfere or preconditions **mutex** at  $i$
    - **atom pair mutex** at  $i + 1$  if supporting action pairs **mutex** at  $i$
  - A set of atoms  $S$  is **mutex** at  $i$  if it contains a mutex pair at  $i$
- Graphplan also adds 'dummy actions' **NO-OP**( $p$ ) for each atom  $p$  with  $Prec = Add = \{p\}$  that 'carry'  $p$ ; resulting in planning graph

## More informed $h$ in Graphplan (cont'd)

Resulting planning graph:

$$P_0 = \{p \in \text{Init}\}$$

$$A_i = \{a \in O \mid \text{Prec}(a) \text{ in } P_i \text{ and not mutex at } i\}$$

$$P_{i+1} = \{p \in \text{Add}(a) \mid a \in A_i\}$$

with sets  $MP_i$  and  $MA_i$  of **mutex pairs** as defined above

$$h_G(s) \stackrel{\text{def}}{=} \min i \text{ s.t. } G \subseteq P_i \text{ and } G \text{ not mutex at } i$$

# Example

- Initial Situation  $I = \{q\}$
- Actions:
  - $a : Pre(a) = \{q\} ; Add(a) = \{p\} ; Del(a) = \{q\}$
  - $b : Pre(a) = \{q\} ; Add(a) = \{r\} ; Del(a) = \{\}$
  - $a : Pre(a) = \{r\} ; Add(a) = \{q\} ; Del(a) = \{r\}$
- Goal:  $G = \{p, q\}$

**Exercise:** Determine  $h_{max}(G)$ ,  $h_G(G)$ ,  $h_{add}(G)$

## Generalization: $h^m$ heuristics

For fixed  $m = 1, 2, \dots$ , assume cost of achieving set  $C$  given by cost of most costly subset of size  $m$

- For  $m = 1$ ,  $h^m = h_{max}$
- For  $m = 2$ ,  $h^m = h_G$  (Graphplan)
- For any  $m$ ,  $h^m$  admissible and polynomial

$$h^m(s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } s \subseteq s_0 \\ \min_{s' \in R(s)} [1 + h^m(s')] & |s| \leq m \\ \max_{s' \subseteq s, |s'|=m} h^m(s') & |s| > m \end{cases}$$

where  $s' \in R(s)$  (regression set) if  $s' = s - Add(a) + Prec(a)$  for some  $a$  such that  $Del(a) \cap s = \emptyset$

# Searching in Heuristic Search Planning

- Heuristic Search Planners search the state space  $S(P)$  explicitly using the heuristic function  $h(s)$  extracted automatically from problem encoding
- In basic scheme, **branching** is done forward by applying all possible actions  $a$  to selected state
- Alternatively, it's possible to branch **backward** from goal rather than **forward** from the initial state
- These are called (heuristic) **regression planners**
- Other **branching schemes** possible as well . . .

# Branching in Search and Problem Solving

- 15-puzzle, Rubik, . . . :
  - **Branching:** Forward/Backward ok
- TSP (Travelling Salesman Problem)
  - **Branching:** include/exclude edge  $i-j$  in tour
  - **Termination;** when relaxed 'assignment problem' yields single tour
- JSP (Job Shop Scheduling)
  - **Branching:**  $i \prec j, j \prec i$  for pairs of tasks
  - **Termination:** when no overlap among actions requiring same resource

# Branching in Planning

- **Forward:** State-space; extend plan head; totally ordered
- **Backward:** Regression-space ; extend plan tail; totally ordered
- **Temporal:** for action  $a$  and time  $i$ , create splits  $a[i] = true / a[i] = false$
- **POCL:** Partial Order Causal Link Planning
  - precedence constrains  $a_1 \prec a_2$  (promotion/demotion) to elim 'threats'
  - causal links  $a_1 \rightarrow_P a_2$  to 'support' preconds
  - **termination:** when no threats & all preconds supported
  - .....

# Alternative Branching in Heuristic Search Planning

- **Idea:** Construct plans neither from **head** or **tail**, but by making **partial commitments**
- A 'state' (**partial plan**)  $\sigma$  is a **set** of commitments
- $h^*(\sigma)$  measures (parallel) cost of best **complete plan** compatible with **partial plan**  $\sigma$
- Definition of heuristics  $h^m$  can be transformed to provide informative lower bounds on  $h^*(\sigma)$
- Good systems for **optimal** parallel and temporal planning, will most likely rely on good **lower bounds** and suitable **branching schemes**
- Most planners can be understood along these two dimensions

# Summary: Classical Planning

- Planning as general problem solving:
  - languages, models, algorithms
- Focus on **Strips** and **Heuristic Search** approaches
- Two key notions: **Heuristics** and **Branching**
- Challenges
  - richer **action models** (uncertainty, feedback)
  - richer **temporal model** (durations, parallelism)
- Important to keep certain **distinctions** in mind
  - optimal vs. suboptimal
  - sequential vs. parallel