Planning

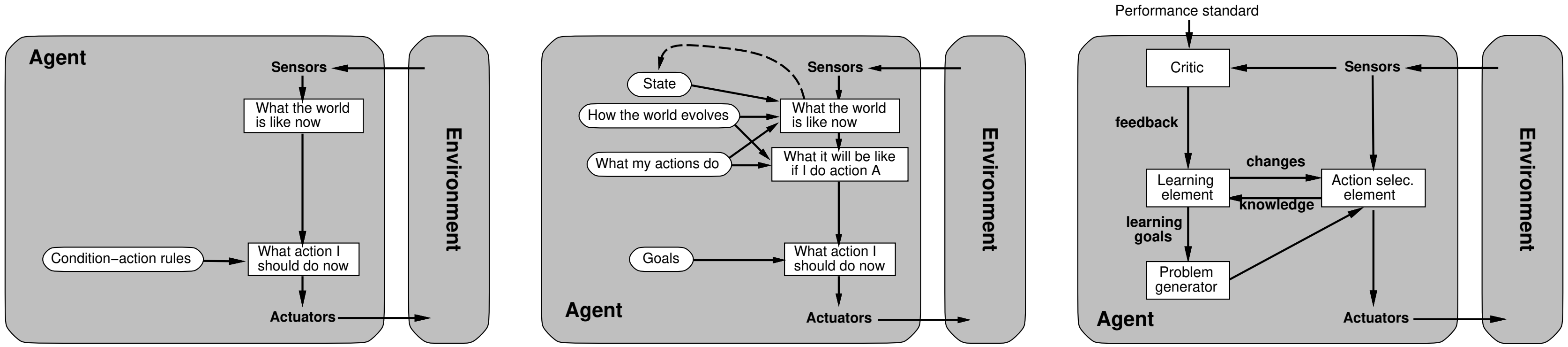
Chapter 10

Let's recap: building intelligent agents

Intelligent agents = agents that behave rationally

Three main approaches:

- programming, e.g. writing rules by hand
- reasoning (model-based) = model + representation + search
- learning



In AI, the model-based reasoning approach is called planning

Relation between planning, search, and KRR

◇ Planning looks for a sequence of actions achieving a goal state (as before)

◇ Planning uses search BUT in conjunction with adequate KR

◇ Planning uses representations of states and actions allowing us to exploit the structure of the problem and lead to general heuristics for planning.

◇ Planners are general problem solvers that take as input a description of the problem in a high-level language

Overall outline

- ♢ Planning
- ♢ Classical planning
- ♢ Representation of planning problems
- ♢ State-space planning
- ♢ Graph-based planning
- ♢ SAT-based planning
- ♢ Plan-space planning

Planning: Introduction & Representation

Chapter 10

Outline

◊ Planning

◊ Classical planning problems and plans

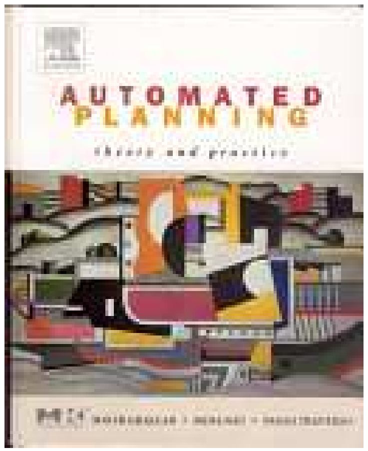
◊ Examples

◊ STRIPS and ADL representations

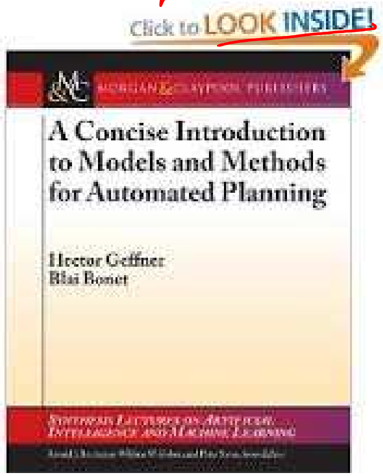
◊ PDDL: Planning Domain Definition Language

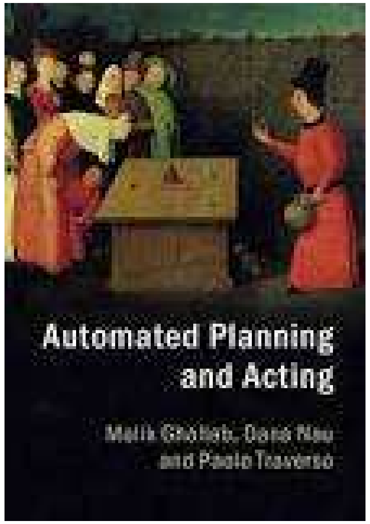
Planning

"Planning is the reasoning side of acting. It is an explicit deliberation process that chooses and organises actions, on the basis of their expected outcomes, in order to achieve some objective as best as possible." [Ghallab *et. al.*, 2003]



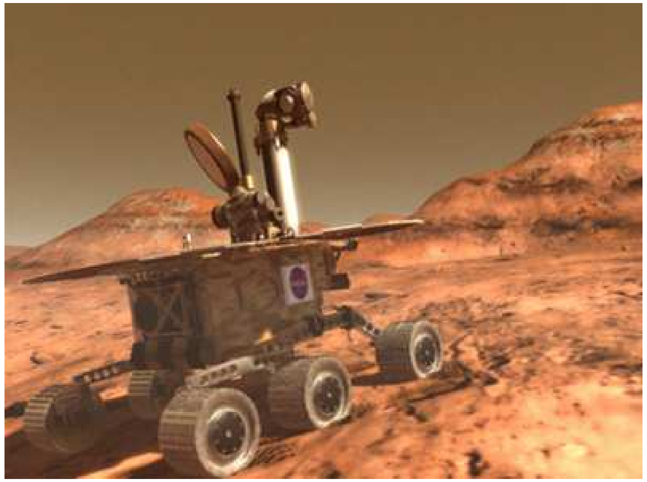
yeah sure.





Planning

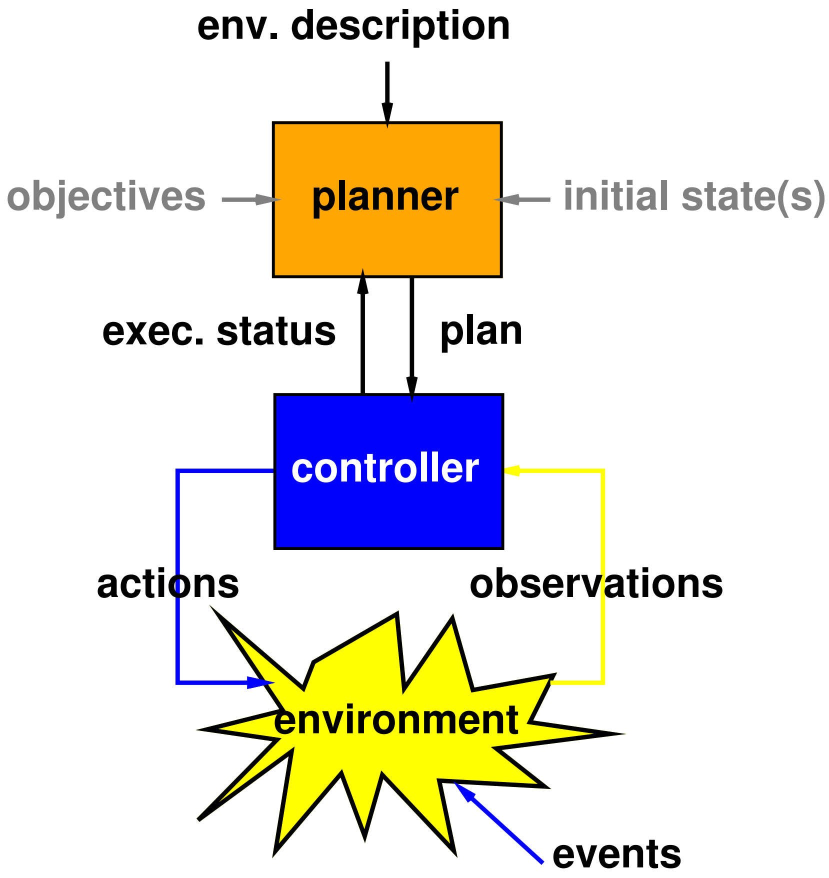
"Planning is the reasoning side of acting. It is an explicit deliberation process that chooses and organises actions, on the basis of their expected outcomes, in order to achieve some objective as best as possible." [Ghallab *et. al.*, 2003]



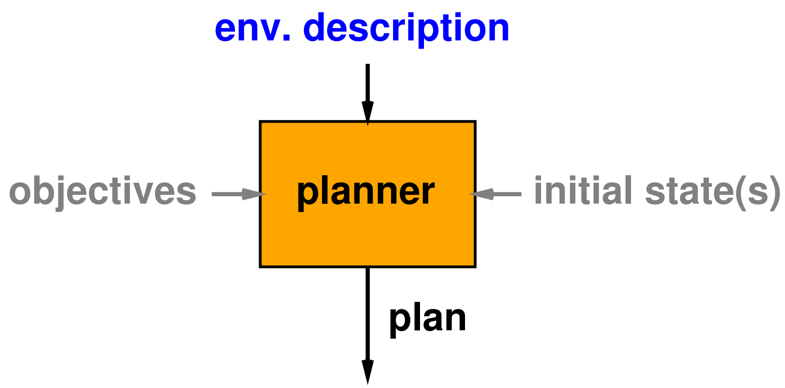
Application examples:

- spacecraft flying (NASA)
- Mars rover control (NASA)
- power supply restoration (EDF)
- elevator control (Shindler, Rockwell)
- sheet-metal bending (Amada)
- operations planning (DSTO-NICTA)
- bridge playing (University of Maryland)

Planning agents



Domain-independent planning



We often seek to build domain-independent planners taking both the environment description and the problem description (initial state, objectives) as input in a suitable language.

Planner = solver over a class of planning models

Other examples of solvers: linear equations solvers, linear programming solvers, SAT solvers, constraint programming solvers, etc

Classical planning assumptions

- finite: states, actions, observations are finite
- static, single agent: no event outside of the planner's control
- deterministic: unique initial state, unique resulting state
- fully observable: sensors provide all relevant aspects of the current state
- off-line planning: planning decoupled from execution
- implicit time: no durations, instantaneous actions
- sequential: solution is a sequence of actions
- reachability goals: acceptable sequences end in a goal state
- cost function: length or path-cost of the sequence

Classical planning model

- a finite set of states S
- a finite set of actions A
- a transition function $\gamma : S \times A \mapsto S$
- an initial state s_0
- a set S_G of goal states
- a (step) cost function $c : A \mapsto \mathsf{R}^+$

6 components of classical planning model

Note: if action a is not applicable in state s then $\gamma(s, a)$ is undefined

simply b/c s cannot be leed to any state by action a.

state

Classical planning problem

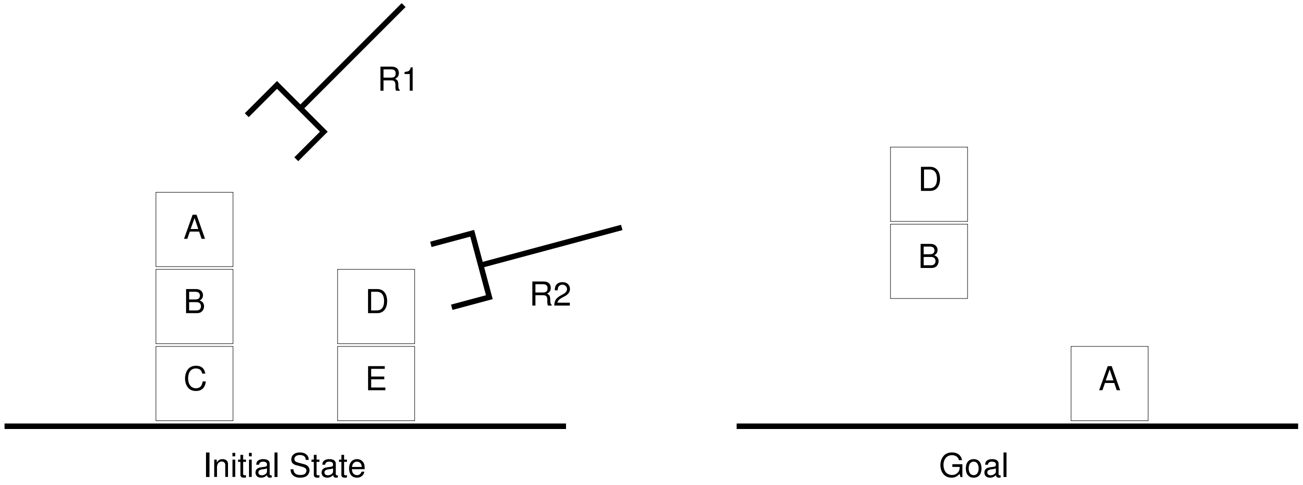
Given a planning model $(S, A, \gamma, s_0, S_G, c)$, find a sequence of actions $\langle a_1, a_2, \ldots, a_n \rangle$, $a_i \in A$, leading the environment from the initial state s_0 to a goal state in S_G (at minimum cost).

That is, the sequence of actions must induce a sequence of state transitions:

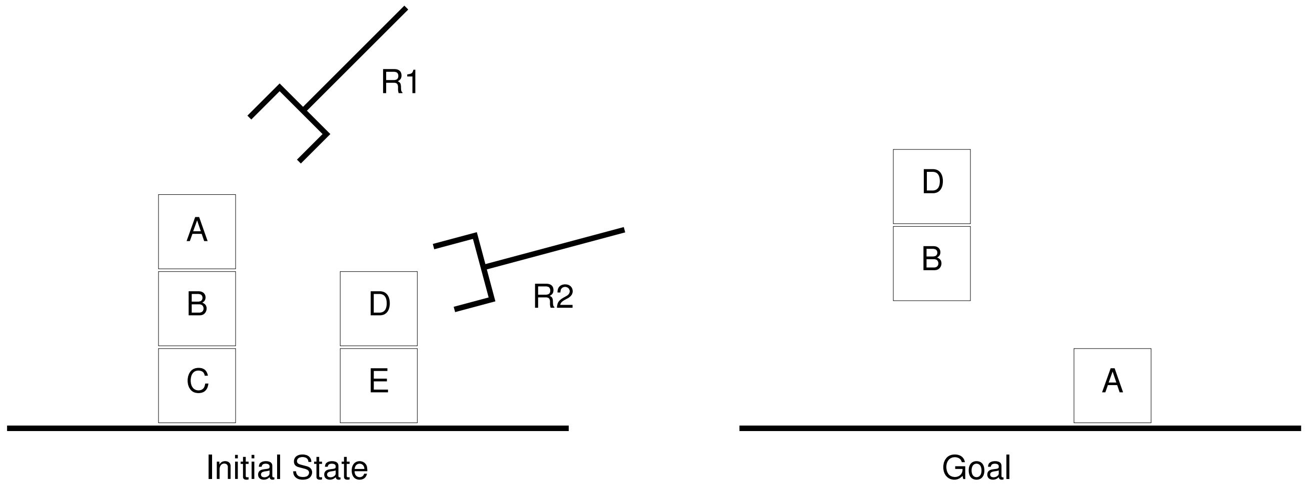
$$
\begin{aligned}
&s_1 = \gamma(s_0, a_1) \\
&s_2 = \gamma(s_1, a_2) \\
&\ldots \\
&s_i = \gamma(s_{i-1}, a_i) \\
&\ldots \\
&s_n = \gamma(s_{n-1}, a_n) \in S_G
\end{aligned}
$$

where γ is defined at each step (and $\underbrace{\sum_{i=1}^{n} c(a_i)}_{\text{total step cost.}}$ is minimal)

Example: blocks world



Example: blocks world



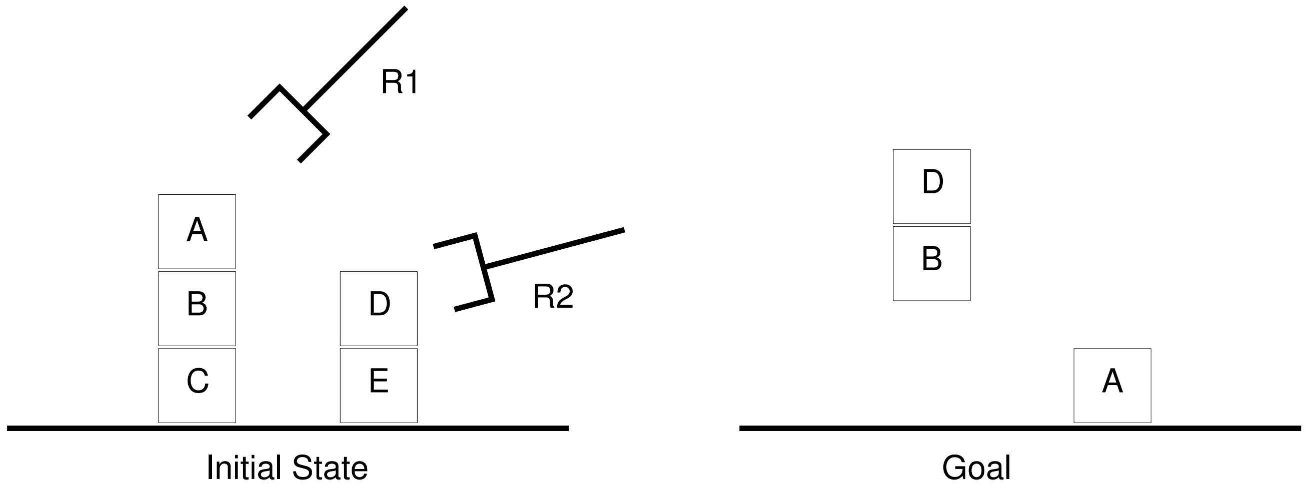
states??: configurations of n blocks, i.e., the data of the object on which each given block is, if any, and the block held by each given robot, if any.

actions??: robot picks up clear block from table,
robot puts down held block onto the table,
robot unstacks clear block from top of another block,
robot stacks held block on top of another clear block.

initial state??: given configuration goal??: given (partial) configuration

cost??: 1 per action

Example: blocks world



plan??:
$\langle$unstack(R1, A, B), unstack(R2, D, E), putdown(R1, A), stack(R2, D, B)$\rangle$

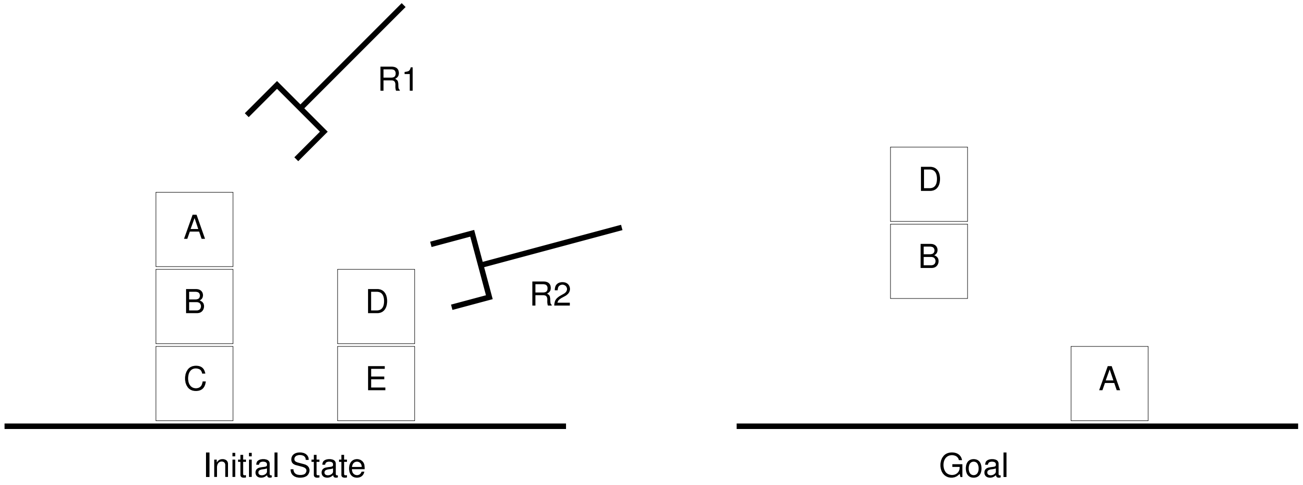
Classical plans

linear plan (or sequence): totally ordered set $\langle a_1, \ldots, a_n \rangle$, $a_i \in A$, such that $\gamma(\ldots\gamma(\gamma(s_0, a_1), a_2), \ldots, a_n) \in S_G$. Produced by state-space planning approaches.

non-linear plan: partially ordered set $\langle \{a_1, \ldots, a_n\}, < \rangle$, $a_i \in A$, such that each linearisation is a valid linear plan. More flexible for execution. Produced by plan-space planning approaches.

parallel plan: sequence of parallel action sets $\langle \{a_{1,1}, \ldots, a_{1,l(1)}\}, \ldots, \{a_{1,n}, \ldots, a_{1,l(n)}\} \rangle$, $a_{i,j} \in A$. Actions in each set must not *interfere*: performing them in any order or in parallel must lead to the same result. Produced by sat-based and graph-based planning approaches.

Example: blocks world



linear plan??:
$\langle$unstack(R1, A, B), unstack(R2, D, E), putdown(R1, A), stack(R2, D, B)$\rangle$

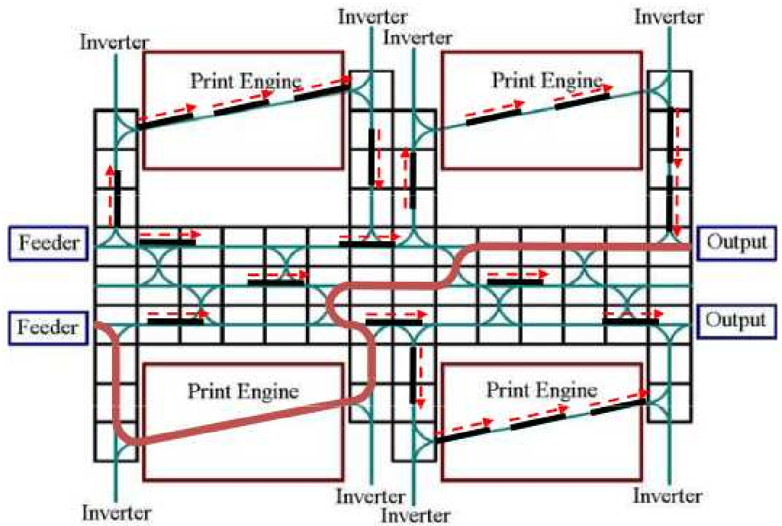
non-linear plan??:
$\langle\{$unstack(R1, A, B), unstack(R2, D, E), putdown(R1, A), stack(R2, D, B)$\}$,
$\{$unstack(R1, A, B) $<$ putdown(R1, A), unstack(R2, D, E) $<$ stack(R2, D, B),
unstack(R1, A, B) $<$ stack(R2, D, B)$\}\rangle$

parallel plan??:
$\langle\{$unstack(R1, A, B), unstack(R2, D, E)$\}$, $\{$putdown(R1, A), stack(R2, D, B)$\}\rangle$

Real-world examples

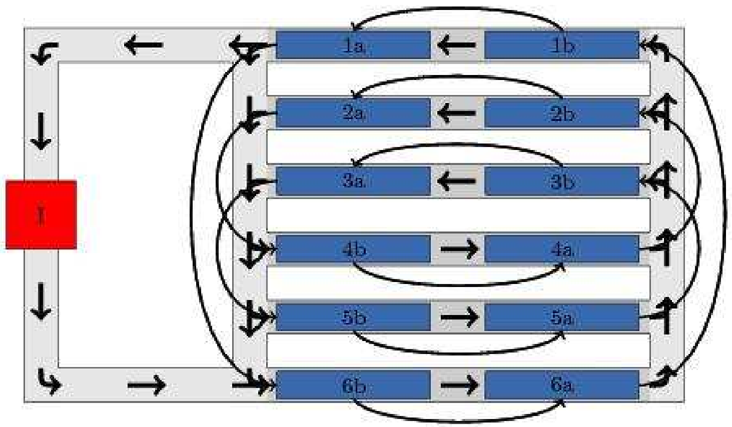
Elevator Control (Koelher and Schuster, ICAPS 2000)

Real-world examples



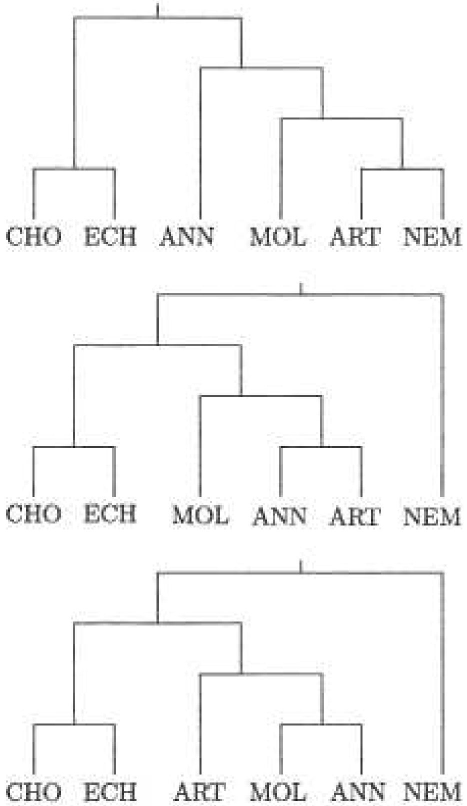
Modular Printers Control (Do et. al ICAPS 2008)

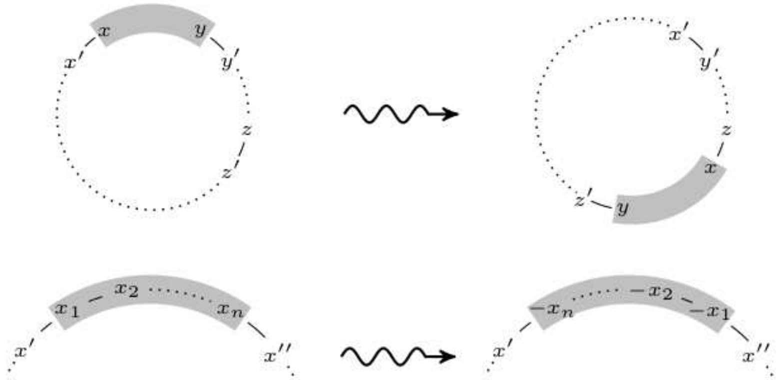
Real-world examples



Greenhouse Logistics (Helmert & Lasinger, ICAPS 2010)

Real-world examples





Genome Edit Distance (Uras & Erdem, ICAPS 2010)

Why do we need representations?

A classical planning model $(S, A, \gamma, s_0, S_G, c)$ can directly be solved by search

However, we cannot use search naively:

- most planning problems are too large to be explicitly described e.g. blocks world with 30 blocks has 197987401295571718915006598239796851 states!
- problems with a large branching factor would require human-supplied heuristics, defeating the goal of building domain-independent planners

We rely on adequate **representations** of the planning problem, enabling

- **consise** problem descriptions
- exploiting the **structure** of the problem (e.g. to derive good domain-independent heuristics)
- scaling to large problems whilst maintaining **domain-independence**

The STRIPS representation - states

- Use fragment of first-order logic to represent states:
  - logical language (predicates, connectives, variables, quantifiers, finite object set, no function)
  - a property of states (a set $S' \subseteq S$) is represented by a formula

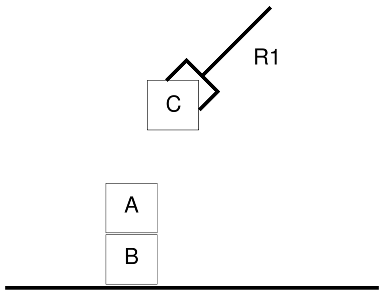
    $\forall x(\mathsf{block}(x) \rightarrow \mathsf{ontable}(x) \vee \exists r\ \mathsf{robot}(r) \wedge \mathsf{holding}(r, x))$

    all blocks are on the table or held by some robot
  - the goal is often represented by a set of ground atoms for simplicity

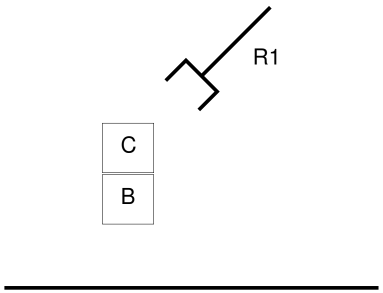
    $\{\mathsf{on}(\mathsf{C}, \mathsf{B}), \mathsf{handempty}(\mathsf{R1})\}$

  - a state $s \in S$ is represented by a set of ground atoms under the closed world assumption

    $\{\mathsf{on}(\mathsf{A}, \mathsf{B}), \mathsf{clear}(\mathsf{A}), \mathsf{ontable}(\mathsf{B}), \mathsf{holding}(\mathsf{R1}, \mathsf{C})\}$



Initial State



Goal

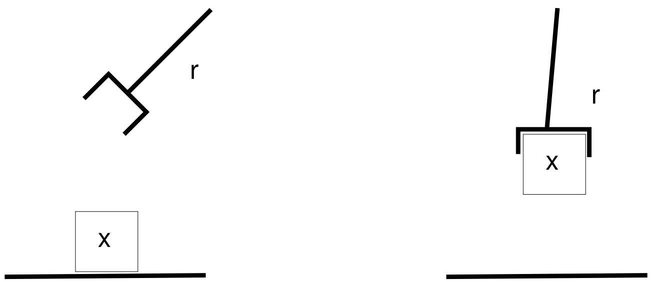
The STRIPS representation - actions

- Use operators with logical pre-post conditions to represent actions:
  - operator o has a name and parameters: pickup(r, x)
  - precondition $\text{PRE}(o)$ is a set of positive literals that must be true for the action to be applicable: $\{$ontable(x), clear(x), handempty$(r)\}$
  - effect (postcondition) $\text{EFF}(o)$ is a set of literals that change in the resulting state: $\{$holding(r, x), $\neg$ontable(x), $\neg$clear(x), $\neg$handempty$(r)\}$
  - the effect is often split into two sets of *positive* literals:

    add list $\text{EFF}^{+}(o) = \{$holding$(r, x)\}$

    delete list $\text{EFF}^{-}(o) = \{$ontable(x), clear(x), handempty$(r)\}$
  - an action $a \in A$ is represented by an instance of an operator

    e.g. pickup(R1, C).



The STRIPS representation - transition

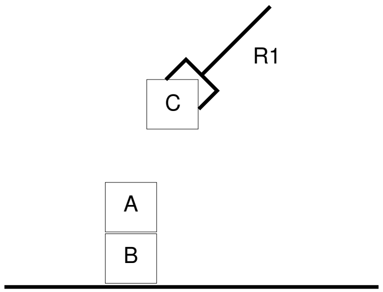
- Use the STRIPS rule to represent the transition function γ:

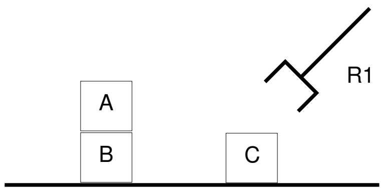
$$\gamma(s,a) = \begin{cases} (s \setminus \text{EFF}^-(a)) \cup \text{EFF}^+(a) & \text{if } \text{PRE}(a) \subseteq s \\ \text{undefined otherwise} & \text{(action not executable)} \end{cases}$$

  Assumption of inertia: atoms not affected by the action keep their value

- Example:
  - $s = \{\text{on}(\text{A},\text{B}), \text{clear}(\text{A}), \text{ontable}(\text{B}), \text{holding}(\text{R1},\text{C})\}$
  - $a = \text{putdown}(\text{R1},\text{C})$

    operator putdown(r,x)
    precondition $\{\text{holding}(r,x)\}$
    effect $\{\text{ontable}(x), \text{clear}(x), \text{handempty}(r), \neg\text{holding}(r,x)\}$
  - $\gamma(s,a) = \{\text{on}(\text{A},\text{B}), \text{clear}(\text{A}), \text{ontable}(\text{B}), \text{ontable}(\text{C}), \text{clear}(\text{C}), \text{handempty}(\text{R1})\}$





The STRIPS representation - transition

- Use the STRIPS rule to represent the transition function γ:

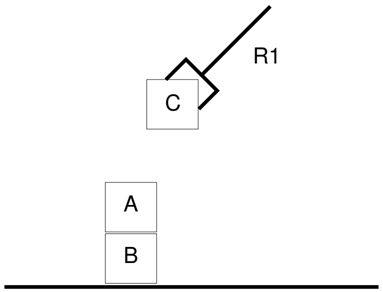
$$\gamma(s,a) = \begin{cases} (s \setminus \text{EFF}^-(a)) \cup \text{EFF}^+(a) & \text{if } \text{PRE}(a) \subseteq s \\ \text{undefined otherwise} & \text{(action not executable)} \end{cases}$$

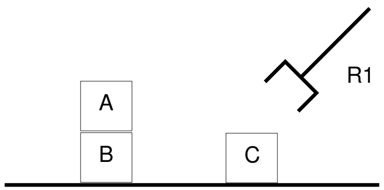
Assumption of inertia: atoms not affected by the action keep their value

- Example:
  - $s = \{\text{on}(\text{A},\text{B}), \text{clear}(\text{A}), \text{ontable}(\text{B}), \text{holding}(\text{R1},\text{C})\}$
  - $a = \text{putdown}(\text{R1},\text{C})$

operator	$\text{putdown}(r,x)$
precondition	$\{\text{holding}(\text{R1},\text{C})\}$
effect	$\{\text{ontable}(\text{C}), \text{clear}(\text{C}), \text{handempty}(\text{R1}), \neg\text{holding}(\text{R1},\text{C})\}$

  - $\gamma(s,a) = \{\text{on}(\text{A},\text{B}), \text{clear}(\text{A}), \text{ontable}(\text{B}), \text{ontable}(\text{C}), \text{clear}(\text{C}), \text{handempty}(\text{R1})\}$





PDDL

PDDL is the standard Planning Domain Definition Language

It is used in benchmarking planners and in the International Planning Competition series

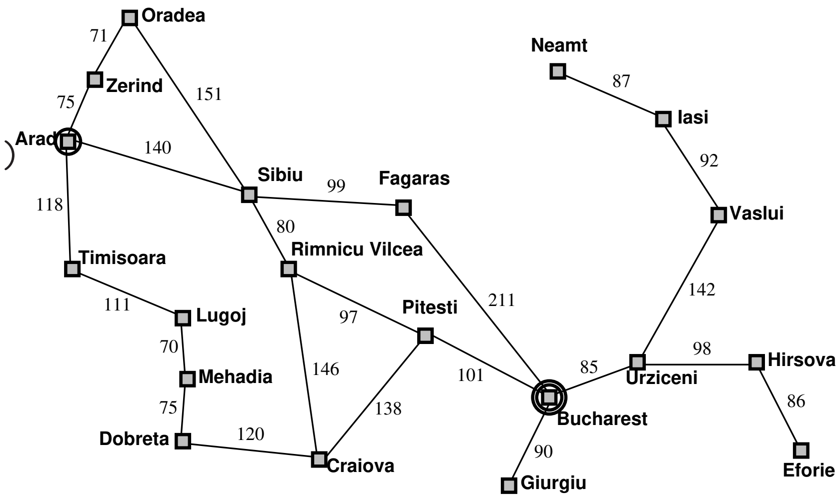
It supports STRIPS and many extensions:

- ADL – mostly syntactic extension
- planning with multi-valued variables and numeric variables
- temporal planning
- planning with temporally extended goals
- planning with continuous variables and processes
- non-deterministic and probabilistic planning
- planning under partial observability

Short PDDL tutorial: `http://users.cecs.anu.edu.au/~patrik/pddlman/writing.html`

PDDL - example

```
(define (domain travel)
  (:requirements :strips)

  (:predicates
    (in ?city)
    (road ?c1 ?c2))

  (:action go
    :parameters (?from ?to)
    :precondition (and (in ?from) (road ?from ?to))
    :effect (and (not (in ?from)) (in ?to))))

(define (problem romania)
  (:domain travel)
  (:objects Arad Bucharest ... Zerind)

  (:init
    (in Arad)
    (road Arad Sibiu) (road Sibiu Arad)
    (road Arad Timisoara) (road Timisoara Arad)
    ...
    (road Zerind Oradea) (road Oradea Zerind))

  (:goal (in Bucharest)))
```



PDDL - minimizing costs

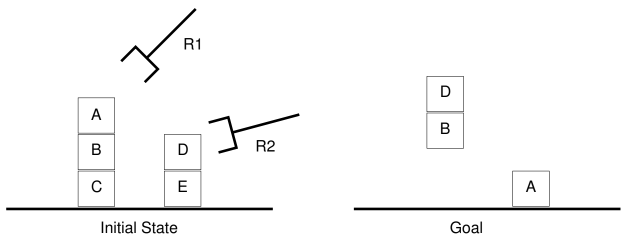
```
(define (domain travel)
  (:requirements :strips :fluents)
...
  (:functions (distance ?c1 ?c2) (total-cost))

  (:action go
    :parameters (?from ?to)
    :precondition (and (in ?to) (road ?from ?to))
    :effect (and (not (in ?from)) (in ?to)
               (increase (total-cost) (distance ?from ?to))))

(define (problem romania)
...
  (:init
    (= (distance Arad Sibiu) 140)     (= (distance Sibiu Arad) 140)
    (= (distance Arad Timisoara) 118) (= (distance Timisoara Arad) 118))
...
  (:goal (in Bucharest)
   :metric minimize (total-cost)))
```

PDDL - blocks world

```
(define (domain blocksworld)
  (:requirements :strips :typing)
  (:types robot block)

(:predicates (clear ?x - block)
             (on-table ?x - block )
             (handempty ?r - robot)
             (holding ?r - robot ?x -block)
             (on ?x ?y - block))

(:action pickup
  :parameters (?r - robot ?x - block)
  :precondition (and (clear ?x) (on-table ?x) (handempty ?r))
  :effect (and (holding ?r ?x) (not (clear ?x)) (not (on-table ?x))
               (not (handempty ?r))))

(:action putdown
  :parameters  (?r - robot ?x - block)
  :precondition (holding ?r ?x)
  :effect (and (clear ?x) (handempty ?r) (on-table ?x)
               (not (holding ?x))))
```

PDDL - blocks world

```
(:action stack
  :parameters  (?r - robot ?x ?y - block)
  :precondition (and (clear ?y) (holding ?r ?x))
  :effect (and (handempty ?r) (clear ?x) (on ?x ?y)
               (not (clear ?y)) (not (holding ?r ?x))))

(:action unstack
  :parameters  (?r - robot ?x ?y - block)
  :precondition (and (on ?x ?y) (clear ?x) (handempty ?r))
  :effect (and (holding ?x) (clear ?y)
               (not (on ?x ?y)) (not (clear ?x)) (not (handempty ?r))))

(define (problem small-bw-problem)
  (:domain blocksworld)
  (:objects a b c d e - block r1 r2 - robot)

  (:init
    (ontable c) (ontable e) (on b c) (on a b) (on d e)
    (clear a) (clear d) (handempty r1) (handempty r2))

  (:goal (and (on d b) (ontable a))))
```



The ADL representation

STRIPS	ADL
Only positive literals in states closed world assumption unmentioned literals are false $\{\text{Poor}, \text{Unknown}\}$	Positive and negative literals in states open world assumption unmentioned literals are unknown $\{\neg\text{Rich}, \neg\text{Famous}\}$
Effect $\{\text{P}, \neg\text{Q}\}$ means add P delete Q	Effect $\{\text{P}, \neg\text{Q}\}$ means add P and ¬Q, delete Q and ¬P
No support for equality and types	Equality predicate (x=y) built in Variables may have types: pickup$(r : \text{robot}, x : \text{block})$
Only positive literals in prec. & goals $\{\text{Rich}, \text{Famous}\}$	Prec. & goals are arbitrary formulae $\forall t(\exists f\ \text{Booked}(t, f)) \Rightarrow \text{At}(t, \text{Bucharest})$
Effects are sets (conjunctions)	Conditional & univ. quantified effects **when** $C : E$ **forall** x $\text{Q}(x)$ E takes place only when C is true

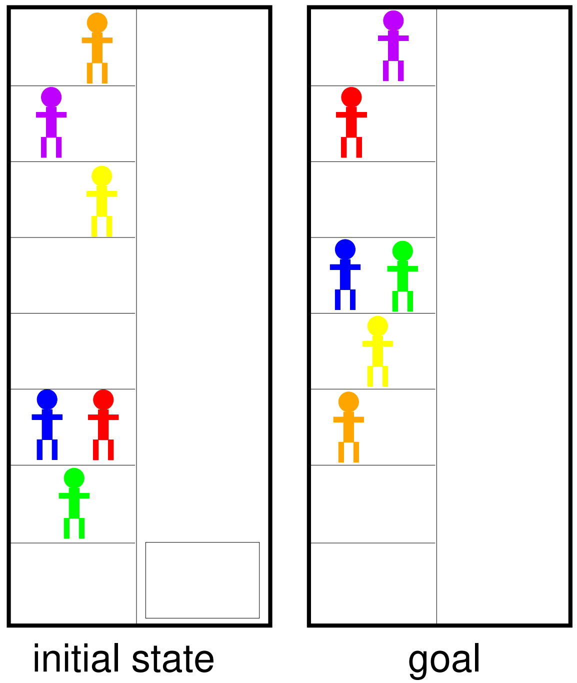
ADL features can be compiled into STRIPS, but some create exponential space domain increase or linear plan length increase

PDDL - elevator

```
(define (domain elevator)
  (:requirements :adl)
  (:types passenger floor)

  (:predicates
    (origin ?p - passenger ?f - floor)
    (destin ?p - passenger ?f - floor)
    (boarded ?p - passenger)
    (served ?p - passenger)
    (lift-at ?f - floor))

  (:action go
    :parameters (?fa ?fb - floor)
    :precondition (lift-at ?fa)
    :effect (and (lift-at ?fb) (not (lift-at ?fa))
                 (forall (?p - passenger)
                    (when (and (boarded ?p) (destin ?p ?fb))
                          (and (not (boarded ?p)) (served  ?p))))
                 (forall (?p - passenger)
                     (when (and (origin ?p ?fb) (not (served ?p)))
                                (boarded ?p))))))
```

PDDL - elevator

```
(define (problem simple)
  (:domain elevator)
  (:objects
     blue red green orange yellow purple - passenger
     f0 f1 f2 f3 f4 f5 f6 f7 - floor)

  (:init
    (origin blue f2)   (destin blue f4)
    (origin red f2)    (destin red f6)
    (origin green f1)  (destin green f4)
    (origin orange f7) (destin orange f2)
    (origin yellow f5) (destin yellow f3)
    (origin purple f6) (destin purple f7)
    (lift-at f0))

  (:goal (forall (?p - passenger) (served ?p))))
```



Plan:

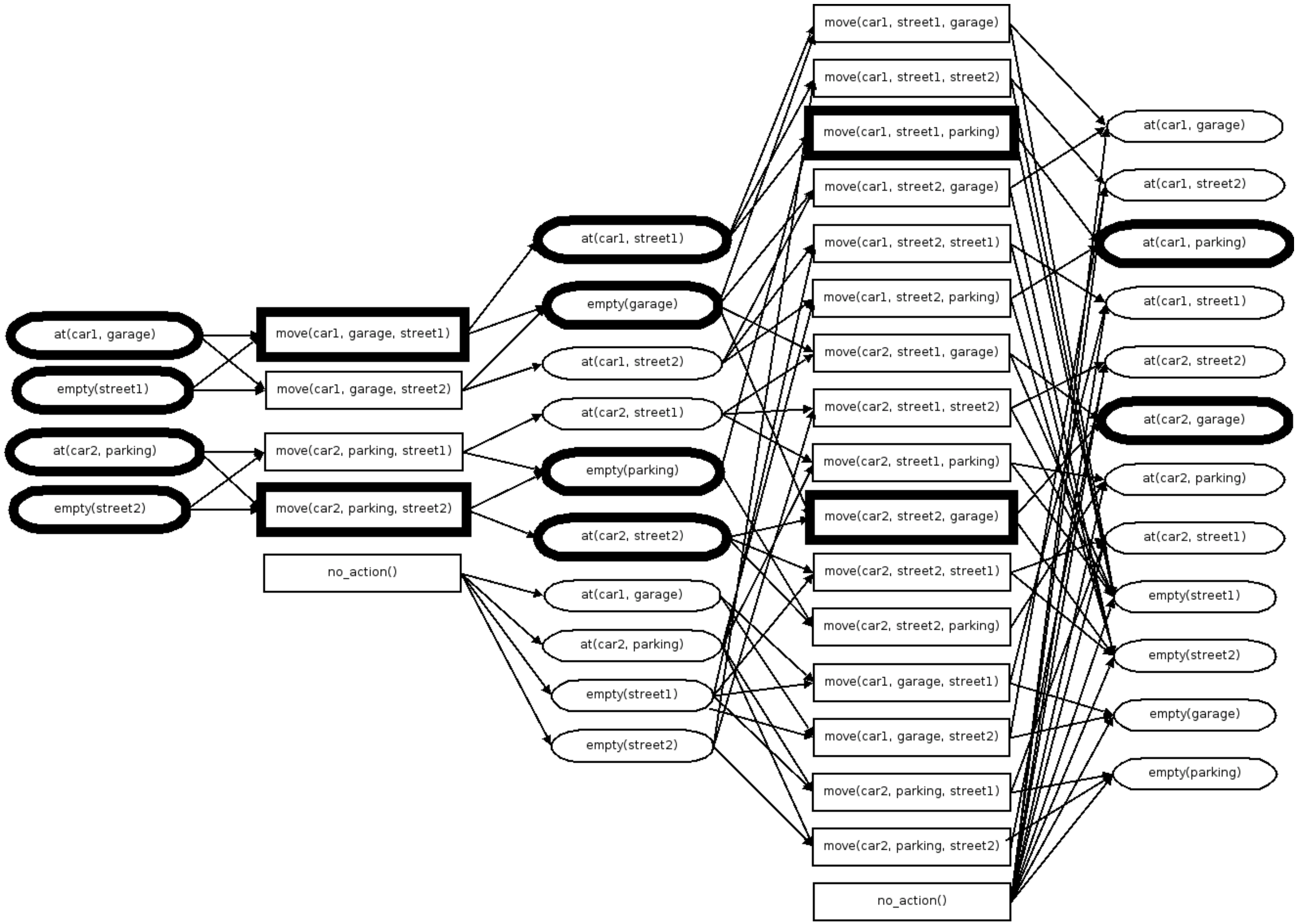
(go f0 f1) (go f1 f2) (go f2 f4)
(go f4 f5) (go f5 f6) (go f6 f7)
(go f7 f3) (go f3 f2)

Complexity of propositional STRIPS planning

Propositional STRIPS Planning: all predicates and operators have been instanciated (grounded). Recall that for STRIPS, preconditions are positive.

- n propositions can result in 2^n states; in the worst case, the shortest plan will visit them all and is exponentially long ($2^n - 1$ actions)
- PLANSAT: Does there exist a plan that solves the problem?
  PSPACE complete. Polynomial if all effects are positive
- PLANMIN: Does there exist a plan of length k or less?
  Also PSPACE complete. NP-complete if all effects are positive
- both are NP-complete if the plan length is polynomially bounded

PLANSAT without delete lists is polynomial



Complexity of STRIPS planning

We consider STRIPS in its first-order (a.k.a. lifted) form.

- n predicates with k arguments and m objects can give up to nm^k atomic propositions
- these can give 2^{nm^k} states
- in the worst case, the shortest plan will visit all of them in $2^{nm^k} - 1$ actions

PLANSAT is EXPSPACE-complete!

Summary

Planning is the reasoning side of acting. Planning = Search + KR.

Classical planning is an off-line process which assumes a single agent and a static environment, determinism, full observability, reachability goals, and ignores qantitative time.

The STRIPS representation uses a logical language to represent properties of states; actions are represented by their preconditions and add/delete effects

STRIPS enables algorithms to exploit the structure of the problem. ADL is a useful extension of STRIPS. PDDL supports both and many extensions

Propositional STRIPS planning is PSPACE-complete

Planning algorithms differ by the search space they explore and the type of classical plan they produce: ① sequential, ② partially ordered, ③ or parallel plan.