# AUTOMATED PLANNING AND ACTING

Malik Ghallab, Dana Nau and Paolo Traverso

Cambridge University Press, 2016

# WHAT IS MEANT BY "PLANNING ALGORITHMS"?

**robotics** *how to move robot from one place to another without hitting anything*
focus on algorithms that generate useful motions by processing complicated geometric models

**AI** *search for a sequence of logical operators / actions that transform an initial world state into a desired goal state*
focus on designing systems that use decision-theoretic models co compute appropriate actions

**control theory** *feasible trajectories for nonlinear systems*
focus on algorithms that compute feasible trajectories for systems, with some additional coverage of feedback and optimality

# ROBOT MOTION PLANNING TASKS

- automotive assembly task
- moving furniture
- navigating mobile robots

# INTRODUCTION

# TERMINOLOGY

agent entity capable of interacting with its environment

action something that an agent does, such as exerting a force, a motion, a perception or a communication

deliberation deciding which actions to undertake and how to perform them to achieve an objective

**artificial agent $\equiv$ actor**

autonomy the actor performs its intended functions without being directly operated by a person

diversity in the tasks it can perform and the environment in which it can operate

# CONCEPTUAL VIEW OF AN ACTOR

Deliberation consists of reasoning with predictive models as well as *acquiring* these models.

An actor may have to *learn* how to adapt to new situations and tasks.

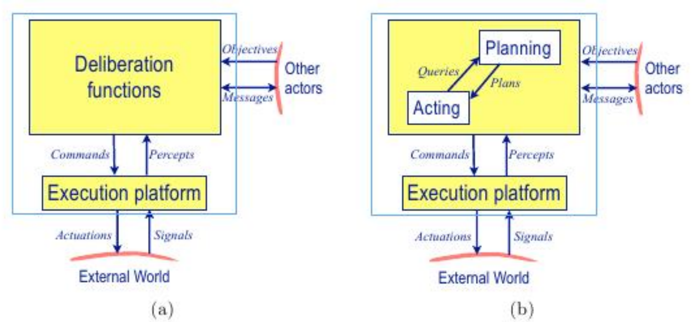



Figure 1.1: Conceptual view of an actor (a); its restriction to planning and acting (b).

# HIERARCHICAL AND CONTINUAL ONLINE DELIBERATION

**hierarchically organized deliberation** some of the actions the actor wishes to perform do not map directly into a command executable by its platform

**continual online deliberation** throughout the acting process, the actor refines and monitors its actions; reacts to events; and extends, updates, and repairs its plan on the basis of its perception focused on the relevant part of the environment

# PLANNING VERSUS ACTING

planning the purpose is to synthesize and organized set of actions to carry out some activity

acting involves deciding how to perform the chosen actions

**receding horizon scheme** - in dynamic environments where exougenous events can take place and are difficult to model and predict; first steps are usually more reliable; plan modification and replanning

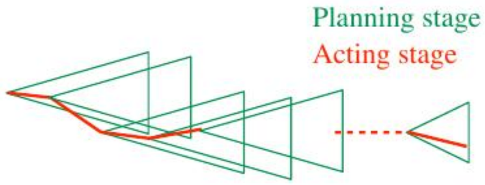



Figure 1.4: Receding horizon scheme for planning and acting.

# ASSUMPTIONS

Deliberation assumptions are usually about how variable, dynamic, observable, and predictable the environment is, and what the actor knows and perceives about it while acting.

*different chapters of the book make different assumptions about time, concurrency, and uncertainty and we'll restrict ourself to discrete approaches*

# DESCRIPTIVE AND OPERATIONAL MODELS OF ACTIONS

descriptive models describe which state or set of possible states may result from perforing an action; they are used by the actor to reason about what actions may achieve the objectives

operational models describe how to perform an action, that is, what commands to execute in the current context

# DESCRIPTION OF STATES FOR DELIBERATION

representational primitives that define the state of an actor and its environment $\equiv$ state variables

predicted states used when an actor reasons about what might happen

observed states used when an actor reasons about how to perform actions in some context

*predicted states are in general less detailed than the observed one*

# DELIBERATION WITH DETERMINISTIC MODELS

# DELIBERATION WITH DETERMINISTIC MODELS

## STATE-VARIABLE REPRESENTATION

# STATE-TRANSITION SYSTEM

a *state-transition system* (aka classical planning domain) is a triple $\Sigma = (S, A, \gamma, cost)$ (states, actions, state transition function, cost function)

- finite, static environment
- no explicit time, no concurenncy
- determinism, no uncertainty

computational aspects of using state-transition system

- if $S$ and $A$ are small enough — lookup table
- otherwise — generative representation in which there are procedures for computing $\gamma(s, a)$ given $s$ and $a$
  - domain-specific representation
  - domain-independent representation

# EXAMPLE

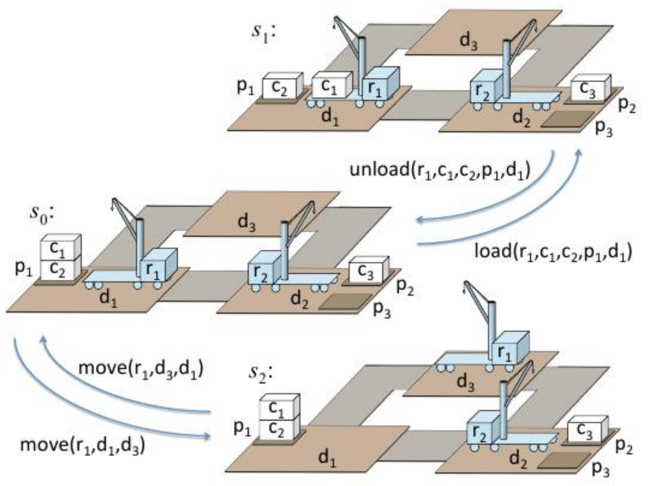



Figure 2.3: A few of the states and transitions in a simple state-transition system. Each robot can hold at most one container, and at most one robot can be at each loading dock.

# OBJECTS AND STATE VARIABLES

objects set of names

$r_1, r_2, d_1, \ldots, p_3$

state variable syntactic term over objects

*pos(c) is containers c's position, which can be a robot, another container, or nil if c is at the bottom of a pile*

variable-assignement function maps objects to values

$$
\begin{aligned}
s_0 = \{ & \mathsf{cargo}(\mathsf{r}_1) = \mathsf{nil}, & & \mathsf{cargo}(\mathsf{r}_2) = \mathsf{nil}, & & \\
& \mathsf{loc}(\mathsf{r}_1) = \mathsf{d}_1, & & \mathsf{loc}(\mathsf{r}_2) = \mathsf{d}_2, & & \\
& \mathsf{occupied}(\mathsf{d}_1) = \mathsf{T}, & & \mathsf{occupied}(\mathsf{d}_2) = \mathsf{T}, & & \mathsf{occupied}(\mathsf{d}_3) = \mathsf{F}, \\
& \mathsf{pile}(\mathsf{c}_1) = \mathsf{p}_1, & & \mathsf{pile}(\mathsf{c}_2) = \mathsf{p}_1, & & \mathsf{pile}(\mathsf{c}_3) = \mathsf{p}_2, \\
& \mathsf{pos}(\mathsf{c}_1) = \mathsf{c}_2, & & \mathsf{pos}(\mathsf{c}_2) = \mathsf{nil}, & & \mathsf{pos}(\mathsf{c}_3) = \mathsf{nil}, \\
& \mathsf{top}(\mathsf{p}_1) = \mathsf{c}_1, & & \mathsf{top}(\mathsf{p}_2) = \mathsf{c}_3, & & \mathsf{top}(\mathsf{p}_3) = \mathsf{nil}\}.
\end{aligned}
\tag{2.4}
$$

state-variable state space specified with consistency constraints

*not all combinations are possible*

if *r* is at a loading dock and is not already carrying anything , *r* can load a container form the top o a pile

$\mathsf{load}(r, c, c', p, d)$
pre: $\mathsf{at}(p, d)$, $\mathsf{cargo}(r) = \mathsf{nil}$, $\mathsf{loc}(r) = d$, $\mathsf{pos}(c) = c'$, $\mathsf{top}(p) = c$
eff: $\mathsf{cargo}(r) = c$, $\mathsf{pile}(c) \leftarrow \mathsf{nil}$, $\mathsf{pos}(c) \leftarrow r$, $\mathsf{top}(p) \leftarrow c'$

Let $a_1$ be the state-variable action $\mathsf{load}(\mathsf{r}_1, \mathsf{c}_1, \mathsf{c}_2, \mathsf{p}_1, \mathsf{d}_1)$. Then

$$\mathrm{pre}(a_1) = \{\mathsf{at}(\mathsf{p}_1, \mathsf{d}_1), \mathsf{cargo}(\mathsf{r}_1) = \mathsf{nil}, \mathsf{loc}(\mathsf{r}_1) = \mathsf{d}_1, \mathsf{pos}(\mathsf{c}_1) = \mathsf{c}_2, \mathsf{top}(\mathsf{p}_1) = \mathsf{c}_1\}.$$

# PLANS

- a *plan* $\pi$ is a finite sequence of actions, $\pi = \langle a_1, \ldots a_n \rangle$
- *planning problem* is a triple $P = (\Sigma, s_0, g)$, where $\Sigma$ is a state-variable palnning domain, $s_0$ is an initial state, and $g$ is a goal
- each node is written as a pair $v = (\pi, s)$ where $\pi$ is a plan and $s = \gamma(s_0, \pi)$

# DELIBERATION WITH DETERMINISTIC MODELS

## FORWARD STATE-SPACE SEARCH

# FORWARD-SEARCH

```
Forward-search (Σ, s₀, g)
    s ← s₀;  π ← ⟨⟩
    loop
        if s satisfies g, then return π
        A' ← {a ∈ A | a is applicable in s}
        if A' = ∅, then return failure
        nondeterministically choose a ∈ A'    (i)
        s ← γ(s, a);  π ← π.a
```

Algorithm 2.1: Forward-search planning schema.

# DETERMINISTIC FORWARD-SEARCH

```
Deterministic-Search(Σ, s₀, g)
    Frontier ← {(⟨⟩, s₀)}        // (⟨⟩, s₀) is the initial node
    Expanded ← ∅
    while Frontier ≠ ∅ do
        select a node ν = (π, s) ∈ Frontier                    (i)
        remove ν from Frontier and add it to Expanded
        if s satisfies g then                                  (ii)
            return π
        Children ← {(π.a, γ(s, a)) | s satisfies pre(a)}
        prune (i.e., remove and discard) 0 or more nodes
            from Children, Frontier and Expanded               (iii)
        Frontier ← Frontier ∪ Children                         (iv)
    return failure
```

Algorithm 2.2: Deterministic-Search, a deterministic version of Forward-search.

step (i) heuristic function which estimates the minimum cost of getting from $s$ to a goal state

step (iii) remove from *Children* every $(\pi, s)$ that has an ancestor $(\pi', s')$ s.t. $s = s'$

node selection select a node $(\pi, s) \in \mathit{Children}$ that minimizes the length of $\pi$

pruning remove from *Frontier* and *Children* every node $(\pi, s)$ such that *Expanded* contains $(\pi', s)$

## DEPTH-FIRST SEARCH

**node selection** select a node $(\pi, s) \in \mathit{Children}$ that maximizes the length of $\pi$

**pruning** First do cycle-checking.
Then, to eliminate nodes that the algorithm is done with, remove $v$ from *Expanded* if it has no children in $\mathit{Frontier} \cup \mathit{Expanded}$, and do so with each of $v$'s ancestors until no more nodes are removed.

# HILL CLIMBING (GREEDY SEARCH)

depth-first search with no backtracking

node selection select a node $(\pi, s) \in \mathit{Children}$ that minimizes $h(s)$

pruning First do cycle-checking.

Then, assign $\mathit{Frontier} \leftarrow \emptyset$ so that the line (iv) of Algorithm 2.2 be the same as assigning $\mathit{Frontier} \leftarrow \mathit{Children}$

- heuristic function $h\colon S \to \mathbb{R}$ estimates minimum cost of getting from $s$ to a goal state

$$h(s) \approx h^*(s) = \min\{\, \mathrm{cost}(\pi) \mid \gamma(s, \pi) \text{ satisfies } g\}$$

- $h$ is admissible if $0 \leq h(s) \leq h^*(s)$ for every $s$

*no guarantee to return an optimal solution or even a solution at all*

# UNIFORM-COST SEARCH

aka Least-cost first

like breadth-first search, uniform-cost search does not use a heuristic function
unlike breadth-first search, it does node selection using the accumulated cost of each node

node selection
: select a node $(\pi, s) \in \mathit{Children}$ that minimizes $\mathit{cost}(\pi)$

pruning
: remove from *Frontier* and *Children* every node $(\pi, s)$ such that *Expanded* contains $(\pi', s)$

# $A^*$ SEARCH

$A^*$ search is similar to uniform-cost search, but uses a heuristic function

**node selection** select a node $(\pi, s) \in \mathit{Children}$ that minimizes $\mathit{cost}(\pi) + h(s)$

**pruning** for each node $(\pi, s) \in \mathit{Children}$, if $A^*$ has more than one plan that goes to $s$, then keep only the least costly one

$A^*$ terminates and returns a solution if one exists; if $h$ is admissible, then the solution is optimal

# DEPTH-FIRST BRANCH AND BOUND

- depth-first branch and bound(DFBB) is a modified version of depth-first search that uses a different termination test that the one in line (ii) of Algorithm 2.2.
- instead of returning the first solution if finds, DFBB keeps searching until *Frontier* is empty
- DFBB maintains two variables $\pi^*$ and $c^*$, which are the least costly solution that has been found so far
- node selection and prunning are the same as in DFS
- additional pruning step occurs during node expansion: if the selected node $v$ has $cost(\pi) + h(v) \geq c^*$, DFBB discards $v$ rather than expanding it

# GREEDY BEST-FIRST SEARCH

for planning problems where nonoptimal solutions are acceptable

node selection select a node $(\pi, s) \in \mathit{Children}$ that minimizes $h(s)$

pruning same as in A*

BBFS is not guaranteed to return optimal solutions

# ITERATIVE DEEPENING

for $k = 1$ to $\infty$ do

do a depth-first search, bactracking at every node of depth $k$

if the search found a solution, then return it

if the search generated no nodes of depth $k$, return failure

a closely related algorithm, IDA$^*$, uses a cost bound rather than a depth bound

# CHOOSING A FORWARD-SEARCH ALGORITHM

- if a nonoptimal solution is acceptable — greedy best-first search
- if one has a good (admissibel) heuristic function — A*
- if hte state space is too large to hold in main memory — DFBB, IDA*

## HEURISTIC FUNCTIONS

- heuristic function $h$ returns an estimate $h(s)$ of the minimum cost $h^*(s)$ of getting from the state $s$ to a goal state
- $h$ is admissible if $0 \leq h(s) \leq h^*(s)$ for every state $s$

- the best-known way of producing $h$ is relaxation
- given a planning domain $\Sigma = (S, A, \gamma)$ and planning problem $P = (\Sigma, s_0, g)$, relaxing them means weakening some of the constraints that restrict what the states, actions, and plans are

- max-cost heuristic
- additive-cost heuristic
- delete-relaxation heuristics
- landmark heuristics

# DELIBERATION WITH DETERMINISTIC MODELS

## BACKWARD SEARCH

# BACKWARD SEARCH

Backward-search$(\Sigma, s_0, g_0)$
    $\pi \leftarrow \langle\rangle;\;\; g \leftarrow g_0$ $\quad(i)$
    loop
        if $s_0$ satisfies $g$ then return $\pi$
        $A' \leftarrow \{a \in A \mid a \text{ is relevant for } g\}$
        if $A' = \varnothing$ then return failure
        nondeterministically choose $a \in A'$
        $g \leftarrow \gamma^{-1}(g, a)$ $\quad(ii)$
        $\pi \leftarrow a.\pi$ $\quad(iii)$

Algorithm 2.4: Backward-search planning schema. During each loop iteration, $\pi$ is a plan that achieves $g$ from any state that satisfies $g$.

# DELIBERATION WITH DETERMINISTIC MODELS

## PLAN-SPACE SEARCH

# PLAN-SPACE SEARCH

- planning as a constraint satisfaction problem
- use constraint-satisfaction techniques to produce solutions
- solutions can be more flexible (for example actions are partially ordered)

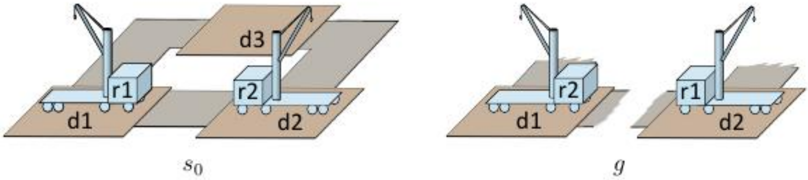
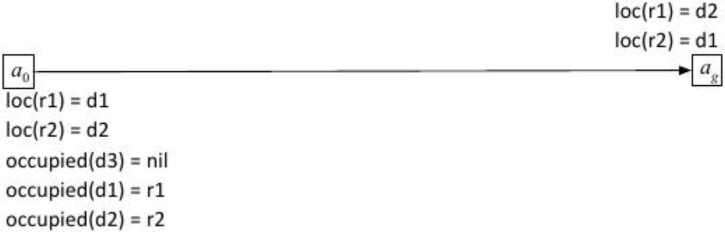



Figure 2.9: Initial state and goal for Example 2.32.



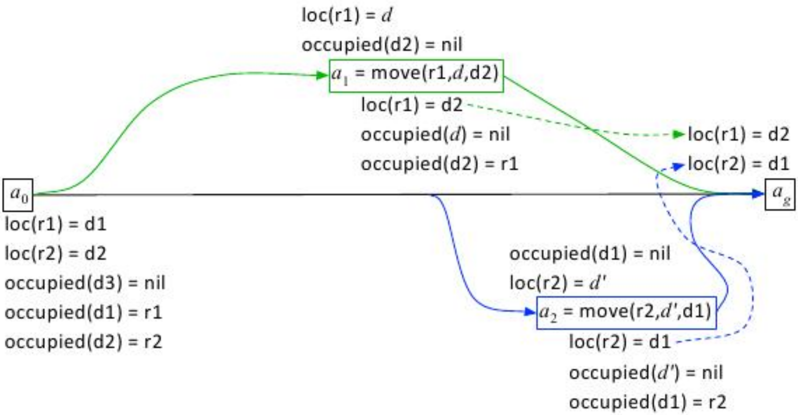



Figure 2.11: Resolving $a_g$'s open-goal flaws. For one of them, PSP adds $a_1$ and a causal link. For the other, PSP adds $a_2$ and another causal link.

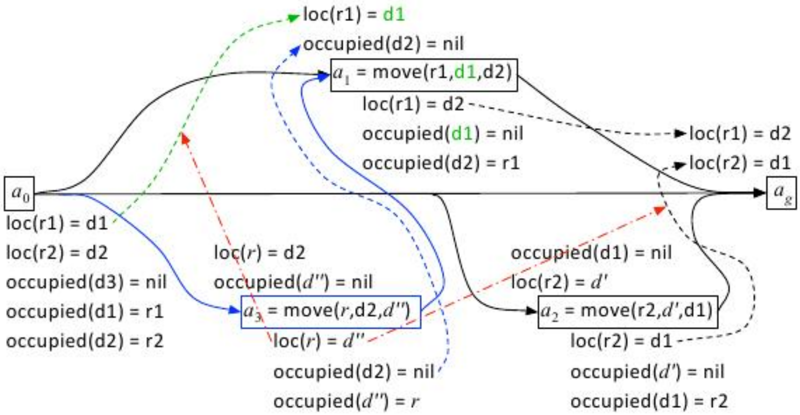



Figure 2.12: Resolving $a_1$'s open-goal flaws. For one of them, PSP substitutes d1 for $d$ (which also resolves $a_1$'s free-variable flaw) and adds a causal link from $a_0$. For the other, PSP adds $a_3$ and a causal link. The new action $a_3$ causes two threats (shown as red dashed-dotted lines).

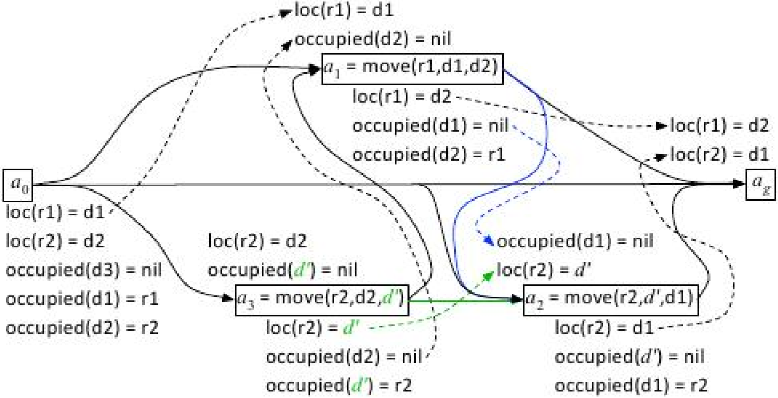



Figure 2.13: Resolving $a_2$'s open-goal flaws. For one of them, PSP substitutes r2 for $r$ and $d'$ for $d''$, and adds a causal link from $a_3$. For the other, PSP adds a causal link from $a_1$. As a side effect, these changes resolve the two threats.

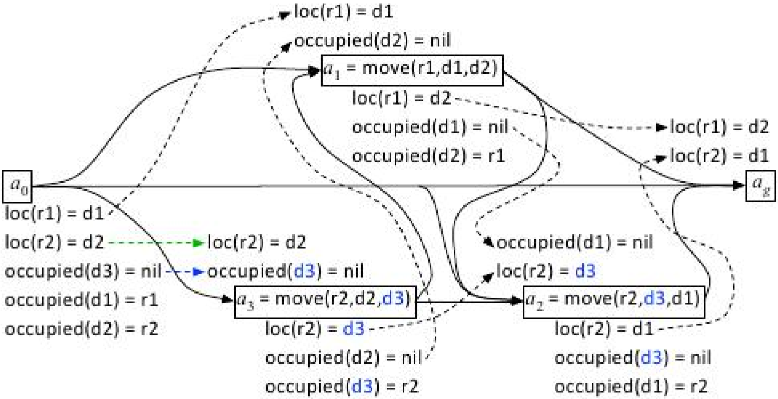



Figure 2.14: Resolving $a_3$'s open-goal flaws. For one of them, PSP adds a causal link. For the other, PSP substitutes d3 for $d'$ and adds a causal link. The resulting partially ordered plan contains no flaws and hence solves the planning problem.

# DELIBERATION WITH REFINEMENT METHODS

operational models

- dynamic environments
- imperfect informations
- overlapping actions
- nondeterminism
- hierarchy
- discrete and continuous variables

# DELIBERATION WITH TEMPORAL MODELS

main motivations for making time explicit in planning and acting

- modeling the duration of actions
- modeling the effects, conditions, and resources borrowed or consumed by an action at various moments along its duration
- handling the concurency of actions
- handling goals with relative or absolute temporal constraints
- planning with actions that maintain a value while being executed

explicit representation of time for the purpose of acting and planning can be

state-oriented keeps notion of global states, time is asocciated with transitions (*timed automata*)

time-oriented dynamics is represented as a collection of partial functions of time, describing local evolutions of state variables instead of a state, the building block is a *timeline*

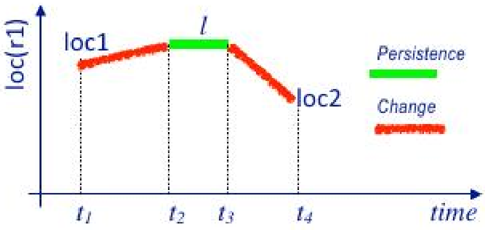



Figure 4.2: A timeline for the state variable loc(r1). The positions of the

r1 leaves loc1 at or after $t_1$, and it arrives at $l$ at or before $t_2$

- consistency
- possibly conflicting
- separation constraint

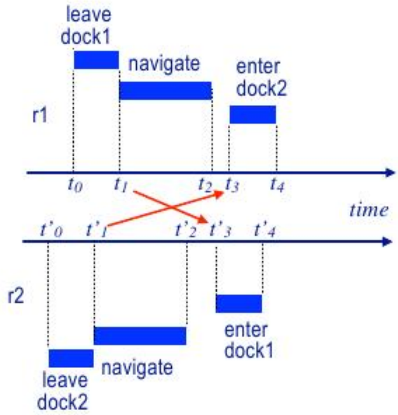



Figure 4.4: Temporally qualified actions of two robots, r1 and r2. The diagonal arrows represent the precedence constraints $t'_1 < t_3$ and $t_1 < t'_3$.

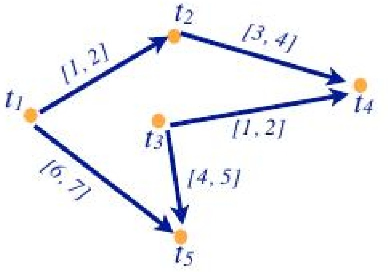



Figure 4.7: A consistent STN.

# DELIBERATION WITH NONDETERMINISTIC MODELS

# MOTIVATION

drops the unrealistic assumption that each action performed in one state leads deterministically to one state

- the search space is no longer represented as a graph; it becomes an AND/OR graph *(And branch corresponds to one action that may lead to many possible states, OR branch corresponds to choosing which action to apply)*
- plans cannot be restricted to sequences of actions
- different types of solution plans - guarantee the achievement, have some chances of success, ...

*the motivation for interleaving acting with planning is even stronger in the case of nondeterministic models*

# PLANNING DOMAIN - AND /OR GRAPH

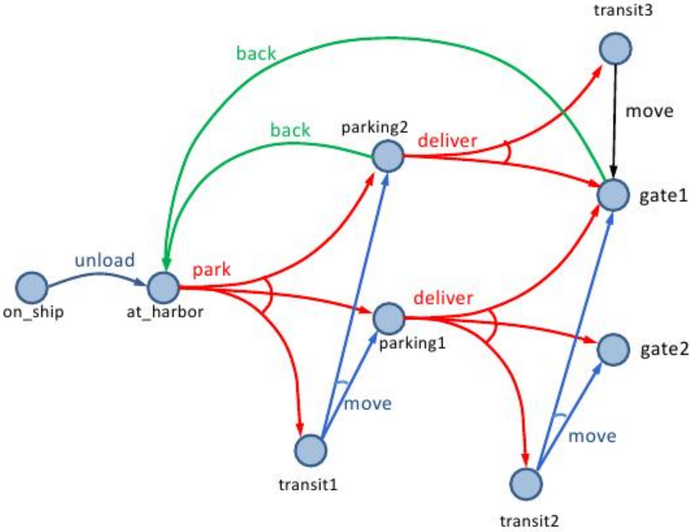



Figure 5.1: A simple nondeterministic planning domain model.

# PLANS AS POLICIES

planning problem a set of goal states $S_g$

memoryless policy partial function $\pi$ that maps states into actions

$\hat{\gamma}(s, \pi)$ the set of states reachable from state $s$ by a policy $\pi$

$\mathit{leaves}(s, \pi)$ reachable states without action ($\mathit{leaf} \notin \mathit{Dom}(\pi)$)

solution policy $\pi$ with $\mathit{leaves}(s_0, \pi) \cap S_g \neq \emptyset$

safe solution policy $\pi$ with $\forall s \in \hat{\gamma}(s_0, \pi)(\mathit{leaves}(s, \pi) \cap S_g \neq \emptyset)$

*acyclic and cyclic safe solutions*

# AND/OR GRAPH SEARCH

planning by forward search

planning by minmax search

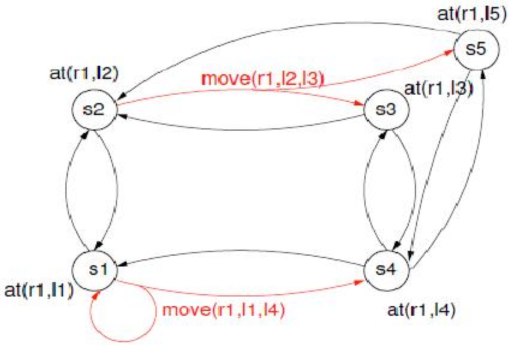
symbolic model checking techniques

determinization techniques *Consider one of the possible many outcomes of a nondeterministi action at a time, find a plan that works in the deterministic case. Then different nondeterminitic outcomes of an action are considered and a new plan for that state is computed, and finally the results are joined in a contingent plan.*

online approaches interleaving planning and acting, various strategies

# CONTEXT DEPENDENT POLICIES

- c. d. p. are more expresive than policies because they can take into account the context in which a step of the plan is executed, and the context can depend of the steps that have been executed so far
- one could address this issue by extending the representation of a state to include all relevant data (*the history of states visited so far*)
- this might work in theory, but its implementation is not practical
- instead, we introduce the notion of *context*




initial state $s_1$

task:

*achieve-acyclic* $s_2$; *achieve-acyclic* $s_4$

| *state* | *context* | *action* | *next state* | *next context* |
|---|---|---|---|---|
| s1 | c1 | move(r1,l1,l2) | s2 | c2 |
| s1 | c2 | move(r1,l1,l4) | s1 | c1 |
| s1 | c2 | move(r1,l1,l4) | s4 | c1 |
| s2 | c2 | move(r1,l2,l1) | s1 | c2 |
| s4 | c1 | move(r1,l4,l1) | s1 | c1 |

# SEARCH AUTOMATA

- states of each search automaton correspond to the contexts of the plan under construction
- search automaton is generated automatically from given task

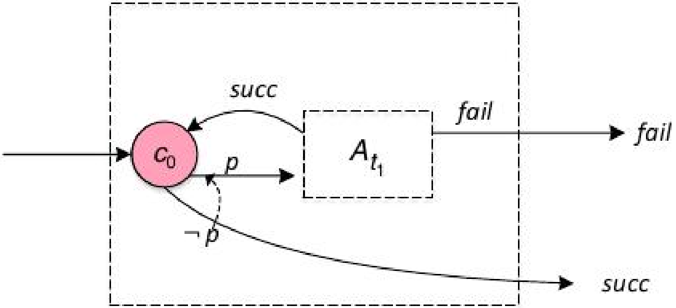



Figure 5.16: Search automaton for loop task *while* $p$ *do* $t_1$.

# PLANNING BASED ON SEARCH AUTOMATA

search automaton $\quad \times \quad$ planning domain

# ACTING WITH INPUT/OUTPUT AUTOMATA

model components that interact with each other

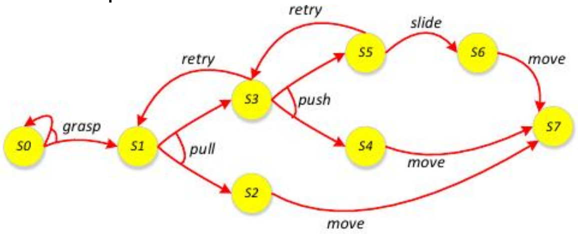



Figure 5.19: Nondeter

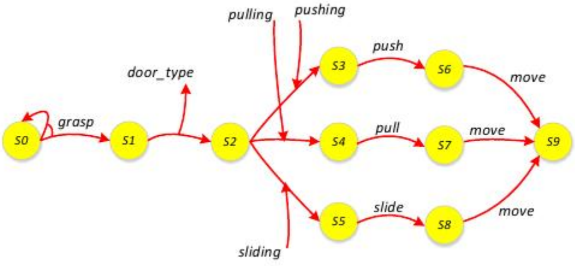
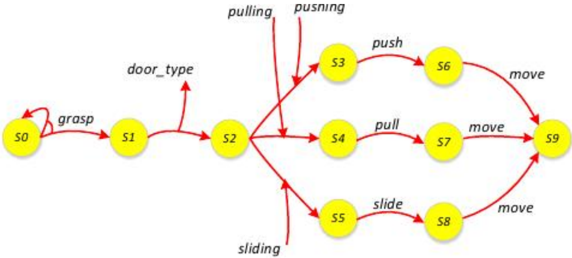
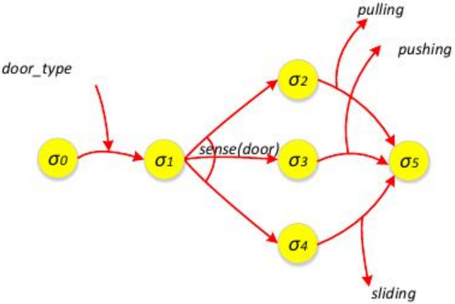



Figure 5.20: Input/output automaton for an open-door method.

# CONTROL AUTOMATA



# AUTOMATED SYNTHESIS OF CONTROL AUTOMATA

- generate control automata automatically, either offline of at run-time
- for different types of tasks

# DELIBERATION WITH PROBABILITIC MODELS

# MOTIVATION

- future is never entirely predictable
- models are necessarily incomplete
- complete deterministic models are often too complex and costly to develop

## PLANNING DOMAIN

- Markov decsion process
- nondeterministic state-transition system together with a probability distribution and a cost distribution

# PLANNING PROBLEM

policy is a partial function $\pi \colon S' \to A$

$\hat{\gamma}(s, \pi)$ set of descendants of $s$ reachable by $\pi$

leaves$(s, \pi)$ states in $\hat{\gamma}(s, \pi)$ that have no successors with $\pi$

SSP problem set $S_g$ of goal states

solution to the SSP problem — policy $\pi$ s.t. leaves$(s_0, \pi) \cap S_g \neq \emptyset$

closed solution policy $\pi$ providing applicable actions, if there are any, to $s_0$ and to its all descendants reachable by $\pi$

safe solution $\Pr(S_g | s_0, \pi) = 1$ [1]

unsafe solution $0 < \Pr(S_g | s_0, \pi) < 1$

[1] $\Pr(S_g | s_0, \pi) = \lim_{l \to \infty} \Pr^l(S_g | s_0, \pi)$

# SAFE POLICY

Run-Policy($\Sigma, s_0, S_g, \pi$)
    $s \leftarrow s_0$
    while $s \notin S_g$ and Applicable($s$) $\neq \varnothing$ do
        perform action $\pi(s)$
        $s \leftarrow$ observe resulting state

Algorithm 6.1: A simple procedure to run a policy.

- a policy $\pi$ is safe iff $\forall s \in \hat{\gamma}(s_0, \pi)$ there is a path from $s$ to a goal
- with a safe policy `Run-Policy` always reaches a goal
- with an unsafe policy `Run-Policy` may or may not terminate; if it does terminate, it may reach either foal or a state with no applicable action

# OPTIMALITY PRINCIPLE

- $V^\pi : Dom(\pi) \rightarrow \mathbb{R}^+$ be a value function giving the expected sum of the cost of the actions obtained by following a safe policy $\pi$ from a sate $s$ to a goal, $V^\pi(s) = \sum_\sigma \Pr(\sigma)\text{cost}(\sigma)$
- $$V^\pi(s) = \begin{cases} 0 & \text{if } s \in S_g \\ \text{cost}(s, \pi(s)) + \sum_{s' \in \gamma(s,\pi(s))} \Pr(s'|s, \pi(s)) V^\pi(s') & \text{otherw} \end{cases}$$
- optimal policy $V^*$ has a minimal expected cost over all possible policies

# POLICY ITERATION ALGORITHM

If $\pi$ is a safe solution, then policy $\pi'$ is safe and $\forall s\ V^{\pi'}(s) \leq V^{\pi}(s)$

$$\pi'(s) = \text{argmin}_a \{\text{cost}(s,a) + \sum_{s' \in \gamma(s,\pi(s))} \Pr(s'|s,a)\ V^{\pi}(s')$$

$\mathsf{PI}(\Sigma, \pi_0)$
  $\pi \leftarrow \pi_0$
  loop until reaching a fixed point
    compute $\{V^{\pi}(s) \mid s \in S\}$ *(i)*
    for every $s \in S \setminus S_g$ do
      $\pi(s) \leftarrow \text{argmin}_a \{\text{cost}(s,a) + \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a) V^{\pi}(s')\}$ *(ii)*

Algorithm 6.2: Policy Iteration algorithm.

computing $V^{\pi}$ for current $\pi$: system of $n$ linear equations with $n$ unknow variables or iterative method

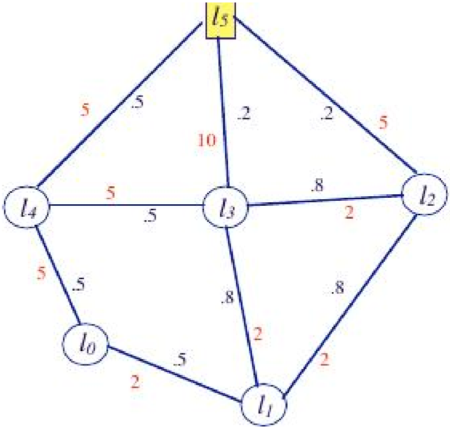
# VALUE ITERATION ALGORITHM

from $V$, a new value function can be computed with the following equation

$$V'(s) = \min_a \{\text{cost}(s,a) + \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\, V(s')\}$$

VI($\Sigma, V_0$)
  $V \leftarrow V_0$
  loop until until reaching a fixed point
    for every $s \in S \setminus S_g$ do
      $V'(s) \leftarrow \min_a \{\text{cost}(s,a) + \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a) V(s')\}$
    $V \leftarrow V'$
  $\pi(s) \leftarrow \text{argmin}_a \{\text{cost}(s,a) + \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a) V(s')\}$

Algorithm 6.3: Synchronous Value Iteration algorithm. $V_0$ is implemented as a function, computed once in every state; $V$, $V'$ and $\pi$ are lookup tables.




| iteration | $l_0$ | $l_1$ | $l_2$ | $l_3$ | $l_4$ |
|---|---|---|---|---|---|
| 1 | 2.00 | 2.00 | 2.00 | 3.60 | 5.00 |
| 2 | 4.00 | 4.00 | 5.28 | 5.92 | 7.50 |
| 3 | 6.00 | 7.00 | 7.79 | 8.78 | 8.75 |
| 10 | 19.52 | 21.86 | 21.16 | 19.76 | 9.99 |
| 11 | 19.75 | 22.18 | 21.93 | 19.88 | 10.00 |
| 12 | 19.87 | 22.34 | 22.29 | 19.94 | 10.00 |

# HEURISTIC SEARCH ALGORITHMS

heuristic search algorithms exploit the guidance of an initial value function $V_0$ to focus a planning proglem on a small part of the search space

- best-fisrt search
- depth-first search
- iterative deepening search