Artificial Intelligence 1
Winter Semester 2024/25

– Lecture Notes –
Part IV: Planning and Acting

Prof. Dr. Michael Kohlhase

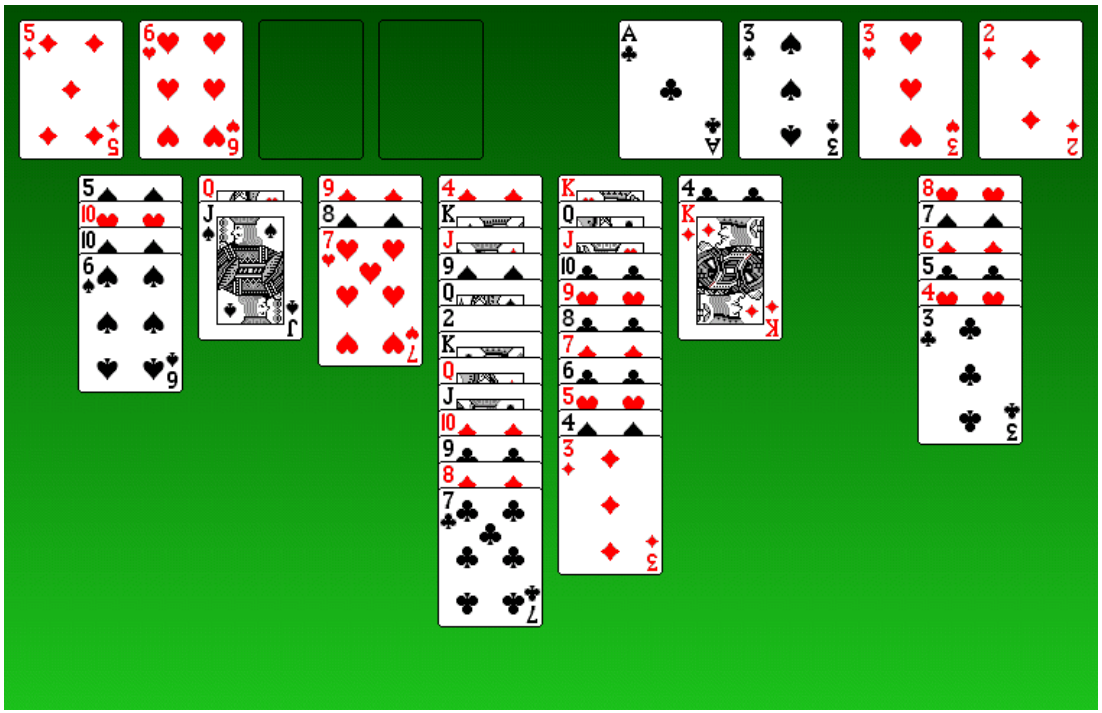
Professur für Wissensrepräsentation und -verarbeitung
Informatik, FAU Erlangen-Nürnberg
Michael.Kohlhase@FAU.de

2025-02-06

Chapter 17
Planning I: Framework

Reminder: Classical Search Problems

- **Example 0.1 (Solitaire as a Search Problem).**



- States: Card positions (e.g. position_Jspades=Qhearts).
- Actions: Card moves (e.g. move_Jspades_Qhearts_freecell4).
- Initial state: Start configuration.
- Goal states: All cards "home".
- Solutions: Card moves solving this game.

Planning

- **Ambition:** Write one program that can solve all classical search problems.
- **Idea:** For CSP, going from "state/action-level search" to "problem-description level search" did the trick.
- **Definition 0.2.** Let Π be a search problem (see **??**)
  - The blackbox description of Π is an API providing functionality allowing to construct the state space: $\mathrm{InitialState}()$, $\mathrm{GoalTest}(s)$, ...
    - "Specifying the problem" $\widehat{=}$ programming the API.
  - The declarative description of Π comes in a problem description language. This allows to implement the API, and much more.
    - "Specifying the problem" $\widehat{=}$ writing a problem description.
- Here, "problem description language" $\widehat{=}$ planning language. (up next)
- **But Wait:** Didn't we do this already in the last chapter with logics? (For the Wumpus?)

17.1 Logic-Based Planning

Fluents: Time-Dependent Knowledge in Planning

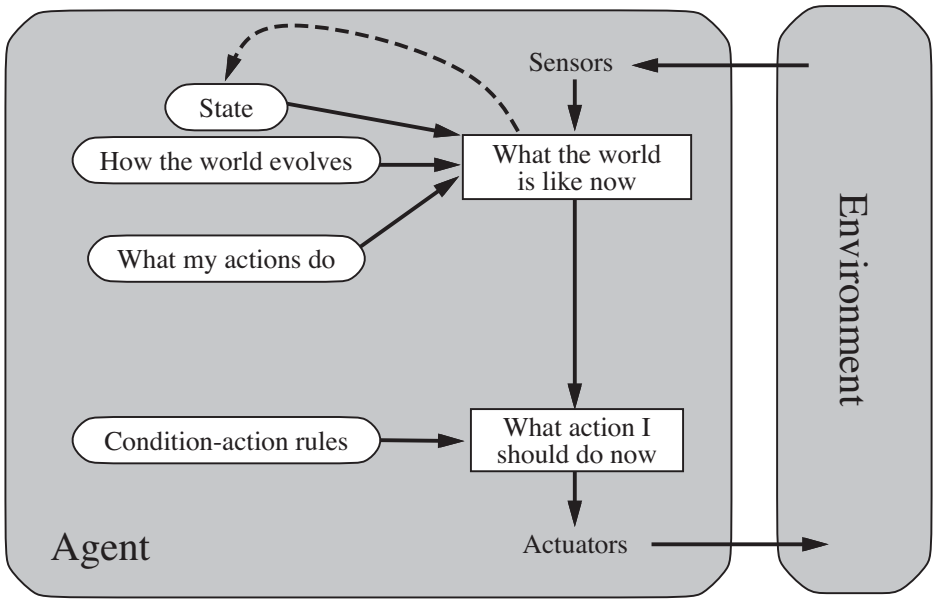
- **Recall from ??:** We can represent the Wumpus rules in logical systems. (propositional/first-order/ALC)
  - Use inference systems to deduce new world knowledge from percepts and actions.
- **Problem:** Representing (changing) percepts immediately leads to contradictions!
- **Example 1.1.** If the agent moves and a cell with a draft at (a perceived breeze) is followed by one without.

Fluents: Time-Dependent Knowledge in Planning

- **Recall from ??:** We can represent the Wumpus rules in logical systems. (propositional/first-order/ALC)
  - Use inference systems to deduce new world knowledge from percepts and actions.
- **Problem:** Representing (changing) percepts immediately leads to contradictions!
- **Example 1.4.** If the agent moves and a cell with a draft at (a perceived breeze) is followed by one without.
- **Obvious Idea:** Make representations of percepts time-dependent
- **Example 1.5.** D^t for $t \in \mathbb{N}$ for PL^0 and $\mathrm{draft}(t)$ in PL^1 and $\mathrm{PL}^{\mathrm{nq}}$.
- **Definition 1.6.** We use the word fluent to refer an aspect of the world that changes, all others we call atemporal.

Recap: Logic-Based Agents

- **Recall:** A model-based agent uses inference to model the environment, percepts, and actions.



Recap: Logic-Based Agents

- **Recall:** A model-based agent uses inference to model the environment, percepts, and actions.

```
function KB−AGENT (percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time
  TELL(KB, MAKE−PERCEPT−SENTENCE(percept,t))
  action := ASK(KB, MAKE−ACTION−QUERY(t))
  TELL(KB, MAKE−ACTION−SENTENCE(action,t))
   t := t+1
  return action
```

Recap: Logic-Based Agents

- **Recall:** A model-based agent uses inference to model the environment, percepts, and actions.

```
function KB–AGENT (percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time
  TELL(KB, MAKE–PERCEPT–SENTENCE(percept,t))
  action := ASK(KB, MAKE–ACTION–QUERY(t))
  TELL(KB, MAKE–ACTION–SENTENCE(action,t))
   t := t+1
  return action
```

- **Still Unspecified:** (up next)
  - MAKE–PERCEPT–SENTENCE: the effects of percepts.
  - MAKE–ACTION–QUERY: what is the best next action?
  - MAKE–ACTION–SENTENCE: the effects of that action.

  In particular, we will look at the effect of time/change. (neglected so far)

Fluents: Modeling the Agent's Sensors

- **Idea:** Relate percept fluents to atemporal cell attributes.
- **Example 1.7.** E.g., if the agent perceives a draft at at time t, when it is in cell $[x, y]$, then there must be a breeze there:

$$\forall t, x, y. \mathrm{Ag@}(t, x, y) \Rightarrow (\mathrm{draft}(t) \Leftrightarrow \mathrm{breeze}(x, y))$$

- Axioms like these model the agent's sensors – here that they are totally reliable: there is a breeze, iff the agent feels a draft at.
- **Definition 1.8.** We call fluents that describe the agent's sensors sensor axioms.
- **Problem:** Where do fluents like $\mathrm{Ag@}(t, x, y)$ come from?

Digression: Fluents and Finite Temporal Domains

- **Observation:** Fluents like $\forall t, x, y.\mathrm{Ag@}(t, x, y) \Rightarrow (\mathrm{draft}(t) \Leftrightarrow \mathrm{breeze}(x, y))$ from **??** are best represented in first-order logic. In PL^0 and $\mathrm{PL}^{\mathrm{nq}}$ we would have to use concrete instances like $\mathrm{Ag@}(7, 2, 1) \Rightarrow (\mathrm{draft}(7) \Leftrightarrow \mathrm{breeze}(2, 1))$ for all suitable t, x, and y.
- **Problem:** Unless we restrict ourselves to finite domains and an end time t_{end} we have infinitely many axioms. Even then, formalization in PL^0 and $\mathrm{PL}^{\mathrm{nq}}$ is very tedious.
- **Solution:** Formalize in first-order logic and then compile down:
  1. enumerate ranges of bound variables, instantiate body, ($\leadsto \mathrm{PL}^{\mathrm{nq}}$)
  2. translate $\mathrm{PL}^{\mathrm{nq}}$ atoms to propositional variables. ($\leadsto \mathrm{PL}^0$)
- **In Practice:** The choice of domain, end time, and logic is up to agent designer, weighing expressivity vs. efficiency of inference.
- WLOG: We will use PL^1 in the following. (easier to read)

Fluents: Effect Axioms for the Transition Model

- **Problem:** Where do fluents like $\mathrm{Ag@}(t,x,y)$ come from?
- **Thus:** We also need fluents to keep track of the agent's actions. (The transition model of the underlying search problem).
- **Idea:** We also use fluents for the representation of actions.
- **Example 1.9.** The action of "going forward" at time t is captured by the fluent $\mathrm{forw}(t)$.
- **Definition 1.10.** Effect axioms describe how the environment changes under an agent's actions.
- **Example 1.11.** If the agent is in cell $[1,1]$ facing east at time 0 and goes forward, she is in cell $[2,1]$ and no longer in $[1,1]$:

$$\mathrm{Ag@}(0,1,1) \wedge \mathrm{faceeast}(0) \wedge \mathrm{forw}(0) \Rightarrow \mathrm{Ag@}(1,2,1) \wedge \neg\mathrm{Ag@}(1,1,1)$$

Generally: (barring exceptions for domain border cells)

$$\forall t,x,y.\mathrm{Ag@}(t,x,y) \wedge \mathrm{faceeast}(t) \wedge \mathrm{forw}(t) \Rightarrow \mathrm{Ag@}(t{+}1,x{+}1,y) \wedge \neg\mathrm{Ag@}(t{+}1,x,y)$$

This compiles down to $16 \cdot t_{\mathrm{end}}$ $\mathrm{PL}^{\mathrm{nq}}/\mathrm{PL}^{0}$ axioms.

Frames and Frame Axioms

- **Problem:** Effect axioms are not enough.
- **Example 1.12.** Say that the agent has an arrow at time 0, and then moves forward at into $[2,1]$, perceives a glitter, and knows that the Wumpus is ahead. To evaluate the action $\mathrm{shoot}(1)$ the corresponding effect axiom needs to know $\mathrm{havarrow}(1)$, but cannot prove it from $\mathrm{havarrow}(0)$.
  Problem: The information of having an arrow has been lost in the move forward.
- **Definition 1.13.** The frame problem describes that for a representation of actions we need to formalize their effects on the aspects they change, but also their non-effect on the static frame of reference.
- **Partial Solution:** (there are many many more; some better)
  Frame axioms formalize that particular fluents are invariant under a given action.
- **Problem:** For an agent with n actions and an environment with m fluents, we need $\mathcal{O}(nm)$ frame axioms.
  Representing and reasoning with them easily drowns out the sensor and transition models.

A Hybrid Agent for the Wumpus World

- **Example 1.14 (A Hybrid Agent).** This agent uses
  - logic inference for sensor and transition modeling,
  - special code and A^* for action selection & route planning.

```
function HYBRID−WUMPUS−AGENT(percept) returns an action
    inputs: percept, a list, [stench,breeze,glitter,bump,scream]
    persistent: KB, a knowledge base, initially the atemporal
                                "wumpus physics"
                t, a counter, initially 0, indicating time
                plan, an action sequence, initially empty
  TELL(KB, MAKE−PERCEPT−SENTENCE(percept,t))
```

then some special code for action selection, and then (up next)

```
  action := POP(plan)
  TELL(KB, MAKE−ACTION−SENTENCE(action,t))
  t := t + 1
  return action
```

So far, not much new over our original version.

A Hybrid Agent: Custom Action Selection

- **Example 1.15 (A Hybrid Agent (continued)).** So that we can plan the best strategy:

TELL(KB, the temporal "physics" sentences **for** time t)
$safe := \{[x,y] \mid \text{ASK}(KB, \text{OK}(t,x,y)) = \text{T}\}$
if ASK(KB,glitter(t)) = T **then**
  $plan := [\text{grab}] + \text{PLAN–ROUTE}(current, \{[1,1]\}, safe) + [\text{exit}]$
if $plan$ is empty **then**
  $unvisited := \{[x,y] \mid \text{ASK}(KB, \text{Ag@}(t',x,y)) = \text{F}\}$ **for** all $t' \leq t$
  $plan := \text{PLAN–ROUTE}(current, unvisited \cup safe, safe)$
if $plan$ is empty and ASK(KB,havarrow(t)) = T **then**
  $possible\_wumpus := \{x,y \mid [x,y]\}\text{ASK}(KB, \neg\text{wumpus}(t,x,y)) = \text{F}$
  $plan := \text{PLAN–SHOT}(current, possible\_wumpus, safe)$
if $plan$ is empty **then** // no choice but **to** take a risk
  $not\_unsafe := \{[x,y] \mid \text{ASK}(KB, \neg\text{OK}(t,x,y)) = \text{F}\}$
  $plan := \text{PLAN–ROUTE}(current, unvisited \cup not\_unsafe, safe)$
if $plan$ is empty **then**
  $plan := \text{PLAN–ROUTE}(current, \{[1,1]\}, safe) + [\text{exit}]$

Note that OK wumpus, and glitter are fluents, since the Wumpus might have died or the gold might have been grabbed.

A Hybrid Agent: Custom Action Selection

- **Example 1.16 (Action Selection).** And the code for PLAN–ROUTE (PLAN–SHOT similar)

```
function PLAN–ROUTE(curr,goals,allowed) returns an action sequence
  inputs: curr, the agent's current position
          goals, a set of squares;
                 try to plan a route to one of them
          allowed, a set of squares that can form part of the route
  problem := ROUTE–PROBLEM(curr,goals,allowed)
  return A*(problem)
```

- **Evaluation:** Even though this works for the Wumpus world, it is not the "universal, logic-based problem solver" we dreamed of!
- Planning tries to solve this with another representation of actions. (up next)

17.2 Planning: Introduction

How does a planning language describe a problem?

- **Definition 2.1.** A planning language is a way of describing the components of a search problem via formulae of a logical system. In particular the
  - states (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$.)

How does a planning language describe a problem?

- **Definition 2.3.** A planning language is a way of describing the components of a search problem via formulae of a logical system. In particular the
  - states (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$.)
  - initial state I (vs. data structures). (E.g.: $Eq(x, 1)$.)

How does a planning language describe a problem?

- **Definition 2.5.** A planning language is a way of describing the components of a search problem via formulae of a logical system. In particular the
  - states (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$.)
  - initial state I (vs. data structures). (E.g.: $Eq(x,1)$.)
  - goal states G (vs. a goal test). (E.g.: $Eq(x,2)$.)

How does a planning language describe a problem?

- **Definition 2.7.** A planning language is a way of describing the components of a search problem via formulae of a logical system. In particular the
  - states (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$.)
  - initial state I (vs. data structures). (E.g.: $Eq(x,1)$.)
  - goal states G (vs. a goal test). (E.g.: $Eq(x,2)$.)
  - set A of actions in terms of preconditions and effects (vs. functions returning applicable actions and successor states). (E.g.: "increment x: pre $Eq(x,1)$, iff $Eq(x \wedge 2) \wedge \neg Eq(x,1)$".)

  A logical description of all of these is called a planning task.

How does a planning language describe a problem?

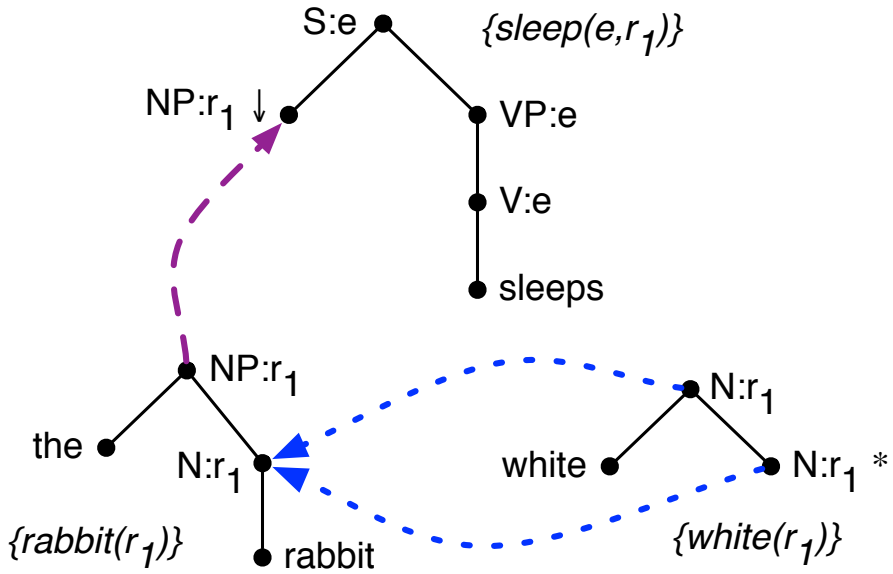
- **Definition 2.9.** A planning language is a way of describing the components of a search problem via formulae of a logical system. In particular the
  - states (vs. blackbox: data structures). (E.g.: predicate $Eq(.,.)$.)
  - initial state I (vs. data structures). (E.g.: $Eq(x,1)$.)
  - goal states G (vs. a goal test). (E.g.: $Eq(x,2)$.)
  - set A of actions in terms of preconditions and effects (vs. functions returning applicable actions and successor states). (E.g.: "increment x: pre $Eq(x,1)$, iff $Eq(x \wedge 2) \wedge \neg Eq(x,1)$".)

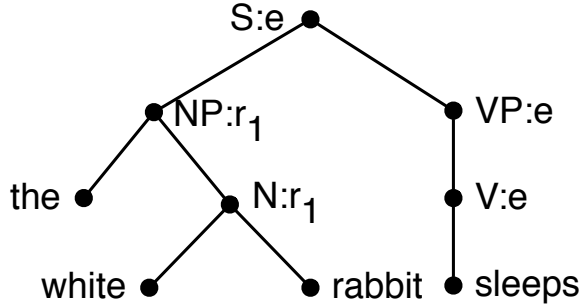
  A logical description of all of these is called a planning task.
- **Definition 2.10.** Solution (plan) $\widehat{=}$ sequence of actions from $\mathcal{A}$, transforming $\mathcal{I}$ into a state that satisfies $\mathcal{G}$. (E.g.: "increment x".)
  The process of finding a plan given a planning task is called planning.

Planning Language Overview

- **Disclaimer:** Planning languages go way beyond classical search problems. There are variants for inaccessible, stochastic, dynamic, continuous, and multi-agent settings.
- We focus on classical search for simplicity (and practical relevance).
- For a comprehensive overview, see [GNT04].

Application: Natural Language Generation

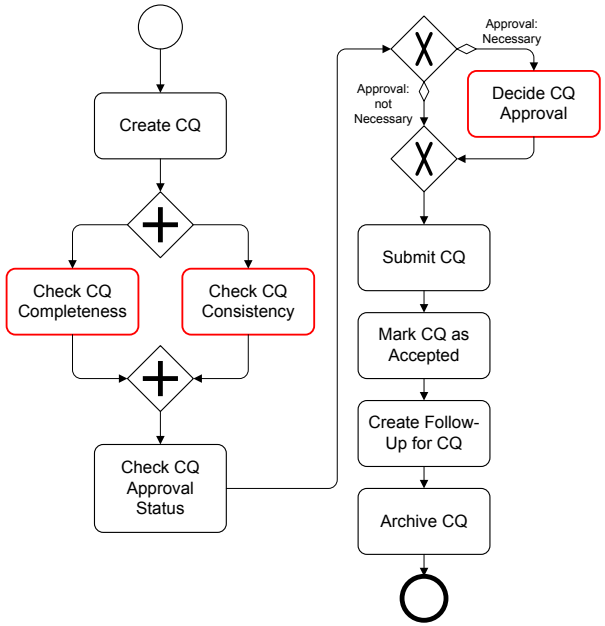




- **Input**: Tree-adjoining grammar, intended meaning.
- **Output**: Sentence expressing that meaning.

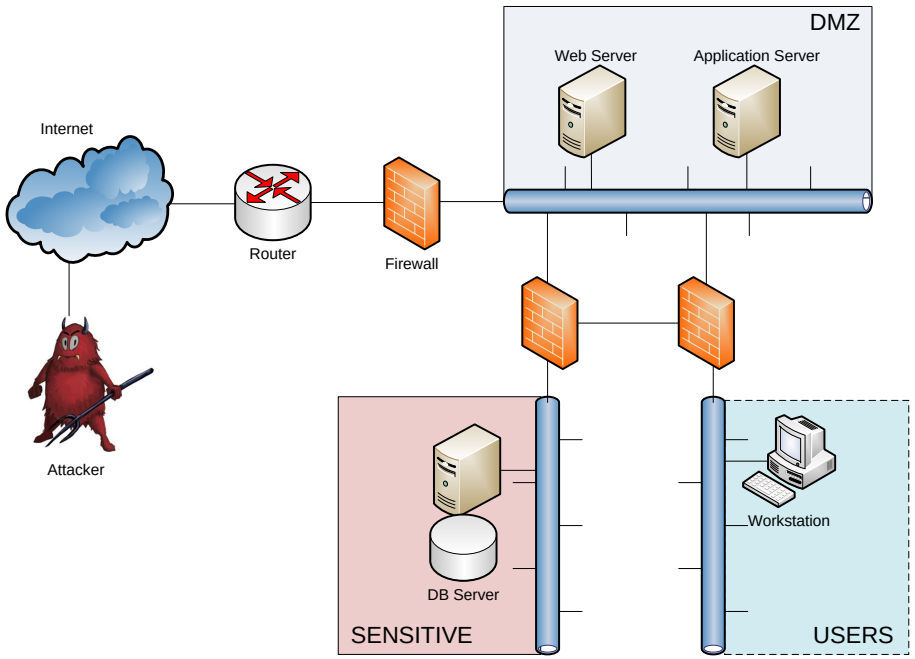
Application: Business Process Templates at SAP

Action name	precondition	effect
Check CQ Completeness	CQ.archiving:notArchived	CQ.completeness:complete OR CQ.completeness:notComplete
Check CQ Consistency	CQ.archiving:notArchived	CQ.consistency:consistent OR CQ.consistency:notConsistent
Check CQ Approval Status	CQ.archiving:notArchived AND CQ.approval:notChecked AND CQ.completeness:complete AND CQ.consistency:consistent	CQ.approval:necessary OR CQ.approval:notNecessary
Decide CQ Approval	CQ.archiving:notArchived AND CQ.approval:necessary	CQ.approval:granted OR CQ.approval:notGranted
Submit CQ	CQ.archiving:notArchived AND (CQ.approval:notNecessary OR CQ.approval:granted)	CQ.submission:submitted
Mark CQ as Accepted	CQ.archiving:notArchived AND CQ.submission:submitted	CQ.acceptance:accepted
Create Follow-Up for CQ	CQ.archiving:notArchived AND CQ.acceptance:accepted	CQ.followUp:documentCreated
Archive CQ	CQ.archiving:notArchived	CQ.archiving:archived



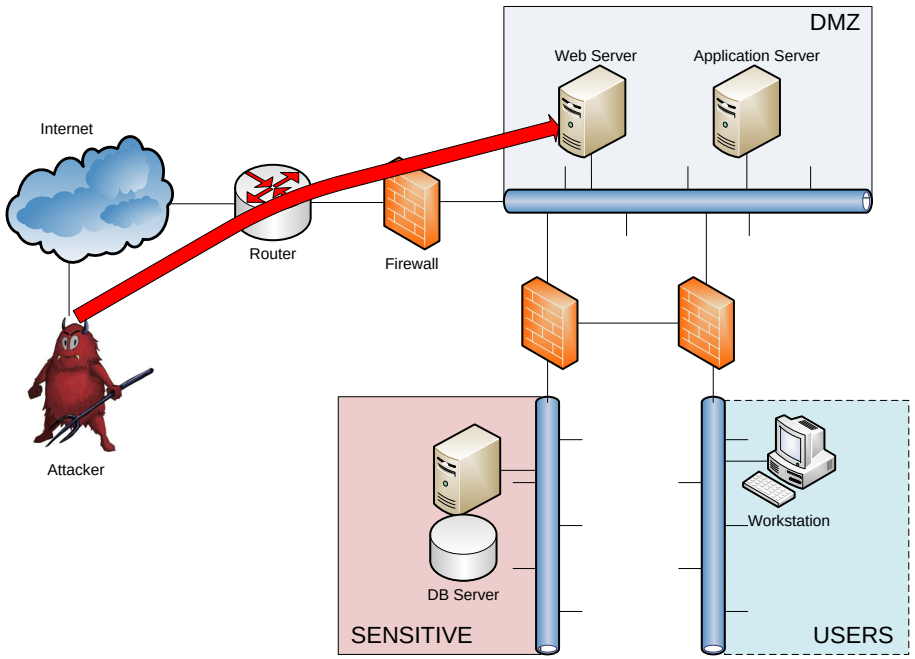
- **Input**: model of behavior of activities on business objects, process endpoint.
- **Output**: Process template leading to this point.

Application: Automatic Hacking



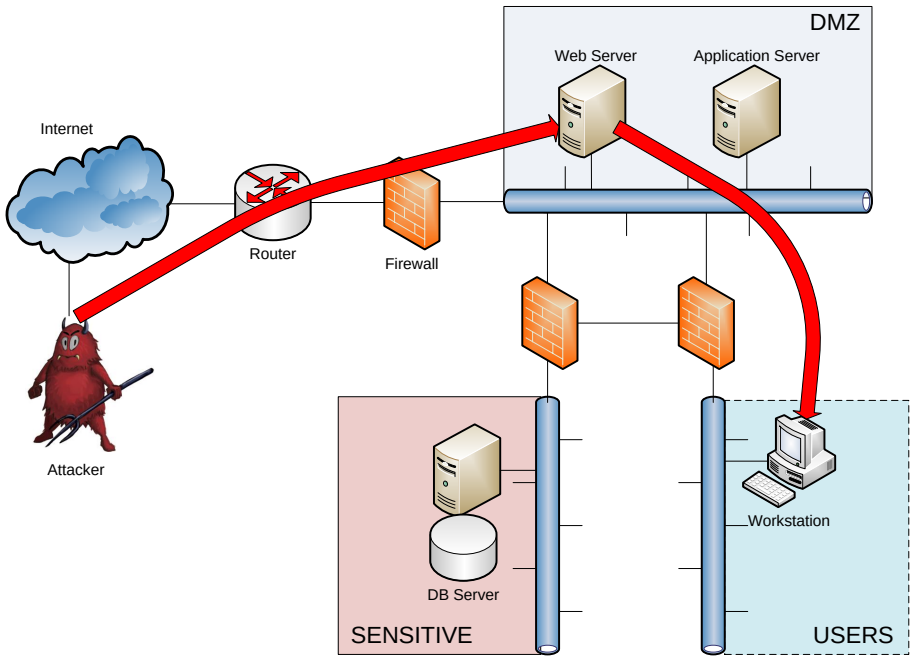
- **Input**: Network configuration, location of sensible data.
- **Output**: Sequence of exploits giving access to that data.

Application: Automatic Hacking



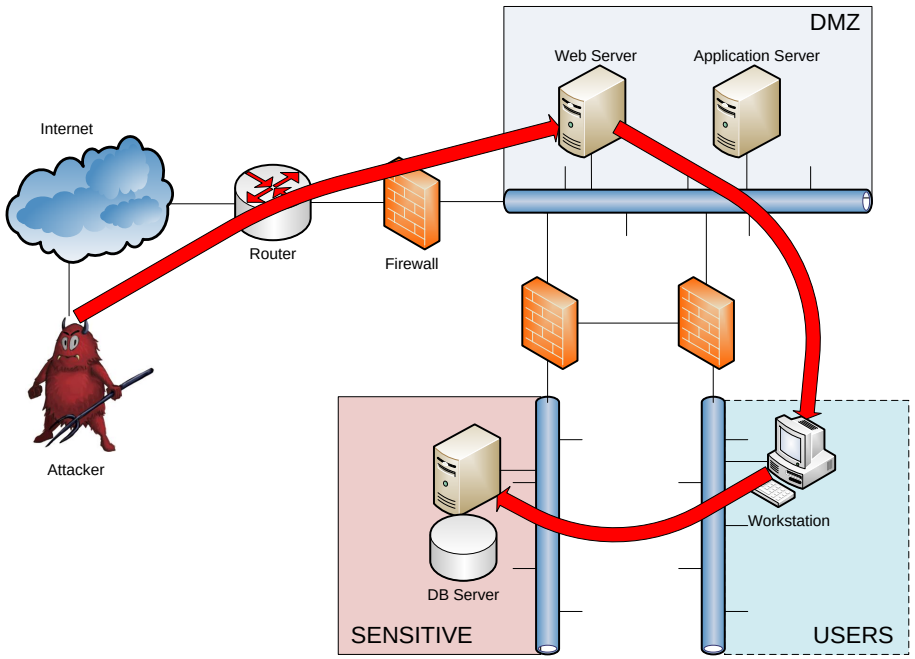
- **Input**: Network configuration, location of sensible data.
- **Output**: Sequence of exploits giving access to that data.

Application: Automatic Hacking



- **Input**: Network configuration, location of sensible data.
- **Output**: Sequence of exploits giving access to that data.

Application: Automatic Hacking



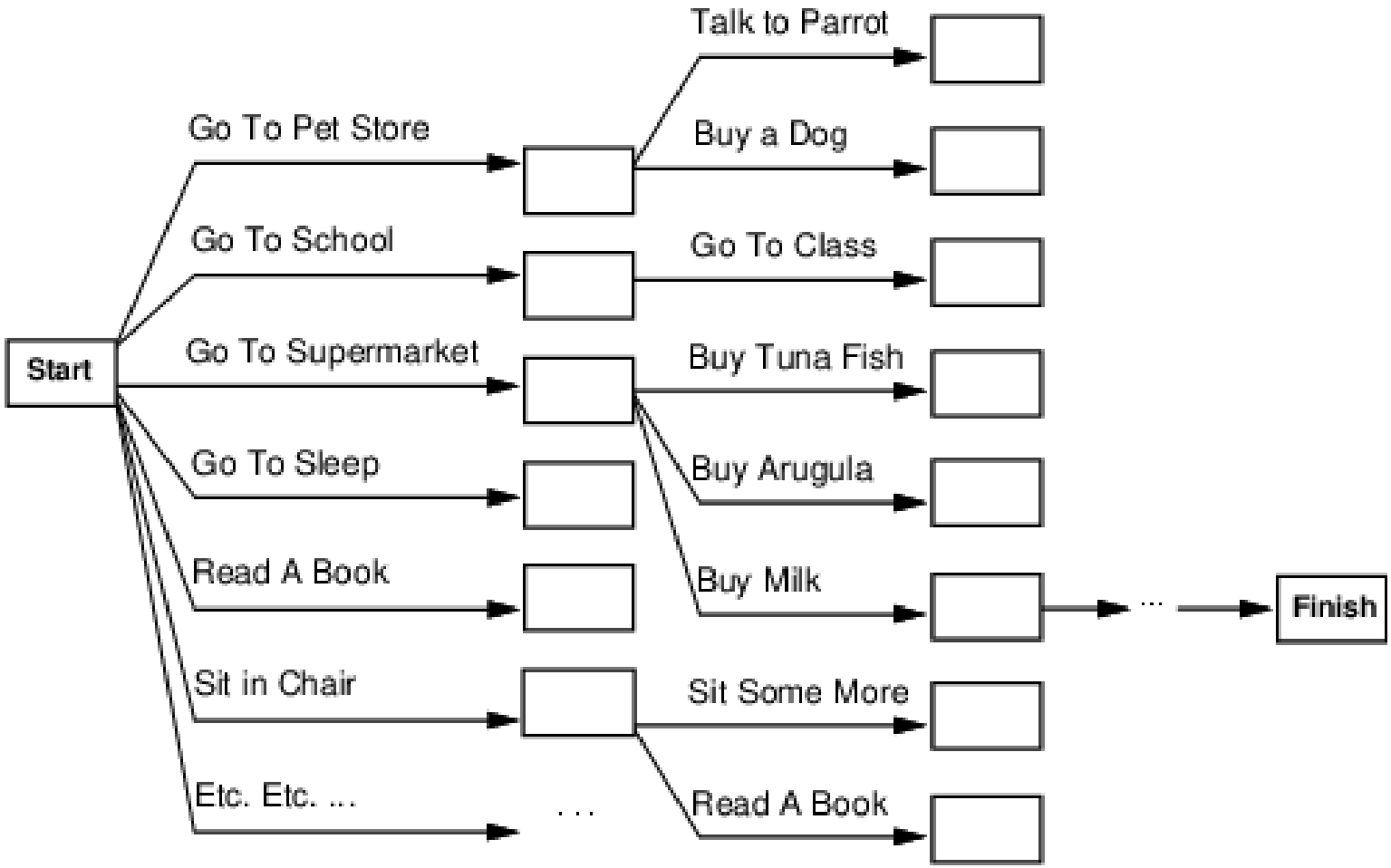
- **Input**: Network configuration, location of sensible data.
- **Output**: Sequence of exploits giving access to that data.

Reminder: General Problem Solving, Pros and Cons

- **Powerful:** In some applications, generality is absolutely necessary. (E.g. SAP)
- **Quick:** Rapid prototyping: 10s lines of problem description vs. 1000s lines of C++ code. (E.g. language generation)
- **Flexible:** Adapt/maintain *the description*. (E.g. network security)
- **Intelligent:** Determines automatically how to solve a complex problem efficiently! (The ultimate goal, no?!)
- **Efficiency loss:** Without any domain-specific knowledge about chess, you don't beat Kasparov . . .
  - Trade-off between "automatic and general" vs. "manual work but efficient".
- **Research Question:** How to make fully automatic algorithms efficient?

Search vs. planning

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



After-the-fact heuristic/goal test inadequate

Search vs. planning (cont.)

- Planning systems do the following:
  1. open up action and goal representation to allow selection
  2. divide-and-conquer by subgoaling
- 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

Reminder: Greedy Best-First Search and A^*

- **Recall:** Our heuristic search algorithms (duplicate pruning omitted for simplicity)

```
function Greedy_Best—First_Search (problem)
  returns a solution, or failure
  n := node with n.state=problem.InitialState
  frontier := priority queue ordered by ascending h, initially [n]
  loop do
    if Empty?(frontier) then return failure
    n := Pop(frontier)
    if problem.GoalTest(n.state) then return Solution(n)
    for each action a in problem.Actions(n.state) do
      n' := ChildNode(problem,n,a)
      Insert(n', h(n'), frontier)
```

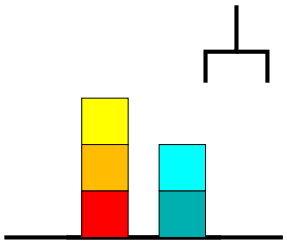
For A^*

  - order *frontier* by $g + h$ instead of h (line 4)
  - insert $g(n') + h(n')$ instead of $h(n')$ to *frontier* (last line)

- Is greedy best-first search optimal? No $\rightsquigarrow$ satisficing planning.
- Is A^* optimal? Yes, but only if h is admissible $\rightsquigarrow$ optimal planning, **with such** h.

ps. "Making Fully Automatic Algorithms Efficient"

- **Example 2.11.**



  - n blocks, 1 hand.
  - A single action either takes a block with the hand or puts a block we're holding onto some other block/the table.

blocks	states	blocks	states
1	1	9	4596553
2	3	10	58941091
3	13	11	824073141
4	73	12	12470162233
5	501	13	202976401213
6	4051	14	3535017524403
7	37633	15	65573803186921
8	394353	16	1290434218669921

- **Observation 2.12.** *State spaces typically are huge even for simple problems.*
- **In other words:** Even solving "simple problems" automatically (without help from a human) requires a form of intelligence.
- With blind search, even the largest super computer in the world won't scale beyond 20 blocks!

Algorithmic Problems in Planning

- **Definition 2.13.** We speak of satisficing planning if
  Input: A planning task Π.
  Output: A plan for Π, or "unsolvable" if no plan for Π exists.
  and of optimal planning if
  Input: A planning task Π.
  Output: An optimal plan for Π, or "unsolvable" if no plan for Π exists.
- The techniques successful for either one of these are almost disjoint. And satisficing planning is *much* more efficient in practice.
- **Definition 2.14.** Programs solving these problems are called (optimal) planner, planning system, or planning tool.

Our Agenda for This Topic

- **Now:** Background, planning languages, complexity.
  - Sets up the framework. Computational complexity is essential to distinguish different algorithmic problems, and for the design of heuristic functions. (see next)
- **Next:** How to automatically generate a heuristic function, given planning language input?
  - Focussing on heuristic search as the solution method, this is the main question that needs to be answered.

Our Agenda for This Chapter

1. **The History of Planning**: How did this come about?
   - Gives you some background, and motivates our choice to focus on heuristic search.
2. **The STRIPS Planning Formalism**: Which concrete planning formalism will we be using?
   - Lays the framework we'll be looking at.
3. **The PDDL Language**: What do the input files for off-the-shelf planning software look like?
   - So you can actually play around with such software. (Exercises!)
4. **Planning Complexity**: How complex is planning?
   - The price of generality is complexity, and here's what that "price" is, exactly.

17.3 The History of Planning

Planning History: In the Beginning . . .

- **In the beginning: Man invented Robots:**
  - "Planning" as in "the making of plans by an autonomous robot".
  - Shakey the Robot (Full video here)
- **In a little more detail:**
  - [NS63] introduced *general problem solving*.
  - *. . . not much happened (well not much we still speak of today) . . .*
  - 1966-72, Stanford Research Institute developed a robot named "Shakey".
  - They needed a "planning" component taking decisions.
  - They took inspiration from general problem solving and theorem proving, and called the resulting algorithm STRIPS.

History of Planning Algorithms

- **Compilation into Logics/Theorem Proving:**
  - e.g. $\exists s_0, a, s_1 . at(A, s_0) \wedge execute(s_0, a, s_1) \wedge at(B, s_1)$
  - **Popular when**: Stone Age – 1990.
  - **Approach**: *From planning task description, generate PL1 formula φ that is satisfiable iff there exists a plan; use a theorem prover on φ.*
  - **Keywords/cites**: Situation calculus, frame problem, ...
- Partial order planning
  - e.g. $open = \{at(B)\}$; apply $move(A, B)$; $\rightsquigarrow$ $open = \{at(A)\}$ ...
  - **Popular when**: 1990 – 1995.
  - **Approach**: *Starting at goal, extend partially ordered set of actions by inserting achievers for open sub-goals, or by adding ordering constraints to avoid conflicts.*
  - **Keywords/cites**: **UCPOP** [PW92], causal links, flaw selection strategies, ...

History of Planning Algorithms, ctd.

- GraphPlan
  - e.g. $F_0 = at(A)$; $A_0 = \{move(A, B)\}$; $F_1 = \{at(B)\}$; mutex $A_0 = \{move(A, B), move(A, C)\}$.
  - **Popular when**: 1995 – 2000.
  - **Approach**: *In a forward phase, build a layered "planning graph" whose "time steps" capture which pairs of action can achieve which pairs of facts; in a backward phase, search this graph starting at goals and excluding options proved to not be feasible.*
  - **Keywords/cites**: [BF95; BF97; Koe+97], action/fact mutexes, step-optimal plans, ...
- **Planning as SAT:**
  - SAT variables $at(A)_0$, $at(B)_0$, $move(A, B)_0$, $move(A, C)_0$, $at(A)_1$, $at(B)_1$; clauses to encode transition behavior e.g. ${at(B)_1}^{\mathsf{F}} \vee {move(A, B)_0}^{\mathsf{T}}$; unit clauses to encode initial state ${at(A)_0}^{\mathsf{T}}$, ${at(B)_0}^{\mathsf{T}}$; unit clauses to encode goal ${at(B)_1}^{\mathsf{T}}$.
  - **Popular when**: 1996 – today.
  - **Approach**: *From planning task description, generate propositional CNF formula φ_k that is satisfiable iff there exists a plan with k steps; use a SAT solver on φ_k, for different values of k.*
  - **Keywords/cites**: [KS92; KS98; RHN06; Rin10], SAT encoding schemes, BlackBox, ...

History of Planning Algorithms, ctd.

- **Planning as Heuristic Search:**
  - init $at(A)$; apply $move(A, B)$; generates state $at(B)$; ...
  - **Popular when**: 1999 – today.
  - **Approach**: Devise a method $\mathcal{R}$ to simplify ("relax") any planning task Π; given Π, solve $\mathcal{R}(\Pi)$ to generate a heuristic function h for informed search.
  - **Keywords/cites**: [BG99; HG00; BG01; HN01; Ede01; GSS03; Hel06; HHH07; HG08; KD09; HD09; RW10; NHH11; KHH12a; KHH12b; KHD13; DHK15], critical path heuristics, ignoring delete lists, relaxed plans, landmark heuristics, abstractions, partial delete relaxation, ...

The International Planning Competition (IPC)

- **Definition 3.1.** The International Planning Competition (IPC) is an event for benchmarking planners (`http://ipc.icapsconference.org/`)
  - **How**: Run competing planners on a set of benchmarks.
  - **When**: Runs every two years since 2000, annually since 2014.
  - **What**: Optimal track vs. satisficing track; others: uncertainty, learning, ...
- **Prerequisite/Result:**
  - Standard representation language: PDDL [McD+98; FL03; HE05; Ger+09]
  - Problem Corpus: $\approx$ 50 **domains**, $\gg$ 1000 **instances**, 74 (!!) planners in 2011

International Planning Competition

- **Question:** If planners x and y compete in IPC'YY, and x wins, is x "better than" y?

International Planning Competition

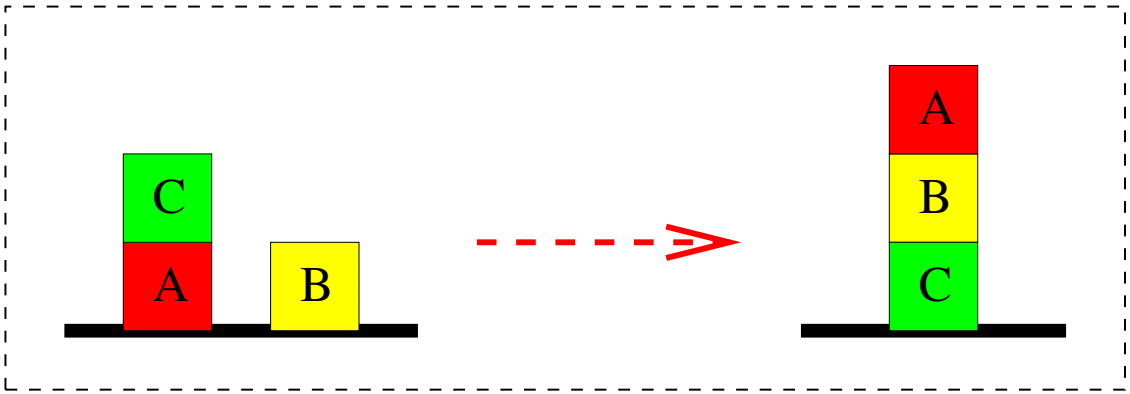
- **Question:** If planners x and y compete in IPC'YY, and x wins, is x "better than" y?
- **Answer:** Yes, but only on the IPC'YY benchmarks, and only according to the criteria used for determining a "winner"! On other domains and/or according to other criteria, you may well be better off with the "looser".

International Planning Competition

- **Question:** If planners x and y compete in IPC'YY, and x wins, is x "better than" y?
- **Answer:** Yes, but only on the IPC'YY benchmarks, and only according to the criteria used for determining a "winner"! On other domains and/or according to other criteria, you may well be better off with the "looser".
- **Generally:** Assessing AI System suitability is complicated, over-simplification is dangerous. (But, of course, nevertheless is being done all the time)

Planning History, p.s.: Planning is Non-Trivial!

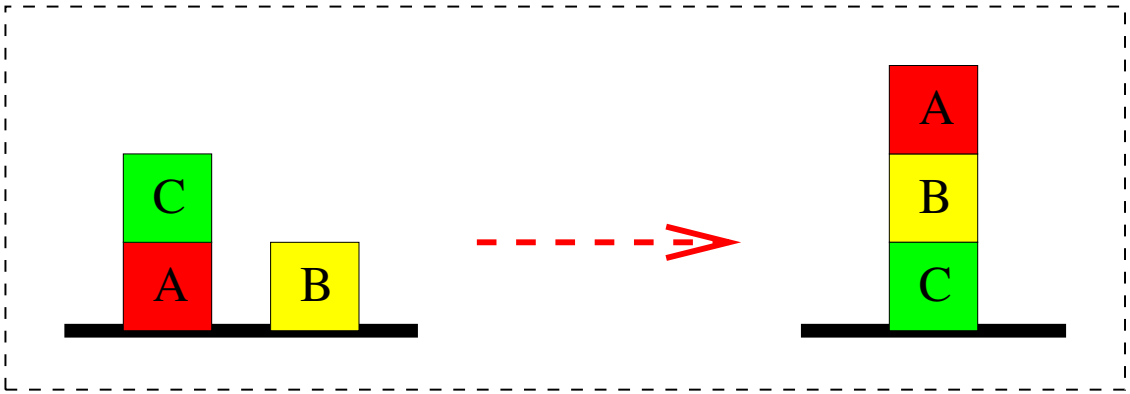
- **Example 3.2.** The Sussman anomaly is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ and $\mathrm{on}(B, C)$ fail:

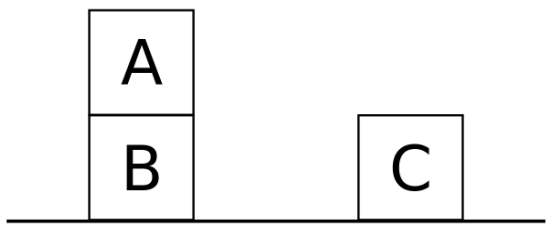
Planning History, p.s.: Planning is Non-Trivial!

- **Example 3.3.** The Sussman anomaly is a simple blocksworld planning problem:



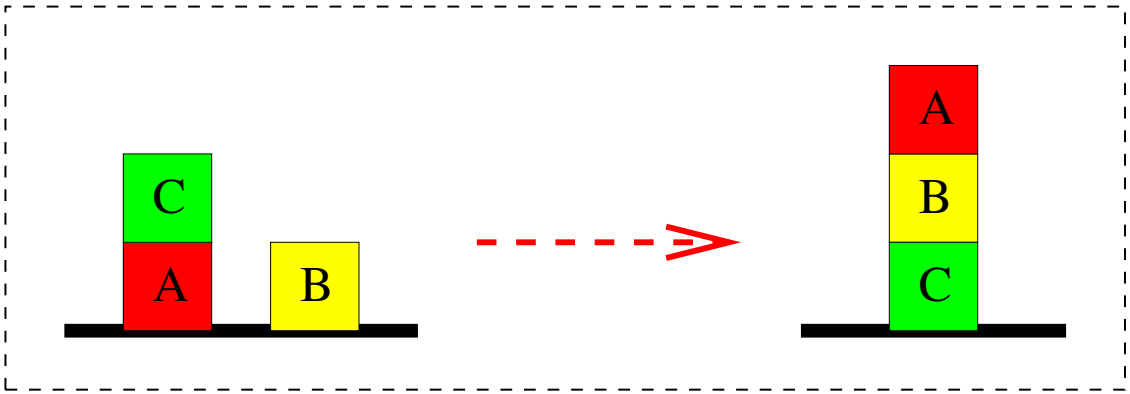
Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ and $\mathrm{on}(B, C)$ fail:

- If we pursue $\mathrm{on}(A, B)$ by unstacking C, and moving A onto B, we achieve the first subgoal, but cannot achieve the second without undoing the first.



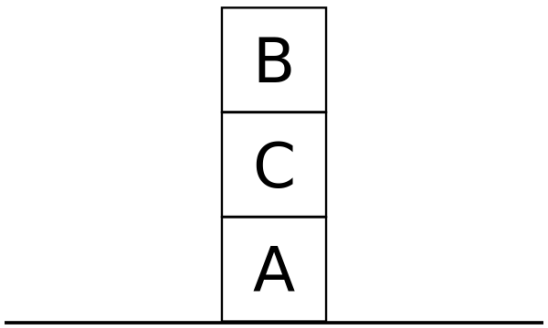
Planning History, p.s.: Planning is Non-Trivial!

- **Example 3.4.** The Sussman anomaly is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ and $\mathrm{on}(B, C)$ fail:

- If we pursue $\mathrm{on}(A, B)$ by unstacking C, and moving A onto B, we achieve the first subgoal, but cannot achieve the second without undoing the first.
- If we pursue $\mathrm{on}(B, C)$ by moving B onto C, we achieve the second subgoal, but cannot achieve the first without undoing the second.



17.4 The STRIPS Planning Formalism

STRIPS Planning

- **Definition 4.1.** STRIPS = Stanford Research Institute Problem Solver.
  STRIPS is the simplest possible (reasonably expressive) logics based planning language.
- STRIPS has only propositional variables as atomic formulae.
- Its preconditions/effects/goals are as canonical as imaginable:
  - Preconditions, goals: conjunctions of atoms.
  - Effects: conjunctions of literals
- We use the common special-case notation for this simple formalism.
- I'll outline some extensions beyond STRIPS later on, when we discuss PDDL.
- **Historical note:** STRIPS [FN71] was originally a planner (cf. Shakey), whose language actually wasn't quite that simple.

STRIPS Planning: Syntax

- **Definition 4.2.** A STRIPS task is a quadruple $\langle P, A, I, G \rangle$ where:
  - P is a finite set of facts: atomic proposition in PL^0 or $\mathrm{PL}^{\mathrm{nq}}$.
  - A is a finite set of actions; each $a \in A$ is a triple $a = \langle \mathrm{pre}_a, \mathrm{add}_a, \mathrm{del}_a \rangle$ of subsets of P referred to as the action's preconditions, add list, and delete list respectively; we require that $\mathrm{add}_a \cap \mathrm{del}_a = \emptyset$.
  - $I \subseteq P$ is the initial state.
  - $G \subseteq P$ is the goal state.

  We will often give each action $a \in A$ a name (a string), and identify a with that name.
- **Note:** We assume, for simplicity, that every action has cost 1. (Unit costs, cf. ??)

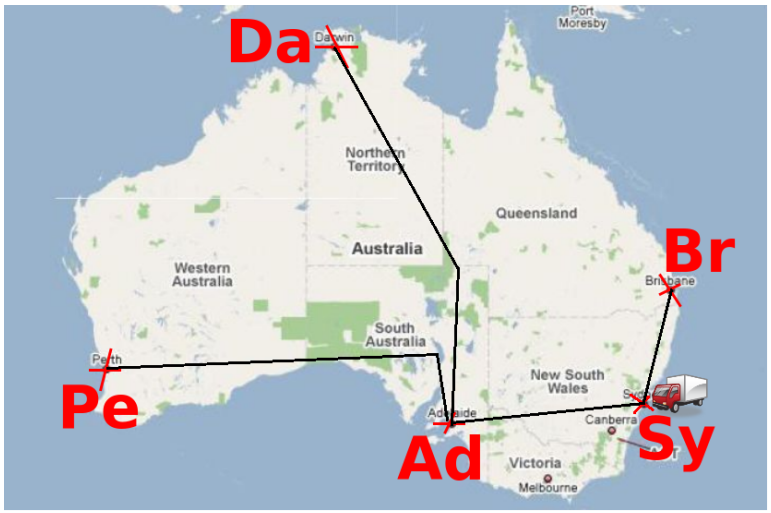
"TSP" in Australia

- **Example 4.3 (Salesman Travelling in Australia).**

Strictly speaking, this is not actually a **TSP** problem instance; simplified/adapted for illustration.

STRIPS Encoding of "TSP"

- **Example 4.4 (continuing).**



- Facts P: $\{\mathrm{at}(x), \mathrm{vis}(x) \mid x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}, \mathrm{Pe}, \mathrm{Da}\}\}$.
- Initial state I: $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
- Goal state G:$\{\mathrm{at}(\mathrm{Sy})\} \cup \{\mathrm{vis}(x) \mid x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}, \mathrm{Pe}, \mathrm{Da}\}\}$.
- Actions $a \in A$: $\mathrm{drv}(x, y)$ where x and y have a road.
  Preconditions pre_a: $\{\mathrm{at}(x)\}$.
  Add list add_a: $\{\mathrm{at}(y), \mathrm{vis}(y)\}$.
  Delete list del_a: $\{\mathrm{at}(x)\}$.
- Plan: $\langle \mathrm{drv}(\mathrm{Sy}, \mathrm{Br}), \mathrm{drv}(\mathrm{Br}, \mathrm{Sy}), \mathrm{drv}(\mathrm{Sy}, \mathrm{Ad}), \mathrm{drv}(\mathrm{Ad}, \mathrm{Pe}), \mathrm{drv}(\mathrm{Pe}, \mathrm{Ad}), \ldots$
  $\ldots, \mathrm{drv}(\mathrm{Ad}, \mathrm{Da}), \mathrm{drv}(\mathrm{Da}, \mathrm{Ad}), \mathrm{drv}(\mathrm{Ad}, \mathrm{Sy})\rangle$

STRIPS Planning: Semantics

- **Idea:** We define a plan for a STRIPS task Π as a solution to an induced search problem Θ_Π. (save work by reduction)
- **Definition 4.5.** Let $\Pi := \langle P, A, I, G\rangle$ be a STRIPS task. The search problem induced by Π is $\Theta_\Pi = \langle S_P, A, T, I, S_G\rangle$ where:
  - The states (also world state) $S_P := \mathcal{P}(P)$ are the subsets of P.
  - A is just Π's action. (so we can define plans easily)
  - The transition model T_A is $\{s \xrightarrow{a} \mathrm{apply}(s,a) \mid \mathrm{pre}_a \subseteq s\}$.
    If $\mathrm{pre}_a \subseteq s$, then $a \in A$ is applicable in s and $\mathrm{apply}(s,a) := (s \cup \mathrm{add}_a)\backslash \mathrm{del}_a$. If $\mathrm{pre}_a \not\subseteq s$, then $\mathrm{apply}(s,a)$ is undefined.
  - I is Π's initial state.
  - The goal states $S_G = \{s \in S_P \mid G \subseteq s\}$ are those that satisfy Π's goal state.

  An (optimal) plan for Π is an (optimal) solution for Θ_Π, i.e., a path from s to some $s' \in S_G$. Π is solvable if a plan for Π exists.
- **Definition 4.6.** For a plan $a = \langle a_1, \ldots, a_n\rangle$, we define

$$\mathrm{apply}(s,a) := \mathrm{apply}(\ldots \mathrm{apply}(\mathrm{apply}(s,a_1),a_2)\ldots,a_n)$$

  if each a_i is applicable in the respective state; else, $\mathrm{apply}(s,a)$ is undefined.

STRIPS Encoding of Simplified TSP

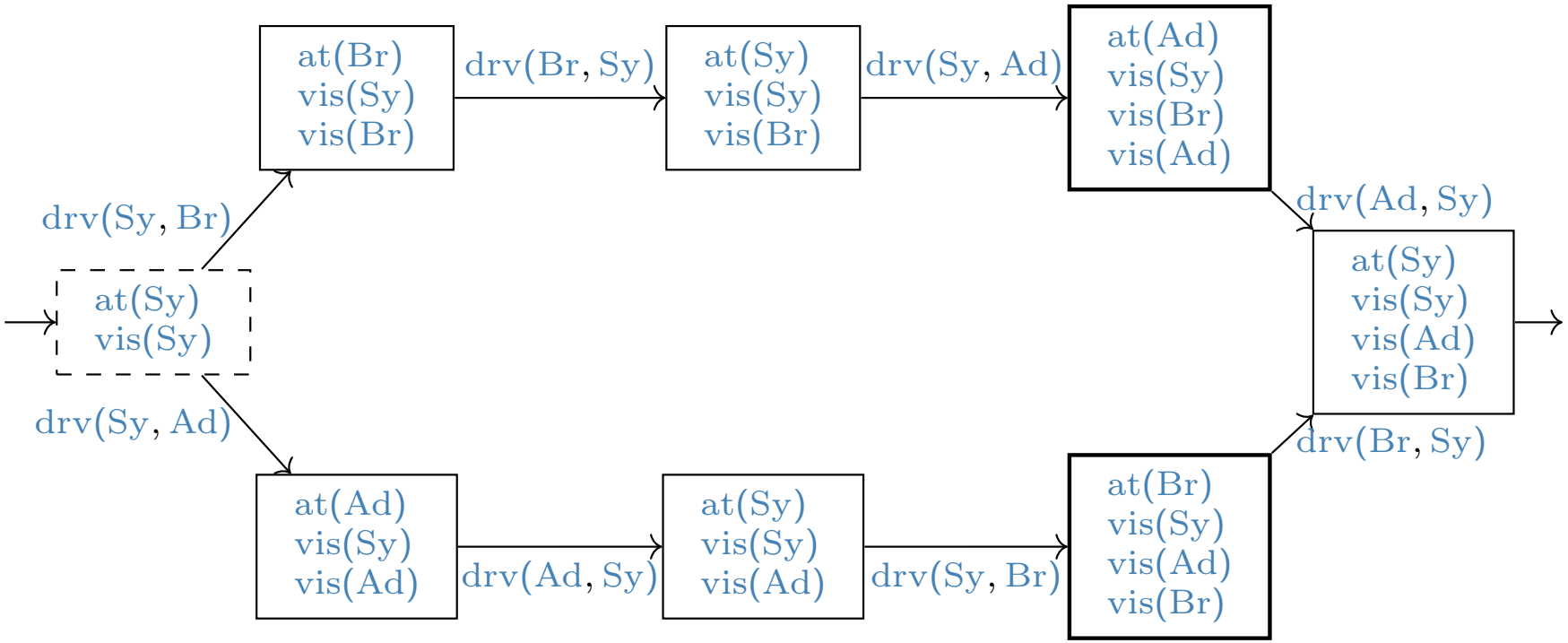
- **Example 4.7 (Simplified traveling salesman problem in Australia).**

Let TSP_- be the STRIPS task, $\langle P, A, I, G\rangle$, where

- Facts P: $\{\mathrm{at}(x), \mathrm{vis}(x) \mid x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}\}\}$.
- Initial state state I: $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
- Goal state G: $\{\mathrm{vis}(x) \mid x \in \{\mathrm{Sy}, \mathrm{Ad}, \mathrm{Br}\}\}$ (note: $\mathrm{noat}(\mathrm{Sy})$)
- Actions A: $a \in A$: $\mathrm{drv}(x, y)$ where x y have a road.
  - preconditions pre_a: $\{\mathrm{at}(x)\}$.
  - add list add_a: $\{\mathrm{at}(y), \mathrm{vis}(y)\}$.
  - delete list del_a: $\{\mathrm{at}(x)\}$.

Questionaire: State Space of TSP_

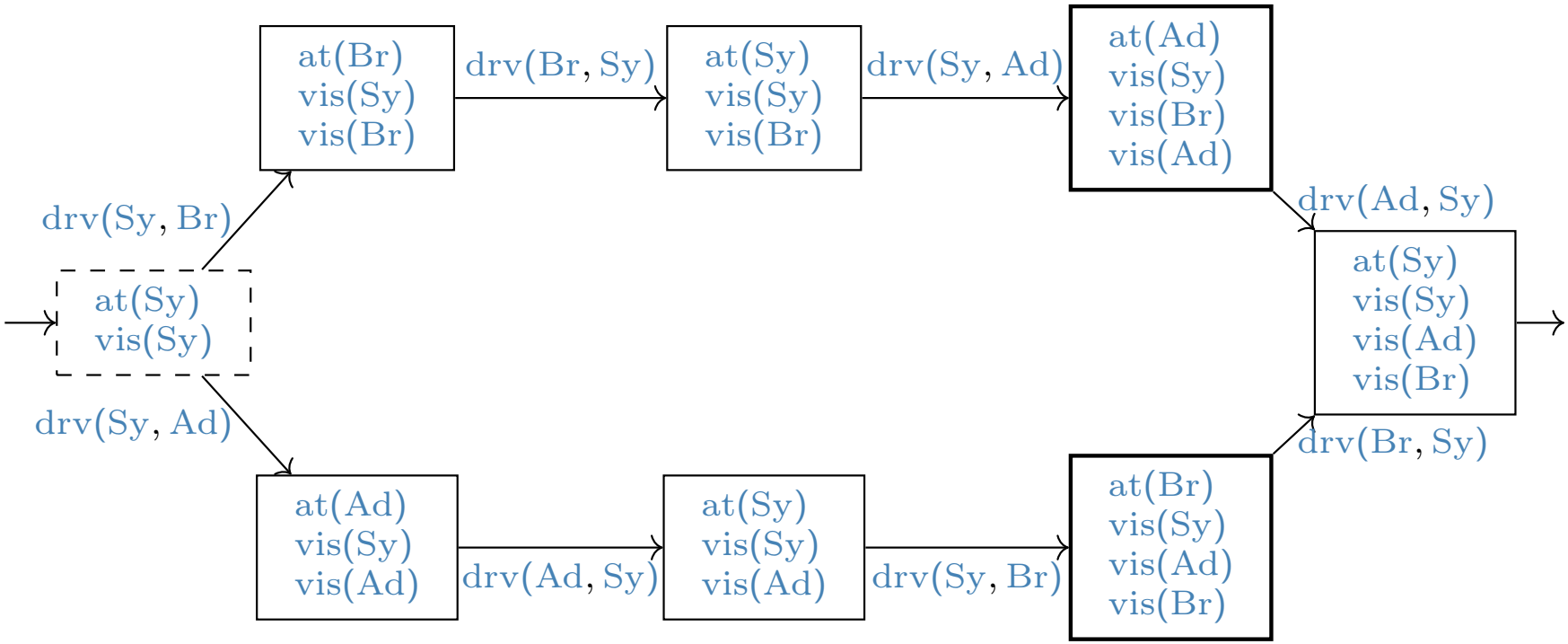
- The state space of the search problem $\Theta_{\text{TSP\_}}$ induced by TSP_ from **??** is



- **Question:** Are there any plans for TSP_ in this graph?

Questionaire: State Space of TSP_

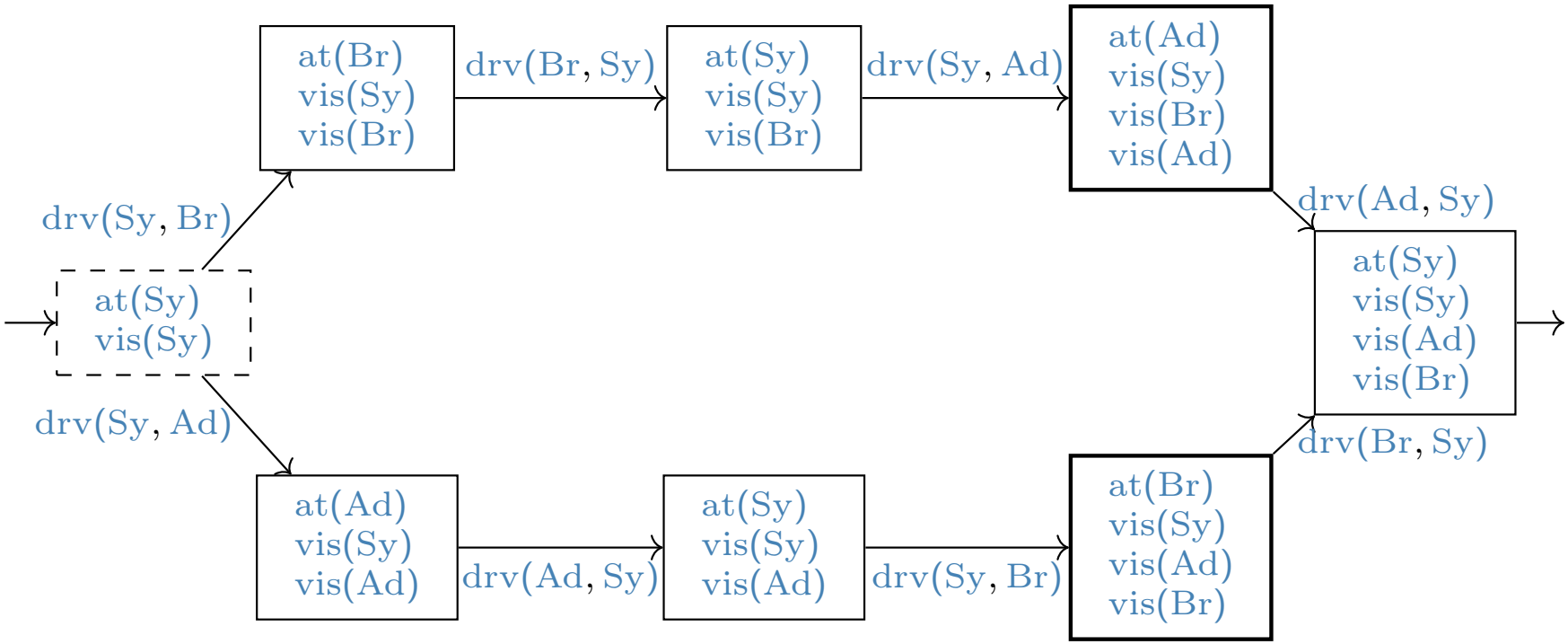
- The state space of the search problem $\Theta_{\text{TSP\_}}$ induced by TSP_ from **??** is



- **Question:** Are there any plans for TSP_ in this graph?
- **Answer:** Yes, two – plans for TSP_ are solutions for $\Theta_{\text{TSP\_}}$, dashed node $\widehat{=} I$, thick nodes $\widehat{=} G$:
  - $\text{drv}(\text{Sy}, \text{Br}), \text{drv}(\text{Br}, \text{Sy}), \text{drv}(\text{Sy}, \text{Ad})$ (upper path)
  - $\text{drv}(\text{Sy}, \text{Ad}), \text{drv}(\text{Ad}, \text{Sy}), \text{drv}(\text{Sy}, \text{Br})$. (lower path)

Questionaire: State Space of TSP_

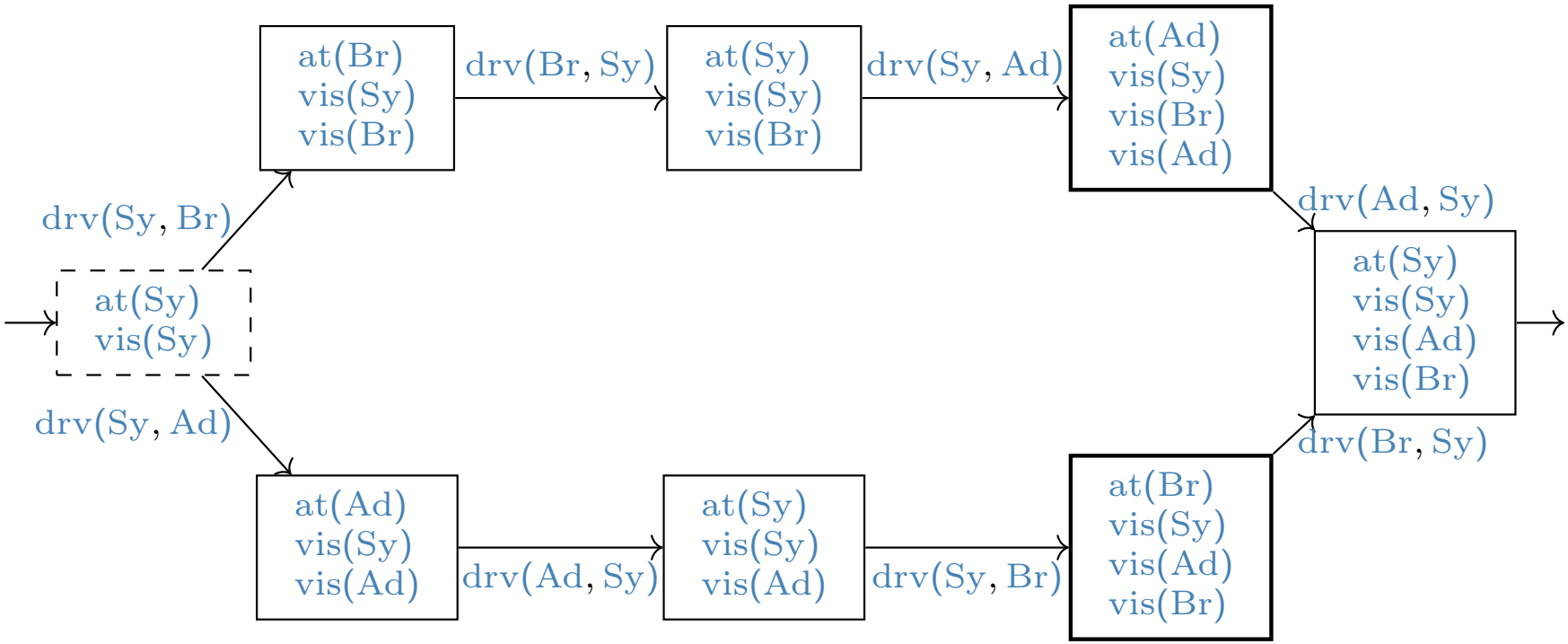
- The state space of the search problem $\Theta_{\text{TSP\_}}$ induced by TSP_ from **??** is



- **Question:** Are there any plans for TSP_ in this graph?
- **Answer:** Yes, two – plans for TSP_ are solutions for $\Theta_{\text{TSP\_}}$, dashed node $\widehat{=}$ I, thick nodes $\widehat{=}$ G:
  - $\text{drv}(\text{Sy}, \text{Br}), \text{drv}(\text{Br}, \text{Sy}), \text{drv}(\text{Sy}, \text{Ad})$ (upper path)
  - $\text{drv}(\text{Sy}, \text{Ad}), \text{drv}(\text{Ad}, \text{Sy}), \text{drv}(\text{Sy}, \text{Br})$. (lower path)
- **Question:** Is the graph above actually the state space induced by ?

Questionaire: State Space of TSP_

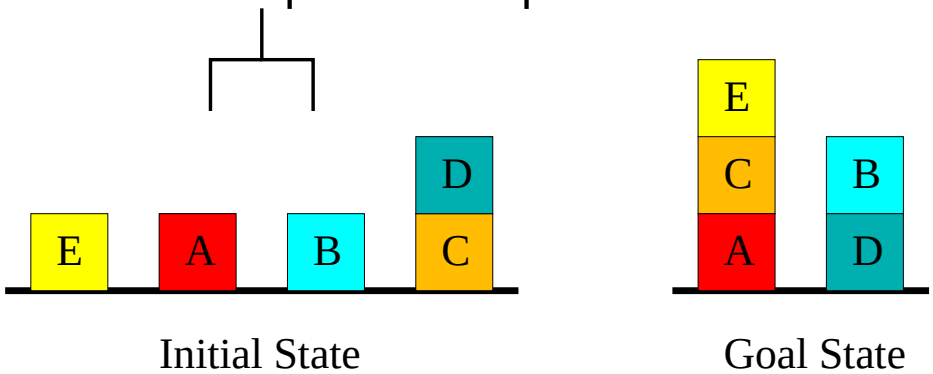
- The state space of the search problem $\Theta_{\text{TSP\_}}$ induced by TSP_ from **??** is



- **Question:** Are there any plans for TSP_ in this graph?
- **Answer:** Yes, two – plans for TSP_ are solutions for $\Theta_{\text{TSP\_}}$, dashed node $\widehat{=}$ I, thick nodes $\widehat{=}$ G:
  - $\text{drv}(\text{Sy}, \text{Br}), \text{drv}(\text{Br}, \text{Sy}), \text{drv}(\text{Sy}, \text{Ad})$ (upper path)
  - $\text{drv}(\text{Sy}, \text{Ad}), \text{drv}(\text{Ad}, \text{Sy}), \text{drv}(\text{Sy}, \text{Br})$. (lower path)
- **Question:** Is the graph above actually the state space induced by ?
- **Answer:** No, only the part reachable from I. The state space of $\Theta_{\text{TSP\_}}$ also includes e.g. the states $\{\text{vis}(\text{Sy})\}$ and $\{\text{at}(\text{Sy}), \text{at}(\text{Br})\}$.

The Blocksworld

- **Definition 4.8.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.9.**

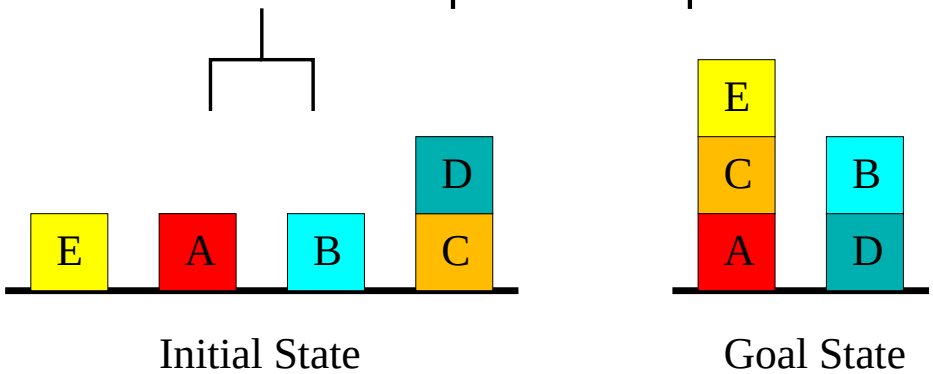


Initial State Goal State

  - Facts: $\mathrm{on}(x, y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.

The Blocksworld

- **Definition 4.10.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.11.**

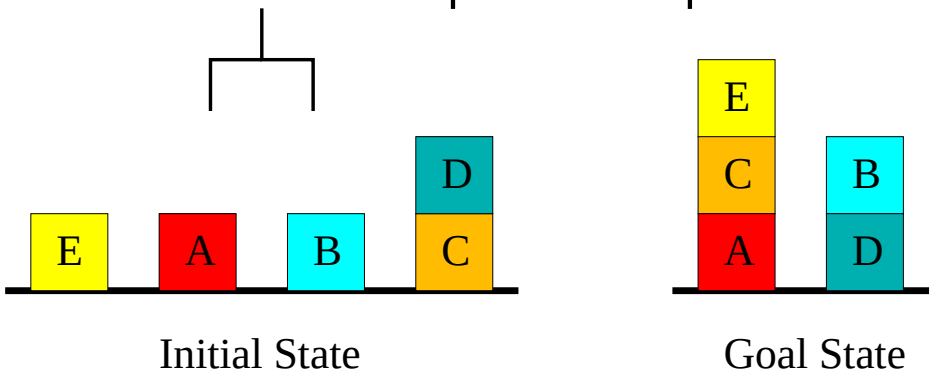


Initial State Goal State

  - Facts: $\mathrm{on}(x, y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.
  - initial state:
    $\{\mathrm{onTable}(E), \mathrm{clear}(E), \ldots, \mathrm{onTable}(C), \mathrm{on}(D, C), \mathrm{clear}(D), \mathrm{armEmpty}\}$.

The Blocksworld

- **Definition 4.12.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.13.**

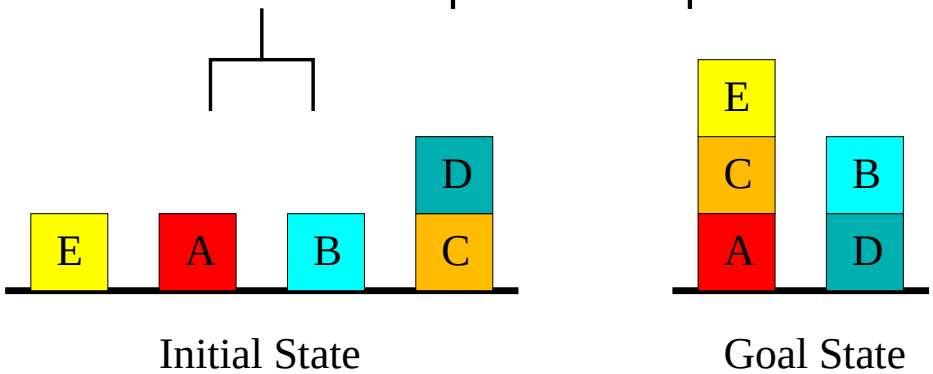


Initial State Goal State

  - Facts: $\mathrm{on}(x, y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.
  - initial state:
    $\{\mathrm{onTable}(E), \mathrm{clear}(E), \ldots, \mathrm{onTable}(C), \mathrm{on}(D, C), \mathrm{clear}(D), \mathrm{armEmpty}\}$.
  - Goal state: $\{\mathrm{on}(E, C), \mathrm{on}(C, A), \mathrm{on}(B, D)\}$.

The Blocksworld

- **Definition 4.14.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.15.**

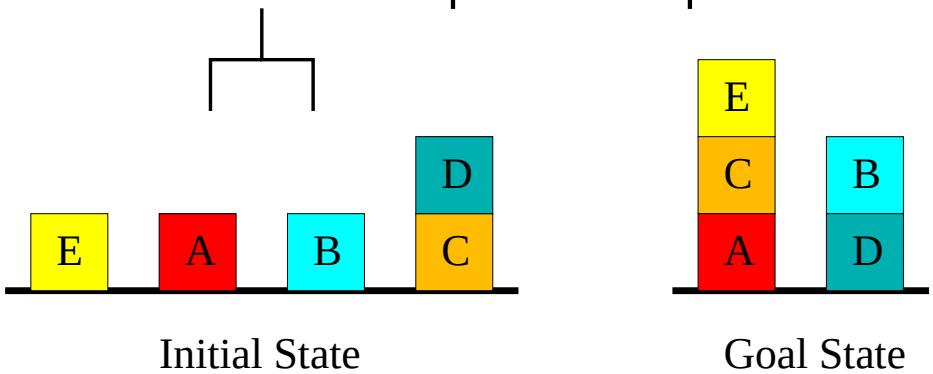


Initial State Goal State

  - Facts: $\mathrm{on}(x, y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.
  - initial state:
    $\{\mathrm{onTable}(E), \mathrm{clear}(E), \ldots, \mathrm{onTable}(C), \mathrm{on}(D, C), \mathrm{clear}(D), \mathrm{armEmpty}\}$.
  - Goal state: $\{\mathrm{on}(E, C), \mathrm{on}(C, A), \mathrm{on}(B, D)\}$.
  - Actions: $\mathrm{stack}(x, y)$, $\mathrm{unstack}(x, y)$, $\mathrm{putdown}(x)$, $\mathrm{pickup}(x)$.

The Blocksworld

- **Definition 4.16.** The blocks world is a simple planning domain: a set of wooden blocks of various shapes and colors sit on a table. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block.
- **Example 4.17.**



Initial State Goal State

  - Facts: $\mathrm{on}(x,y)$, $\mathrm{onTable}(x)$, $\mathrm{clear}(x)$, $\mathrm{holding}(x)$, $\mathrm{armEmpty}$.
  - initial state:
    $\{\mathrm{onTable}(E), \mathrm{clear}(E), \ldots, \mathrm{onTable}(C), \mathrm{on}(D,C), \mathrm{clear}(D), \mathrm{armEmpty}\}$.
  - Goal state: $\{\mathrm{on}(E,C), \mathrm{on}(C,A), \mathrm{on}(B,D)\}$.
  - Actions: $\mathrm{stack}(x,y)$, $\mathrm{unstack}(x,y)$, $\mathrm{putdown}(x)$, $\mathrm{pickup}(x)$.
  - $\mathrm{stack}(x,y)$?
    $\mathrm{pre} : \{\mathrm{holding}(x), \mathrm{clear}(y)\}$
    $\mathrm{add} : \{\mathrm{on}(x,y), \mathrm{armEmpty}, \mathit{clearx}\}$
    $\mathrm{del} : \{\mathrm{holding}(x), \mathrm{clear}(y)\}$.

STRIPS for the Blocksworld

- **Question:** Which are correct encodings (ones that are part of some correct overall model) of the STRIPS Blocksworld $\mathrm{pickup}(x)$ action schema?

(A)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x)\}$

(B)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{armEmpty}\}$

(C)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x), \mathrm{armEmpty}, \mathrm{clear}(x)\}$

(D)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x), \mathrm{armEmpty}\}$

Recall: an actions a represented by a tuple $\langle \mathrm{pre}_a, \mathrm{add}_a, \mathrm{del}_a \rangle$ of lists of facts.

- **Hint:** The only differences between them are the delete lists

STRIPS for the Blocksworld

- **Question:** Which are correct encodings (ones that are part of some correct overall model) of the STRIPS Blocksworld $\mathrm{pickup}(x)$ action schema?

(A)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x)\}$

(B)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{armEmpty}\}$

(C)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x), \mathrm{armEmpty}, \mathrm{clear}(x)\}$

(D)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x), \mathrm{armEmpty}\}$

Recall: an actions a represented by a tuple $\langle \mathrm{pre}_a, \mathrm{add}_a, \mathrm{del}_a \rangle$ of lists of facts.

- **Hint:** The only differences between them are the delete lists
- **Answer:**

(A) No, must delete $\mathrm{armEmpty}$

STRIPS for the Blocksworld

- **Question:** Which are correct encodings (ones that are part of some correct overall model) of the STRIPS Blocksworld $\mathrm{pickup}(x)$ action schema?

(A)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x)\}$

(B)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{armEmpty}\}$

(C)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x), \mathrm{armEmpty}, \mathrm{clear}(x)\}$

(D)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x), \mathrm{armEmpty}\}$

Recall: an actions a represented by a tuple $\langle \mathrm{pre}_a, \mathrm{add}_a, \mathrm{del}_a \rangle$ of lists of facts.

- **Hint:** The only differences between them are the delete lists
- **Answer:**
  - (A) No, must delete $\mathrm{armEmpty}$
  - (B) No, must delete $\mathrm{onTable}(x)$.

STRIPS for the Blocksworld

- **Question:** Which are correct encodings (ones that are part of some correct overall model) of the STRIPS Blocksworld $\mathrm{pickup}(x)$ action schema?

(A)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x)\}$

(B)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{armEmpty}\}$

(C)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x), \mathrm{armEmpty}, \mathrm{clear}(x)\}$

(D)
$\{\mathrm{onTable}(x), \mathrm{clear}(x), \mathrm{armEmpty}\}$
$\{\mathrm{holding}(x)\}$
$\{\mathrm{onTable}(x), \mathrm{armEmpty}\}$

Recall: an actions a represented by a tuple $\langle \mathrm{pre}_a, \mathrm{add}_a, \mathrm{del}_a \rangle$ of lists of facts.

- **Hint:** The only differences between them are the delete lists
- **Answer:**

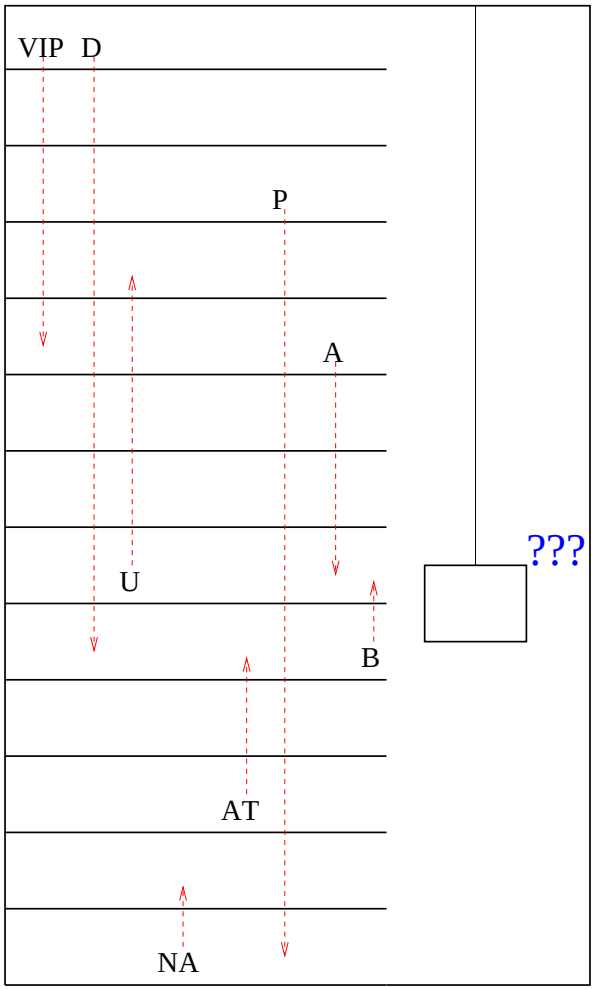
(A) No, must delete $\mathrm{armEmpty}$

(B) No, must delete $\mathrm{onTable}(x)$.

(C) (D) Both yes: We can, but don't have to, encode the *single-arm* Blocksworld so that the block currently in the hand is not clear.
For (C), $\mathrm{stack}(x, y)$ and $\mathrm{putdown}(x)$ need to add $\mathrm{clear}(x)$, so the encoding on the previous slide does not work.

Miconic-10: A Real-World Example

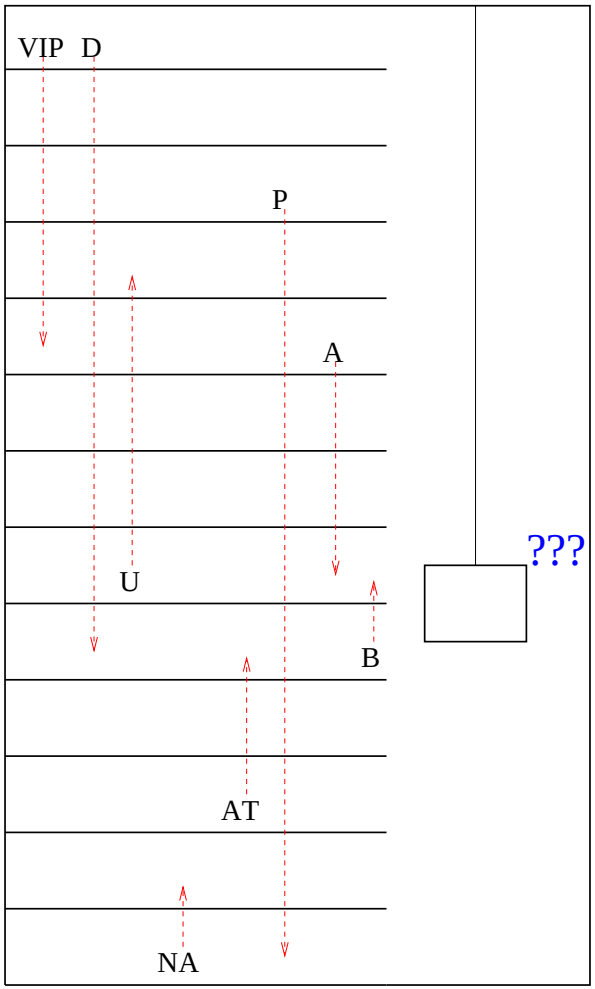
- **Example 4.18.** Elevator control as a planning problem; details at [KS00]
  Specify mobility needs before boarding, let a planner schedule/otimize trips



- VIP:
- D:
- NA:
- AT:
- A, B:
- P:

Miconic-10: A Real-World Example

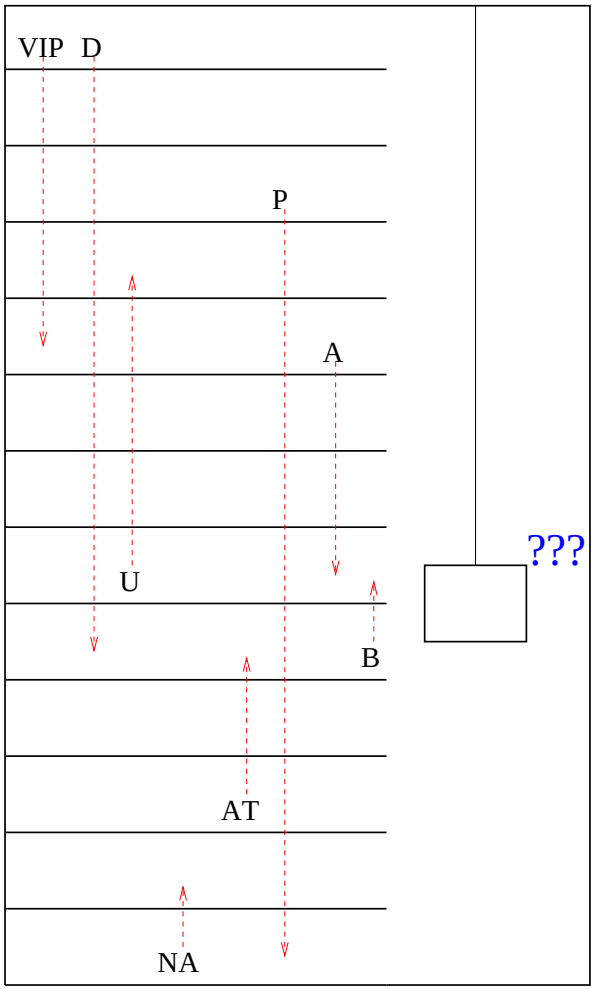
- **Example 4.19.** Elevator control as a planning problem; details at [KS00]
  Specify mobility needs before boarding, let a planner schedule/otimize trips



- VIP: Served first.
- D:
- NA:
- AT: .
- A, B:
- P:

Miconic-10: A Real-World Example

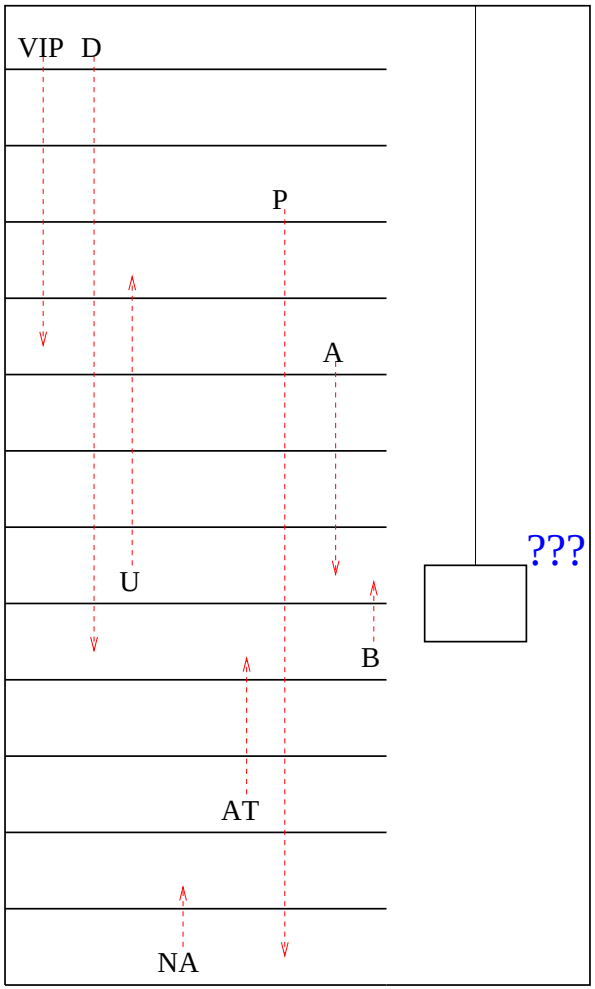
- **Example 4.20.** Elevator control as a planning problem; details at [KS00]
  Specify mobility needs before boarding, let a planner schedule/otimize trips



- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA:
- AT: .
- A, B:
- P:

Miconic-10: A Real-World Example

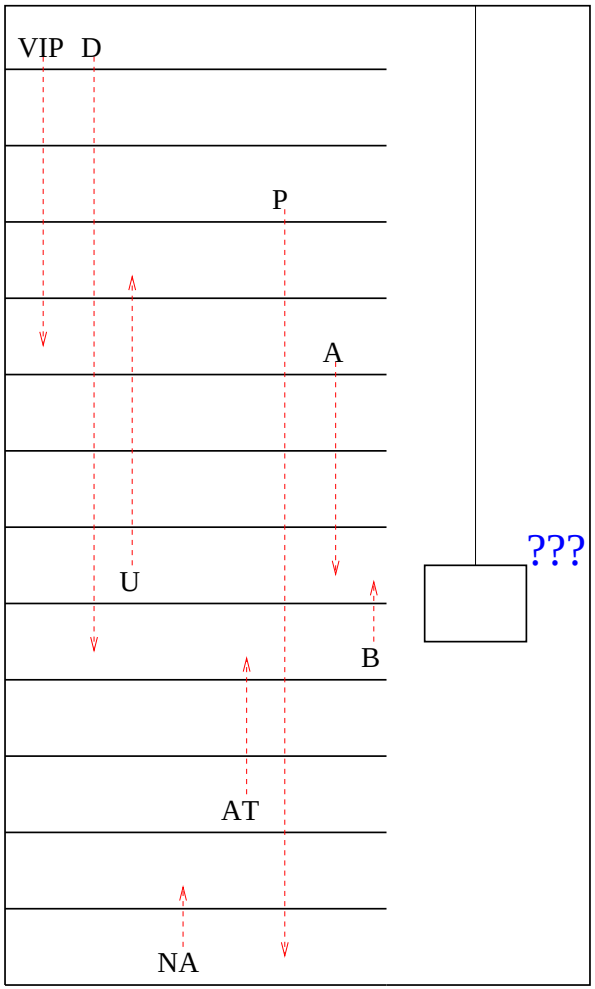
- **Example 4.21.** Elevator control as a planning problem; details at [KS00]
  Specify mobility needs before boarding, let a planner schedule/otimize trips



- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA: Never-alone
- AT: .
- A, B:
- P:

Miconic-10: A Real-World Example

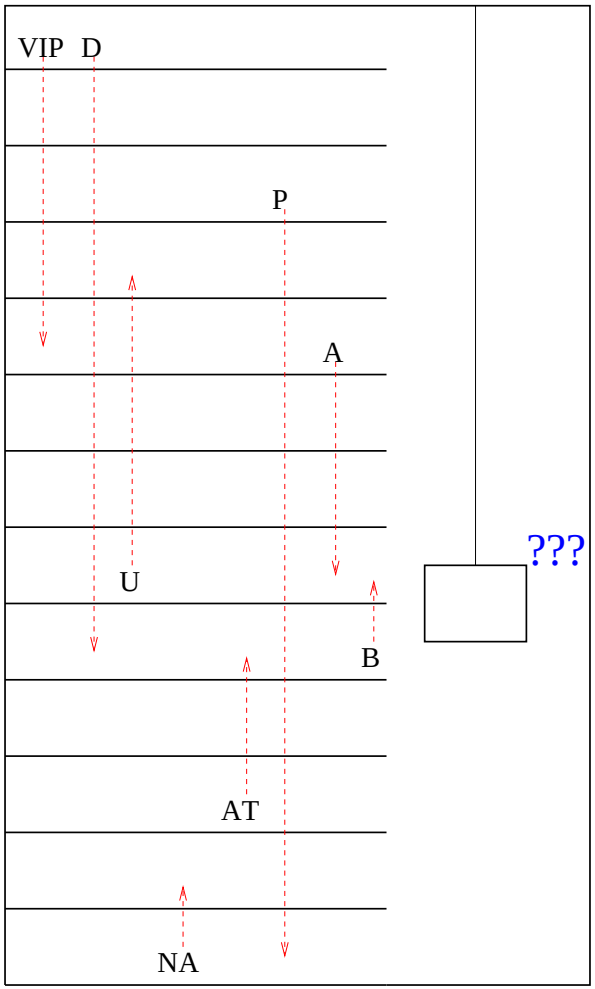
- **Example 4.22.** Elevator control as a planning problem; details at [KS00]
  Specify mobility needs before boarding, let a planner schedule/otimize trips



- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA: Never-alone
- AT: Attendant.
- A, B:
- P:

Miconic-10: A Real-World Example

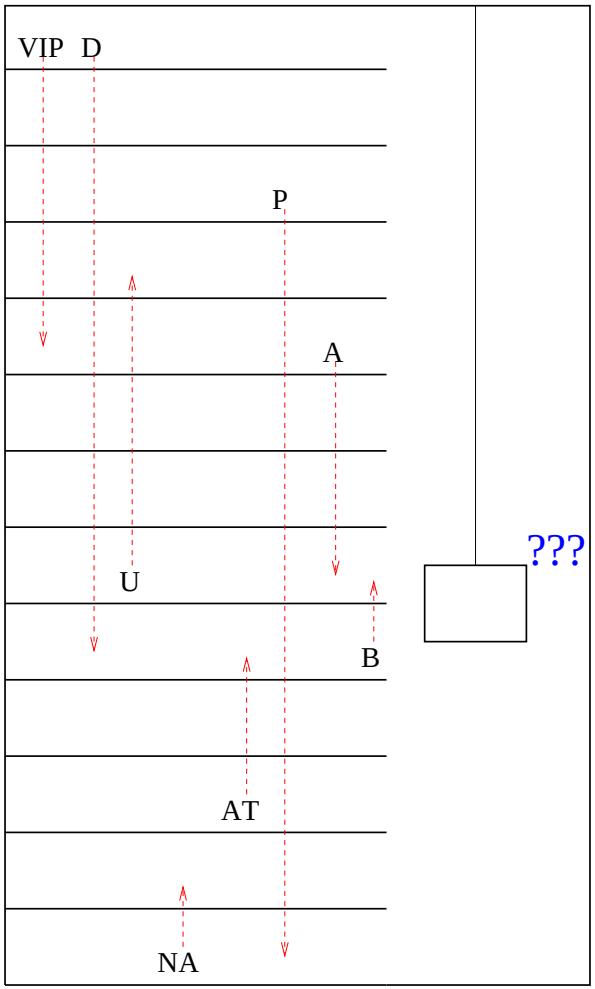
- **Example 4.23.** Elevator control as a planning problem; details at [KS00]
  Specify mobility needs before boarding, let a planner schedule/otimize trips



- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA: Never-alone
- AT: Attendant.
- A, B: Never together in the same elevator
- P:

Miconic-10: A Real-World Example

- **Example 4.24.** Elevator control as a planning problem; details at [KS00]
  Specify mobility needs before boarding, let a planner schedule/otimize trips

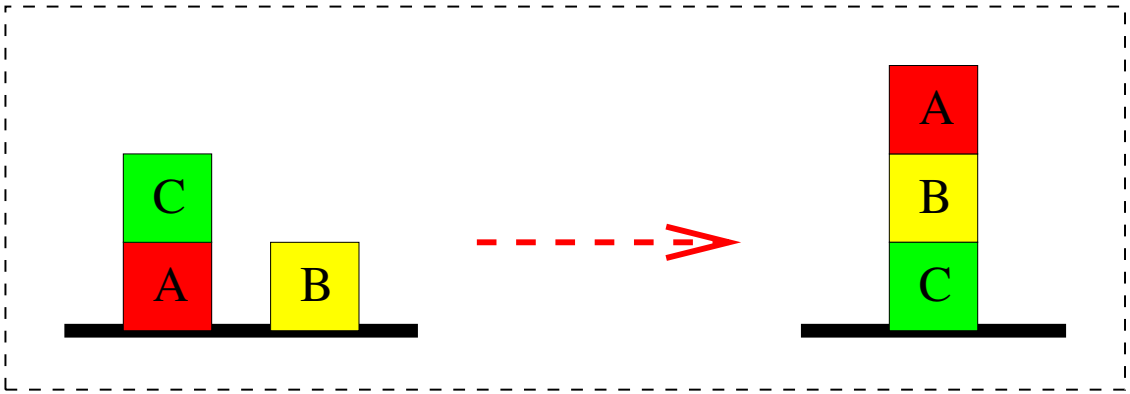


- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA: Never-alone
- AT: Attendant.
- A, B: Never together in the same elevator
- P: Normal passenger

17.5 Partial Order Planning

Planning History, p.s.: Planning is Non-Trivial!

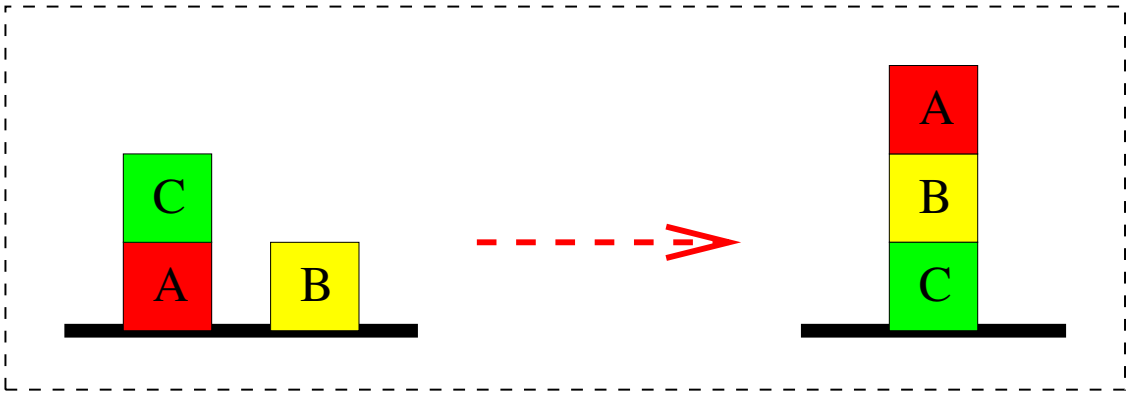
- **Example 5.1.** The Sussman anomaly is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ and $\mathrm{on}(B, C)$ fail:

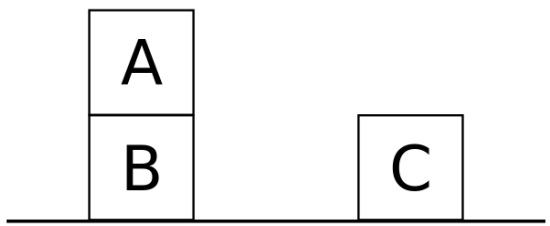
Planning History, p.s.: Planning is Non-Trivial!

- **Example 5.2.** The Sussman anomaly is a simple blocksworld planning problem:



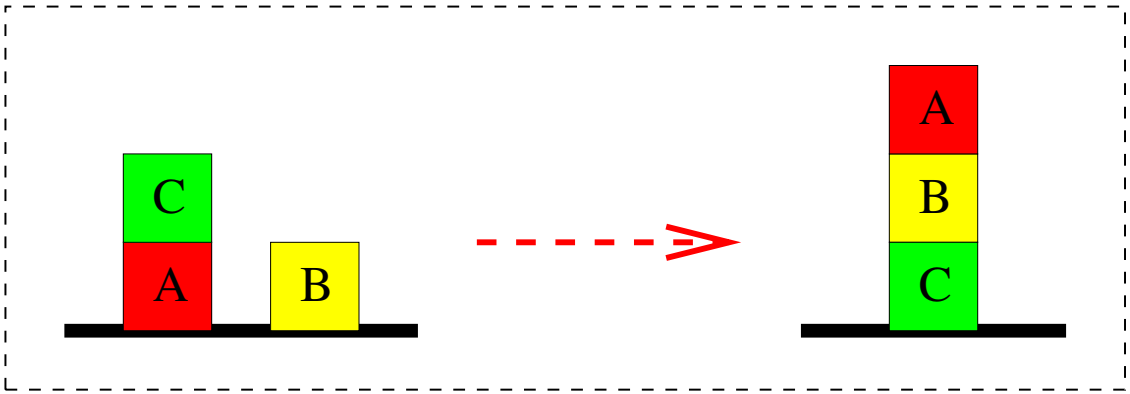
Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ and $\mathrm{on}(B, C)$ fail:

- If we pursue $\mathrm{on}(A, B)$ by unstacking C, and moving A onto B, we achieve the first subgoal, but cannot achieve the second without undoing the first.



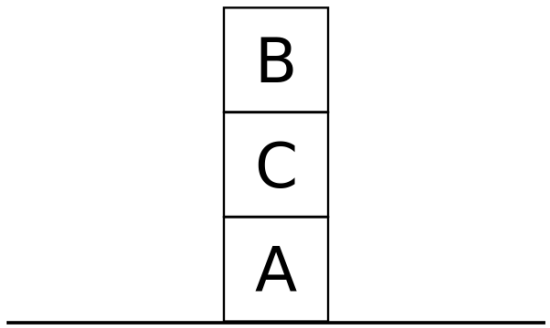
Planning History, p.s.: Planning is Non-Trivial!

- **Example 5.3.** The Sussman anomaly is a simple blocksworld planning problem:



Simple planners that split the goal into subgoals $\mathrm{on}(A, B)$ and $\mathrm{on}(B, C)$ fail:

- If we pursue $\mathrm{on}(A, B)$ by unstacking C, and moving A onto B, we achieve the first subgoal, but cannot achieve the second without undoing the first.
- If we pursue $\mathrm{on}(B, C)$ by moving B onto C, we achieve the second subgoal, but cannot achieve the first without undoing the second.



Partial Order Planning

- **Definition 5.4.** Any algorithm that can place two actions into a plan without specifying which comes first is called as partial order planning.

Partial Order Planning

- **Definition 5.5.** Any algorithm that can place two actions into a plan without specifying which comes first is called as partial order planning.
- **Ideas** for partial order planning:
  - Organize the planning steps in a DAG that supports multiple paths from initial to goal state
    - nodes (steps) are labeled with actions (actions can occur multiply)
    - edges with propositions added by source and presupposed by target

    acyclicity of the graph induces a partial ordering on steps.
  - additional temporal constraints resolve subgoal interactions and induce a linear order.

Partial Order Planning

- **Definition 5.6.** Any algorithm that can place two actions into a plan without specifying which comes first is called as partial order planning.
- **Ideas** for partial order planning:
  - Organize the planning steps in a DAG that supports multiple paths from initial to goal state
    - nodes (steps) are labeled with actions (actions can occur multiply)
    - edges with propositions added by source and presupposed by target

    acyclicity of the graph induces a partial ordering on steps.
  - additional temporal constraints resolve subgoal interactions and induce a linear order.
- **Advantages** of partial order planning:
  - problems can be decomposed $\rightsquigarrow$ can work well with non-cooperative environments.
  - efficient by least-commitment strategy
  - causal links (edges) pinpoint unworkable subplans early.

Partially Ordered Plans

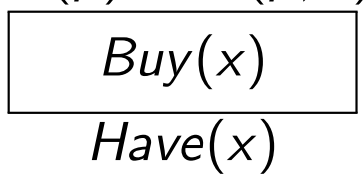
- **Definition 5.7.** Let $\langle P, A, I, G\rangle$ be a STRIPS task, then a partially ordered plan $\mathcal{P} = \langle V, E\rangle$ is a labeled DAG, where the nodes in V (called steps) are labeled with actions from A, or are a
  - start step, which has label "effect" I, or a
  - finish step, which has label "precondition" G.

  Every edge $(S, T) \in E$ is either labeled by:
  - A non-empty set $p \subseteq P$ of facts that are effects of the action of S and the preconditions of that of T. We call such a labeled edge a causal link and write it $S \xrightarrow{p} T$.
  - $\prec$, then call it a temporal constraint and write it as $S \prec T$.

  An open condition is a precondition of a step not yet causally linked.
- **Definition 5.8.** Let Π be a partially ordered plan, then we call a step U possibly intervening in a causal link $S \xrightarrow{p} T$, iff $\Pi \cup \{S \prec U, U \prec T\}$ is acyclic.
- **Definition 5.9.** A precondition is achieved iff it is the effect of an earlier step and no possibly intervening step undoes it.
- **Definition 5.10.** A partially ordered plan Π is called complete iff every precondition is achieved.
- **Definition 5.11.** Partial order planning is the process of computing complete and acyclic partially ordered plans for a given planning task.

A Notation for STRIPS Actions

- **Definition 5.12 (Notation).** In diagrams, we often write STRIPS actions into boxes with preconditions above and effects below.
- **Example 5.13.**
  - Actions: $Buy(x)$
  - Preconditions: $At(p)$, $Sells(p, x)$
  - Effects: $Have(x)$



- **Notation:** A causal link $S \xrightarrow{p} T$ can also be denoted by a direct arrow between the effects p of S and the preconditions p of T in the STRIPS action notation above.
  Show temporal constraints as dashed arrows.

Planning Process

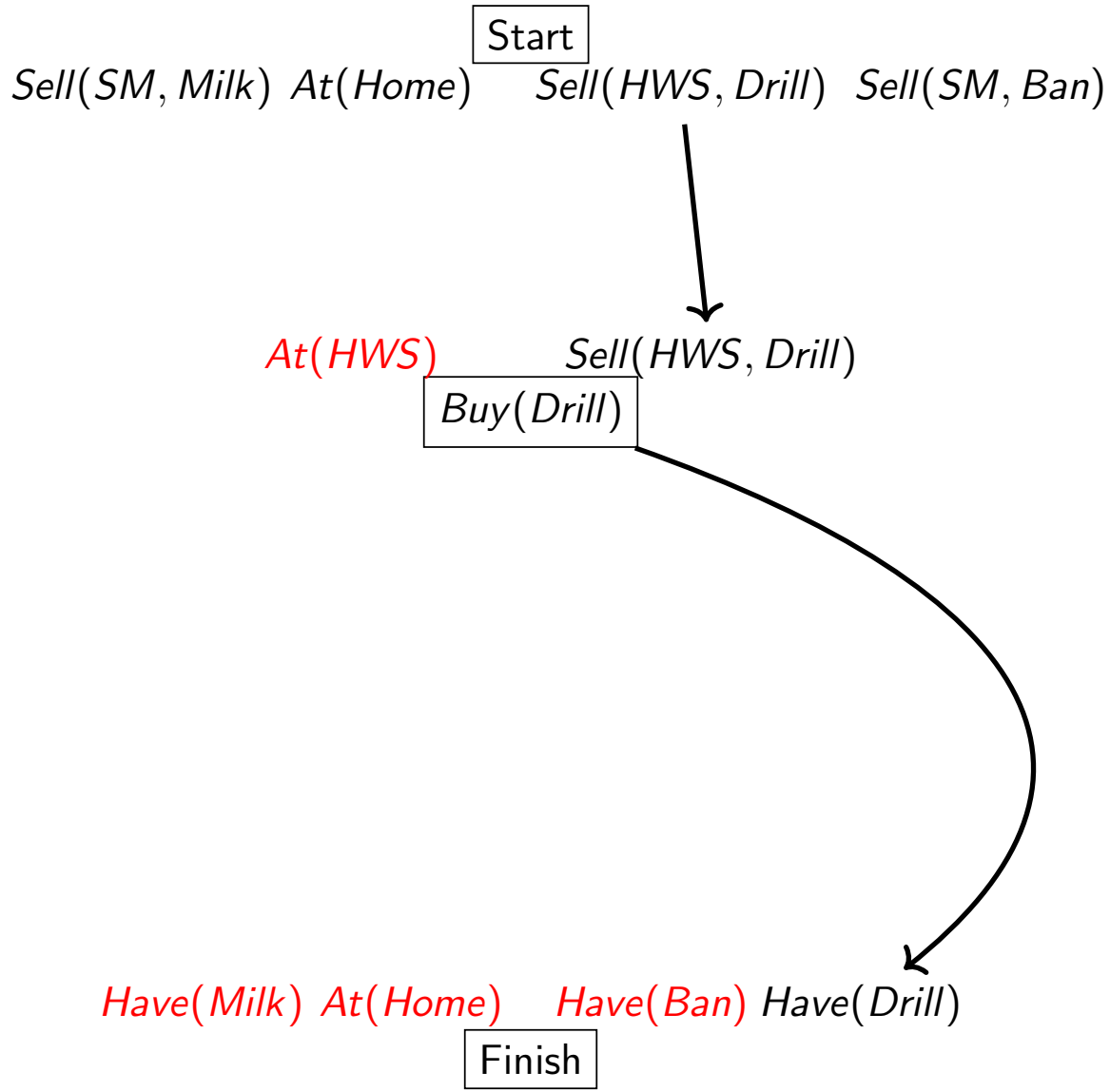
- **Definition 5.14.** Partial order planning is search in the space of partial plans via the following operations:
  - add link from an existing action to an open precondition,
  - add step (an action with links to other steps) to fulfil an open precondition,
  - order one step wrt. another (by adding temporal constraints) to remove possible conflicts.
- **Idea:** Gradually move from incomplete/vague plans to complete, correct plans. backtrack if an open condition is unachievable or if a conflict is unresolvable.

Example: Shopping for Bananas, Milk, and a Cordless Drill

- **Example 5.15.**

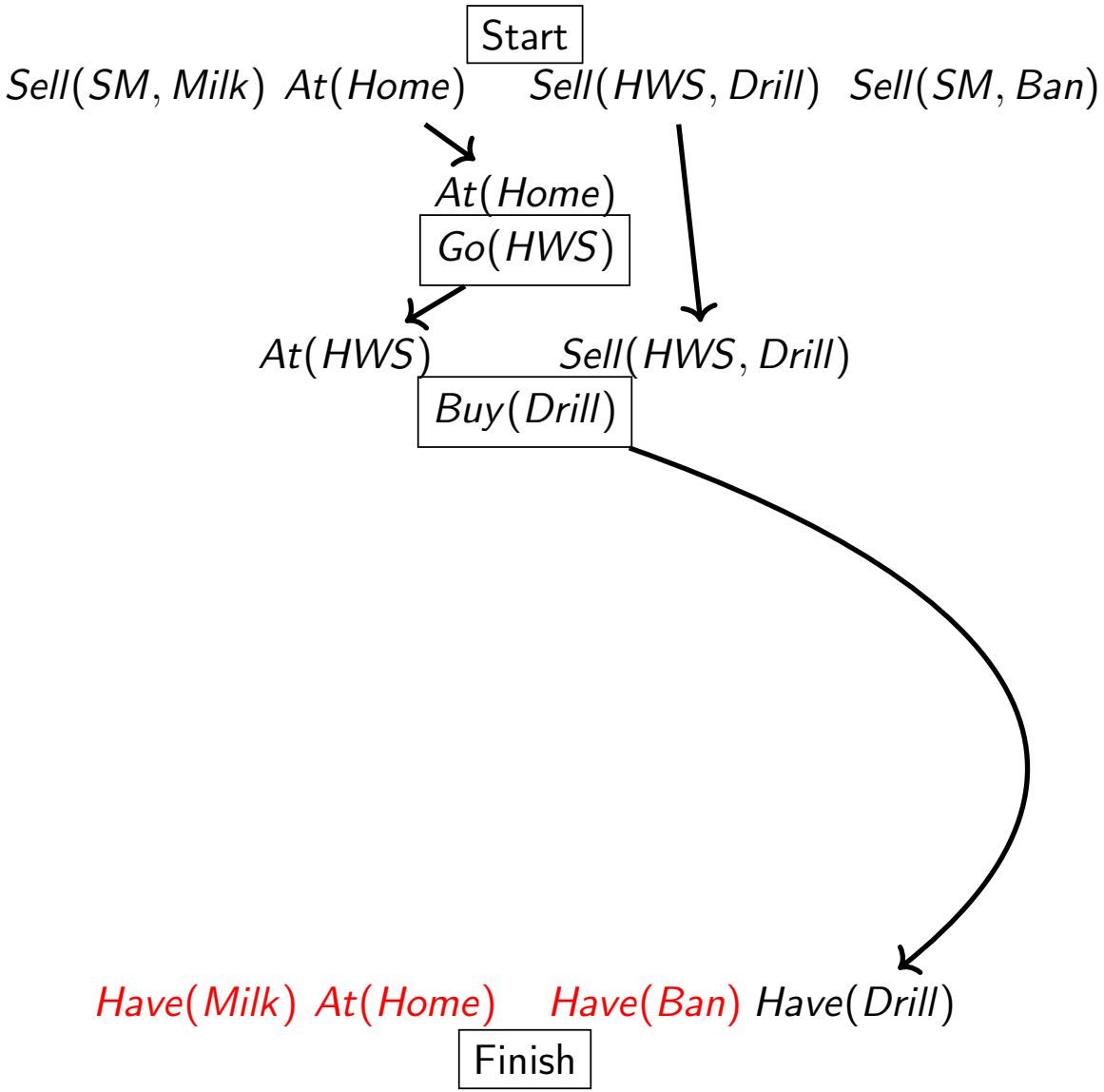
Example: Shopping for Bananas, Milk, and a Cordless Drill

- **Example 5.16.**



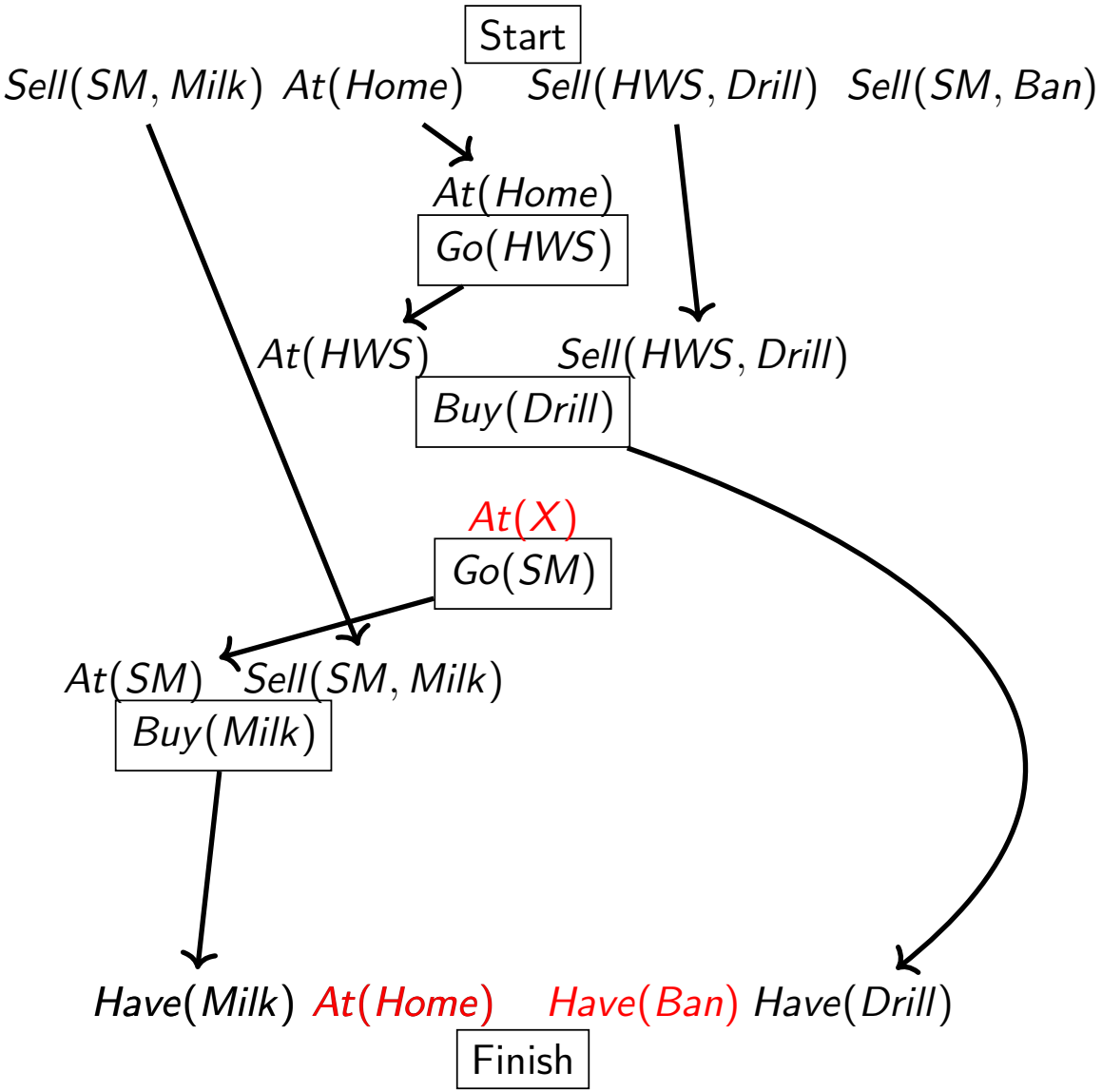
Example: Shopping for Bananas, Milk, and a Cordless Drill

- **Example 5.17.**



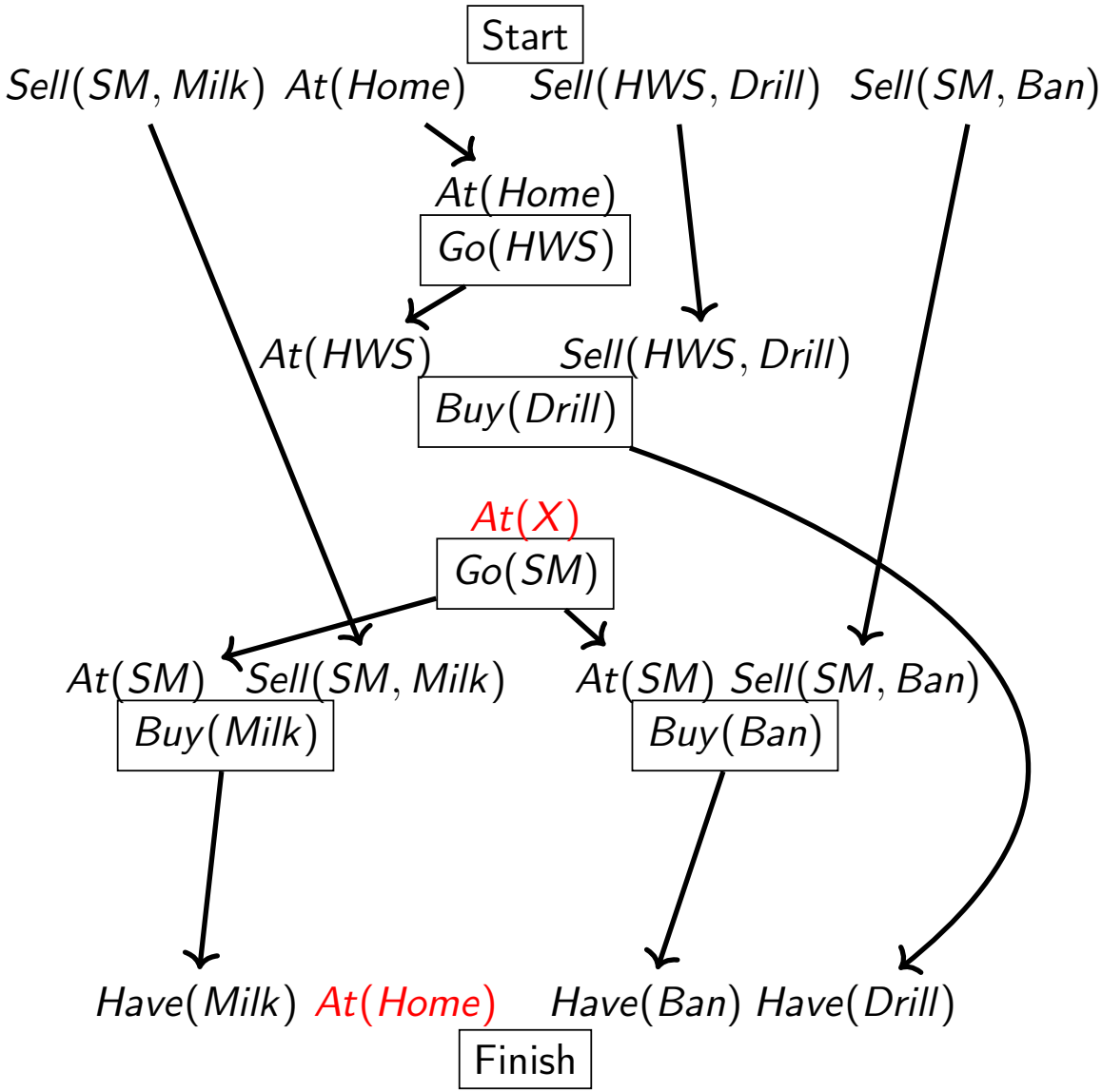
Example: Shopping for Bananas, Milk, and a Cordless Drill

- **Example 5.18.**



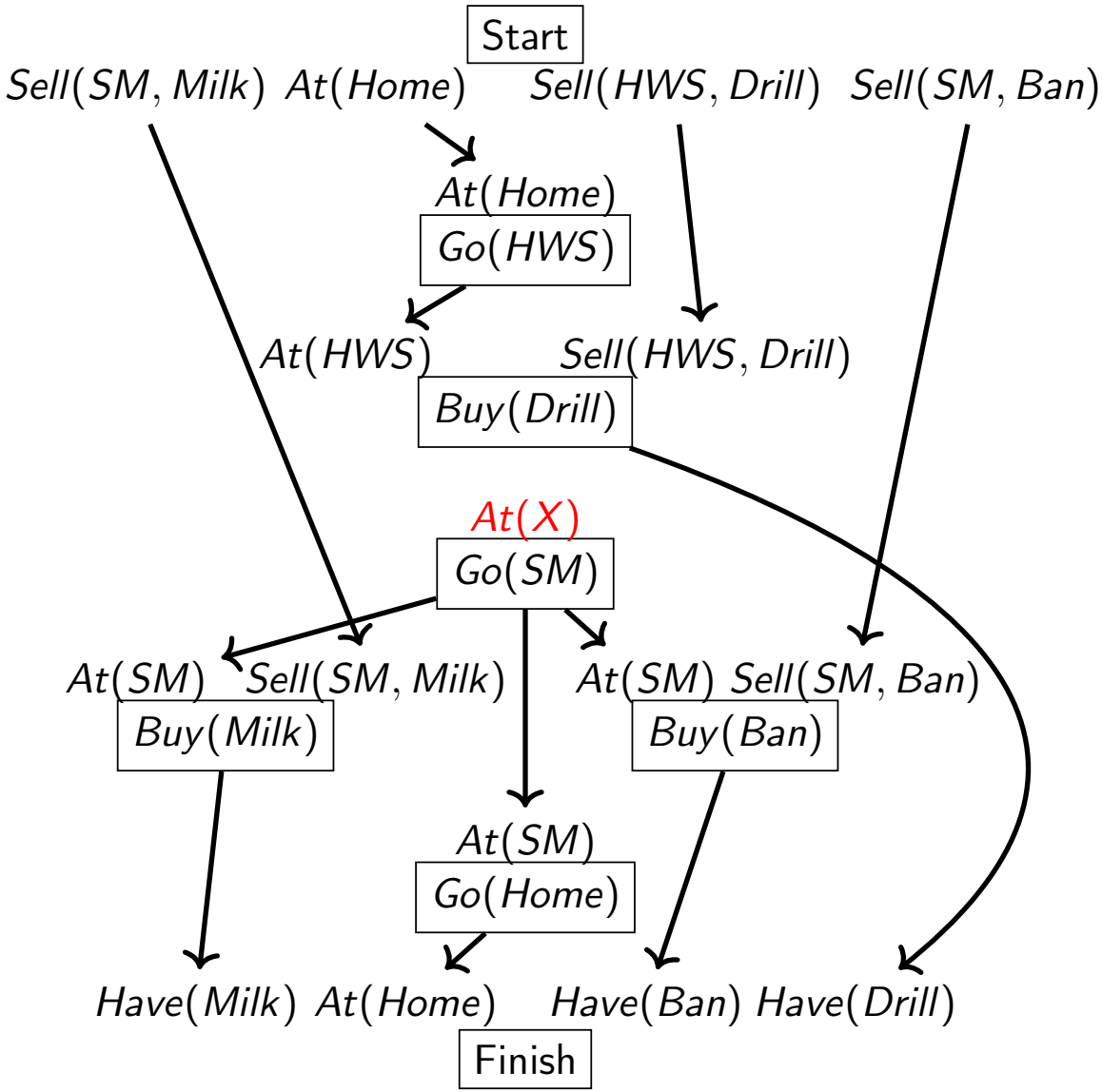
Example: Shopping for Bananas, Milk, and a Cordless Drill

- **Example 5.19.**



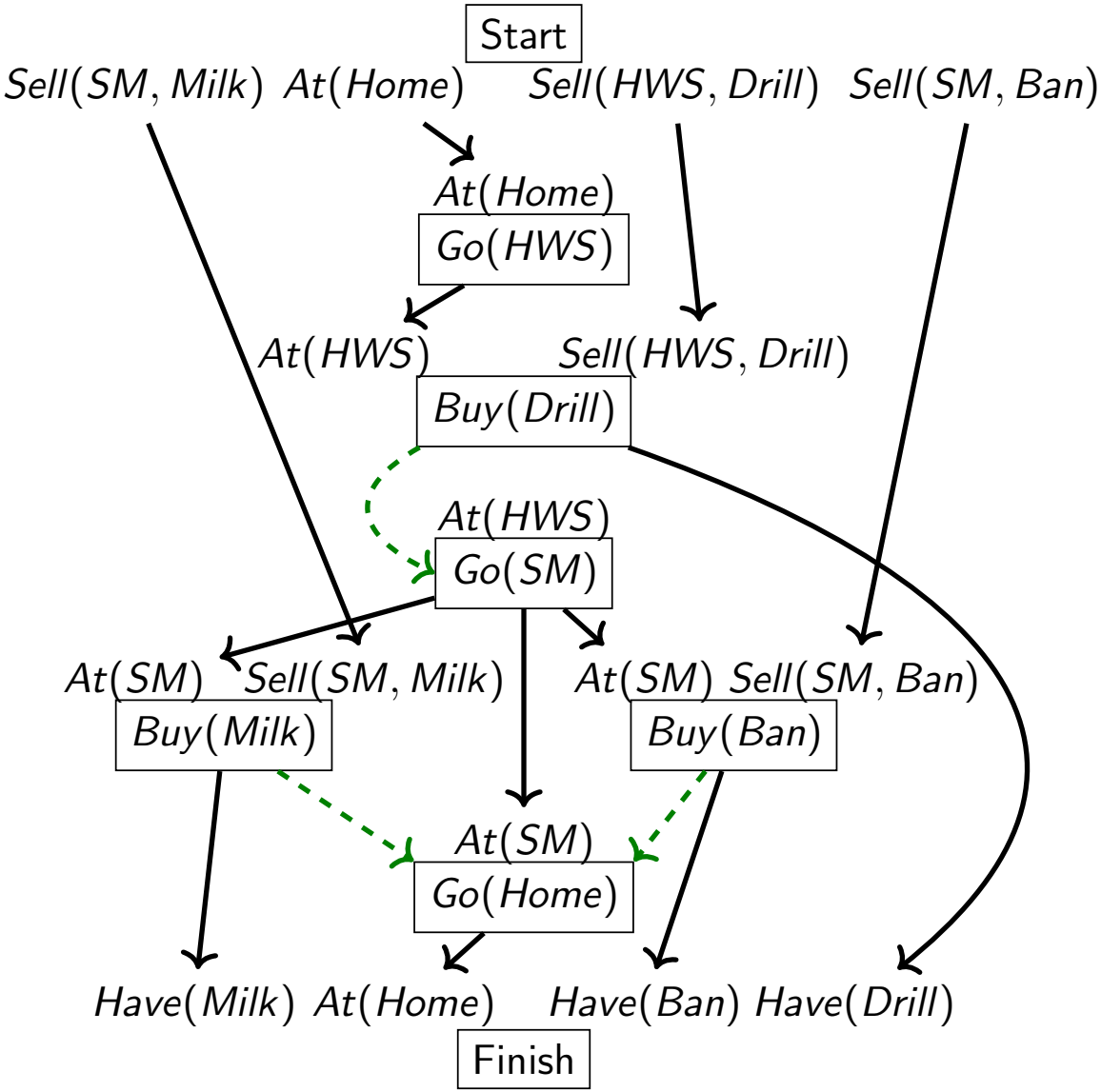
Example: Shopping for Bananas, Milk, and a Cordless Drill

- **Example 5.20.**



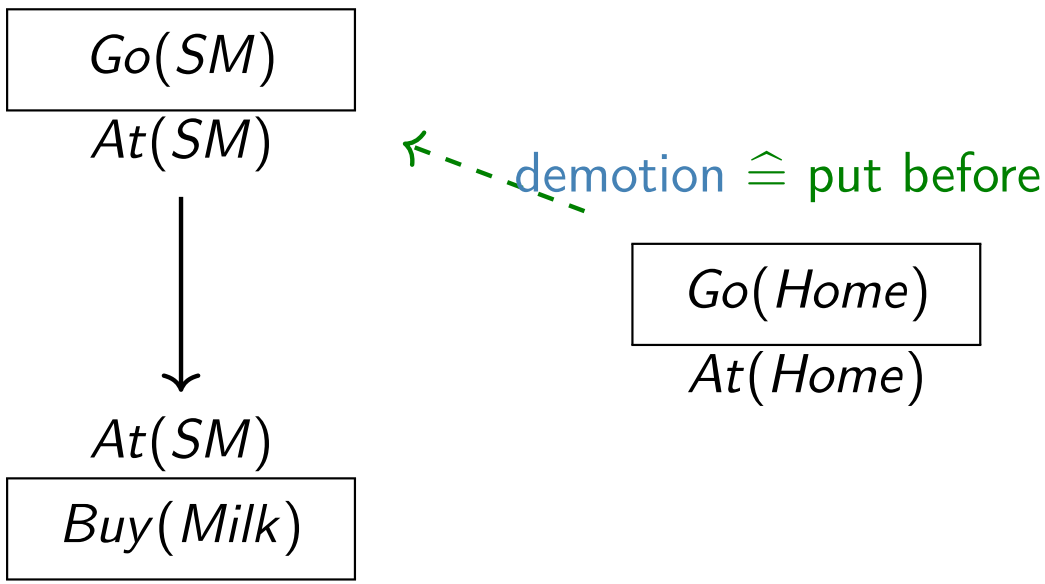
Example: Shopping for Bananas, Milk, and a Cordless Drill

- **Example 5.21.**



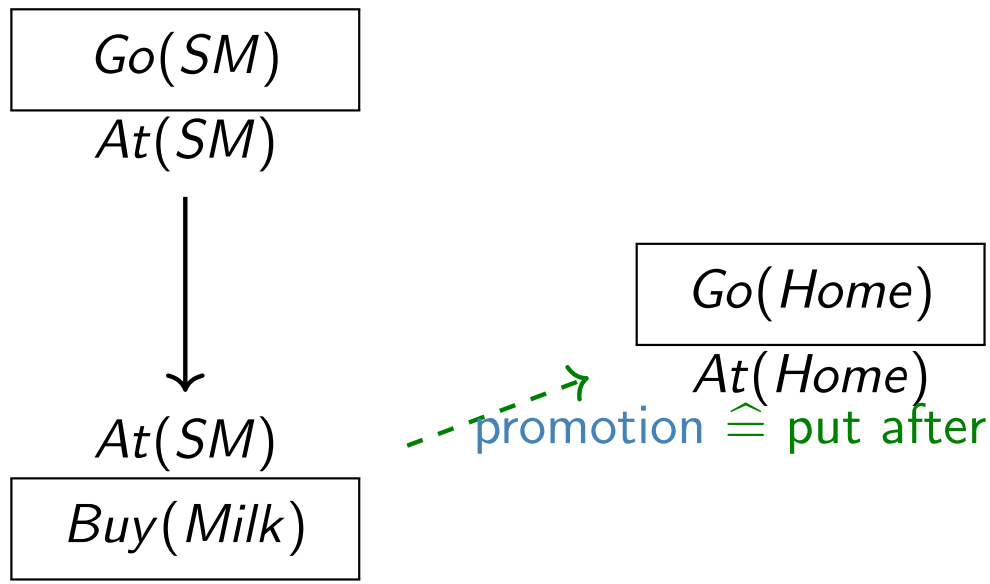
Clobbering and Promotion/Demotion

- **Definition 5.22.** In a partially ordered plan, a step C clobbers a causal link $L := S \xrightarrow{p} T$, iff it destroys the condition p achieved by L.
- **Definition 5.23.** If C clobbers $S \xrightarrow{p} T$ in a partially ordered plan Π, then we can solve the induced conflict by
  - demotion: add a temporal constraint $C \prec S$ to Π, or
  - promotion: add $T \prec C$ to Π.
- **Example 5.24.** $Go(Home)$ clobbers $At(Supermarket)$:



Clobbering and Promotion/Demotion

- **Definition 5.25.** In a partially ordered plan, a step C clobbers a causal link $L := S \xrightarrow{p} T$, iff it destroys the condition p achieved by L.
- **Definition 5.26.** If C clobbers $S \xrightarrow{p} T$ in a partially ordered plan Π, then we can solve the induced conflict by
  - demotion: add a temporal constraint $C \prec S$ to Π, or
  - promotion: add $T \prec C$ to Π.
- **Example 5.27.** $Go(Home)$ clobbers $At(Supermarket)$:



POP algorithm sketch

- **Definition 5.28.** The POP algorithm for constructing complete partially ordered plans:

```
function POP (initial, goal, operators) : plan
    plan:= Make–Minimal–Plan(initial, goal)
    loop do
        if Solution?(goal,plan) then return plan
        S_need, c := Select–Subgoal(plan)
        Choose–Operator(plan, operators, S_need,c)
        Resolve–Threats(plan)
    end

function Select–Subgoal (plan, S_need, c)
    pick a plan step S_need from Steps(plan)
        with a precondition c that has not been achieved
    return S_need, c
```

POP algorithm contd.

- **Definition 5.29.** The missing parts for the POP algorithm.

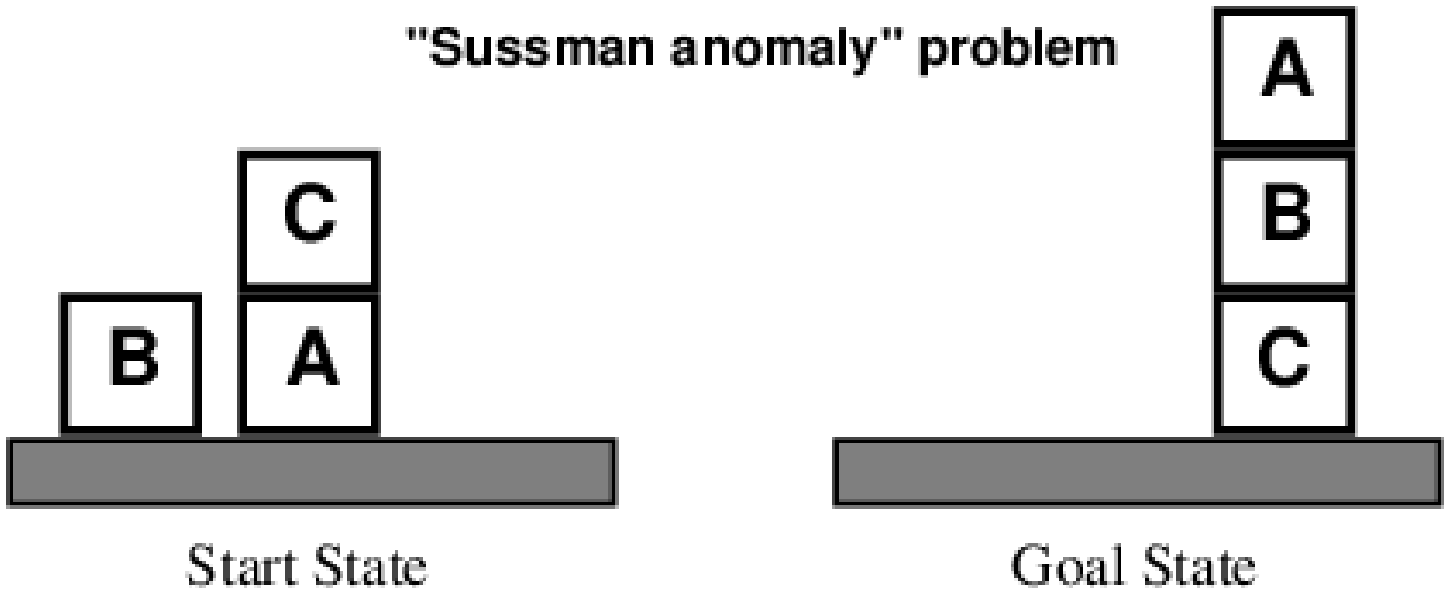
function Choose–Operator (plan, operators, S_{need}, c)
  choose a step S_{add} from operators or Steps(plan) that has c as an effect
  if there is no such step **then fail**
  add the causal–link $S_{add} \xrightarrow{c} S_{need}$ **to** Links(plan)
  add the temporal–constraint $S_{add} \prec S_{need}$ **to** Orderings(plan)
  if S_{add} is a newly added \step from operators **then**
    add S_{add} **to** Steps(plan)
    add $Start \prec S_{add} \prec Finish$ **to** Orderings(plan)

function Resolve–Threats (plan)
  for each S_{threat} that threatens a causal–link $S_i \xrightarrow{c} S_j$ **in** Links(plan) **do**
    choose either
      demotion: Add $S_{threat} \prec S_i$ **to** Orderings(plan)
      promotion: Add $S_j \prec S_{threat}$ **to** Orderings(plan)
    if not Consistent(plan) **then fail**

Properties of POP

- Nondeterministic algorithm: backtracks at choice points on failure:
  - choice of S_{add} to achieve S_{need},
  - choice of demotion or promotion for clobberer,
  - selection of S_{need} is irrevocable.
- **Observation 5.30.** *POP is sound, complete, and systematic i.e. no repetition*
- There are extensions for disjunction, universals, negation, conditionals.
- It can be made efficient with good heuristics derived from problem description.
- Particularly good for problems with many loosely related subgoals.

Example: Solving the Sussman Anomaly



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)

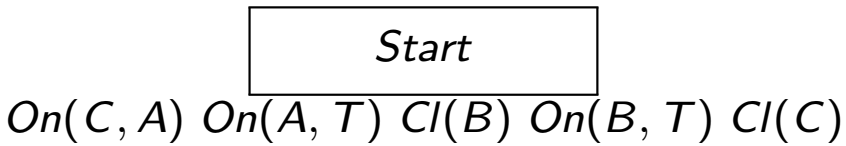
PutOnTable(x)

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

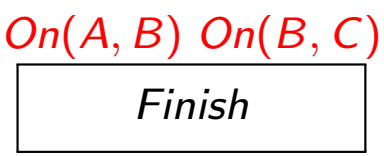
+ several inequality constraints

Example: Solving the Sussman Anomaly (contd.)

- **Example 5.31.** Solving the Sussman anomaly

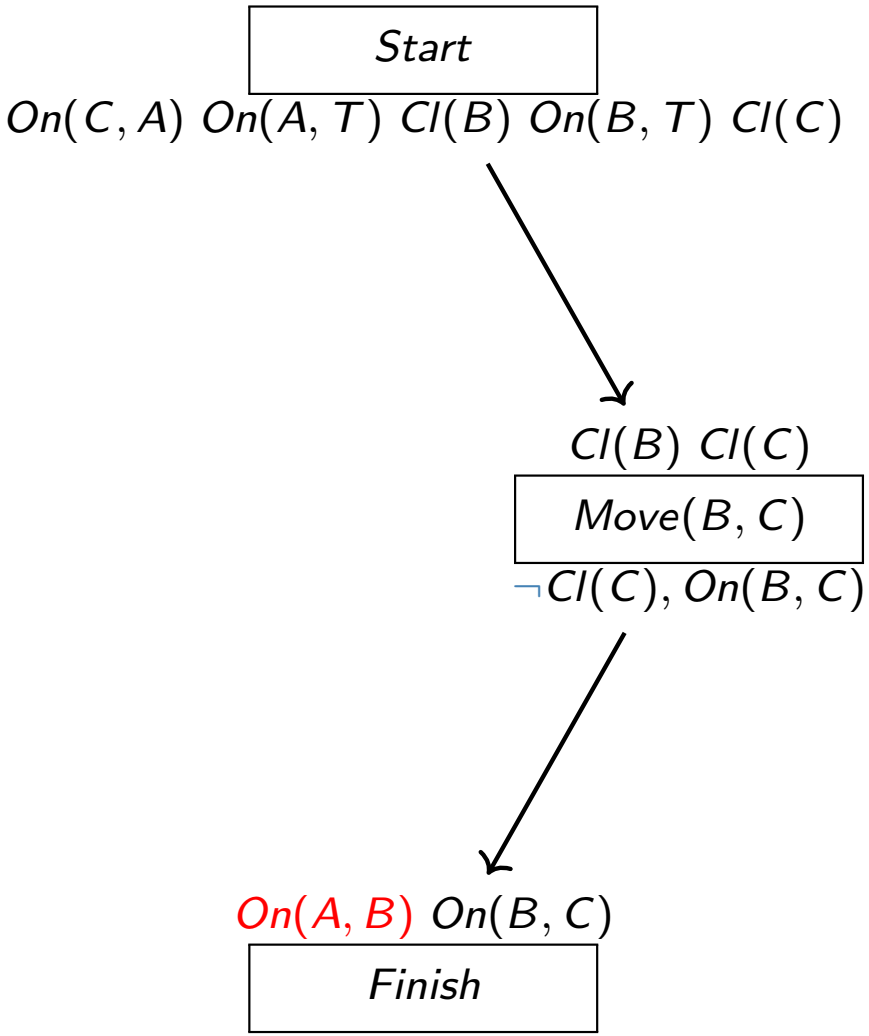


Initializing the partial order plan with with Start and Finish.



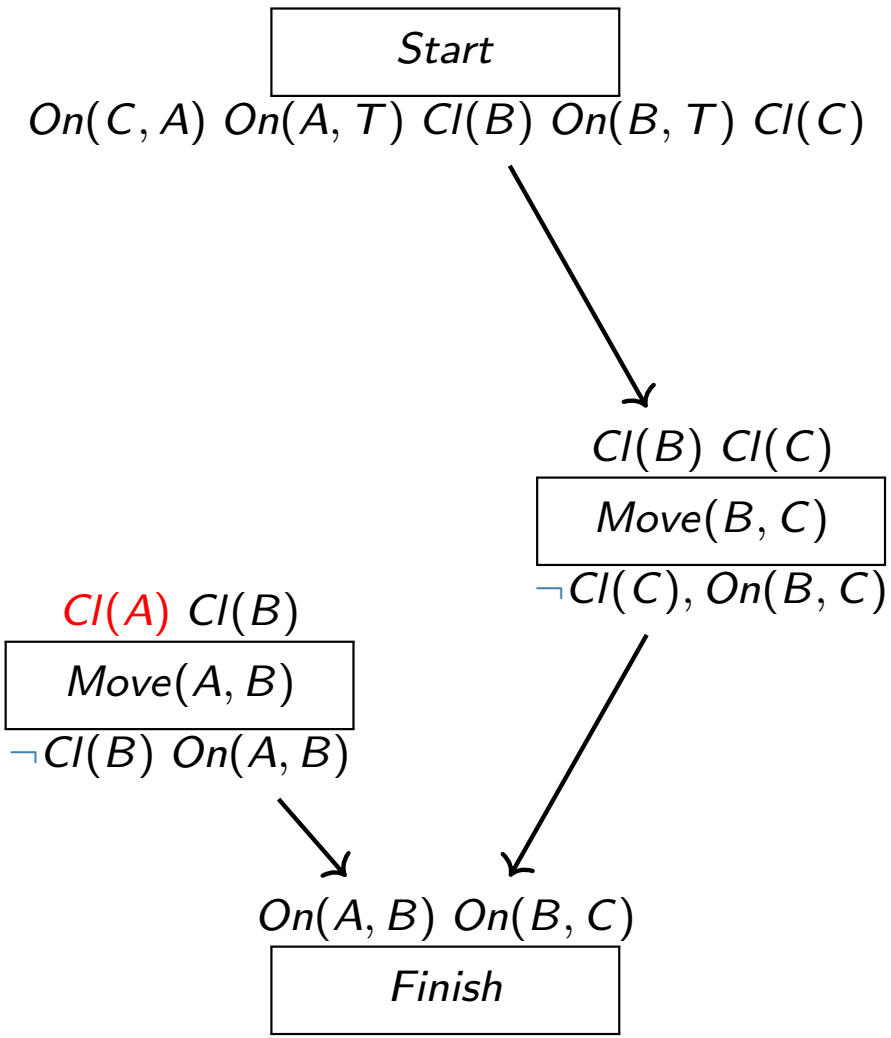
Example: Solving the Sussman Anomaly (contd.)

- **Example 5.32.** Solving the Sussman anomaly



Example: Solving the Sussman Anomaly (contd.)

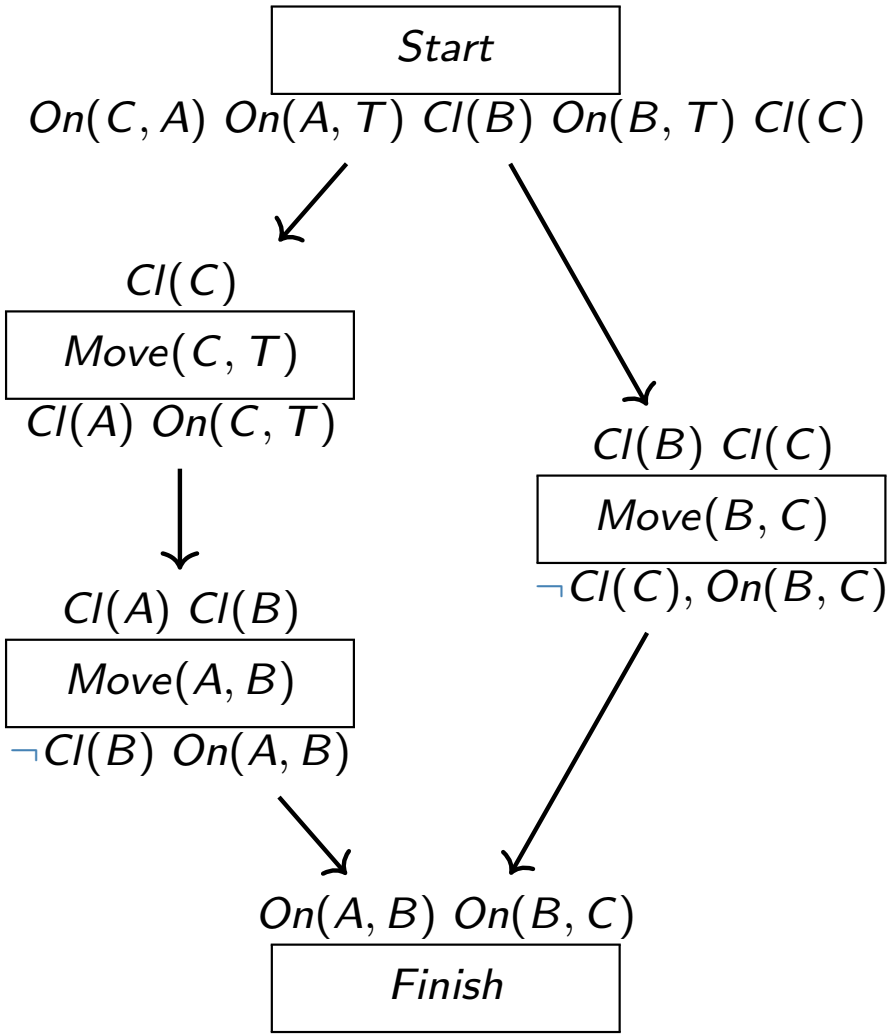
- **Example 5.33.** Solving the Sussman anomaly



Refining for the subgoal $ON(A, C)$.

Example: Solving the Sussman Anomaly (contd.)

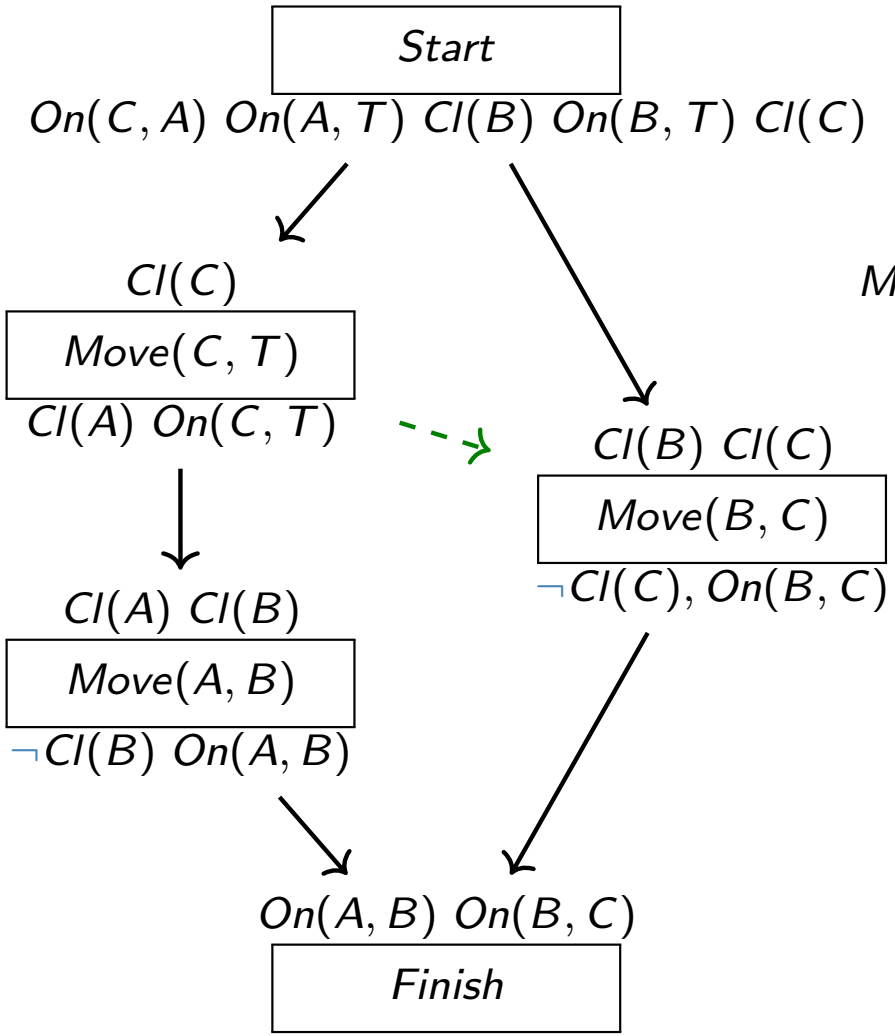
- **Example 5.34.** Solving the Sussman anomaly



Refining for the subgoal $Cl(A)$.

Example: Solving the Sussman Anomaly (contd.)

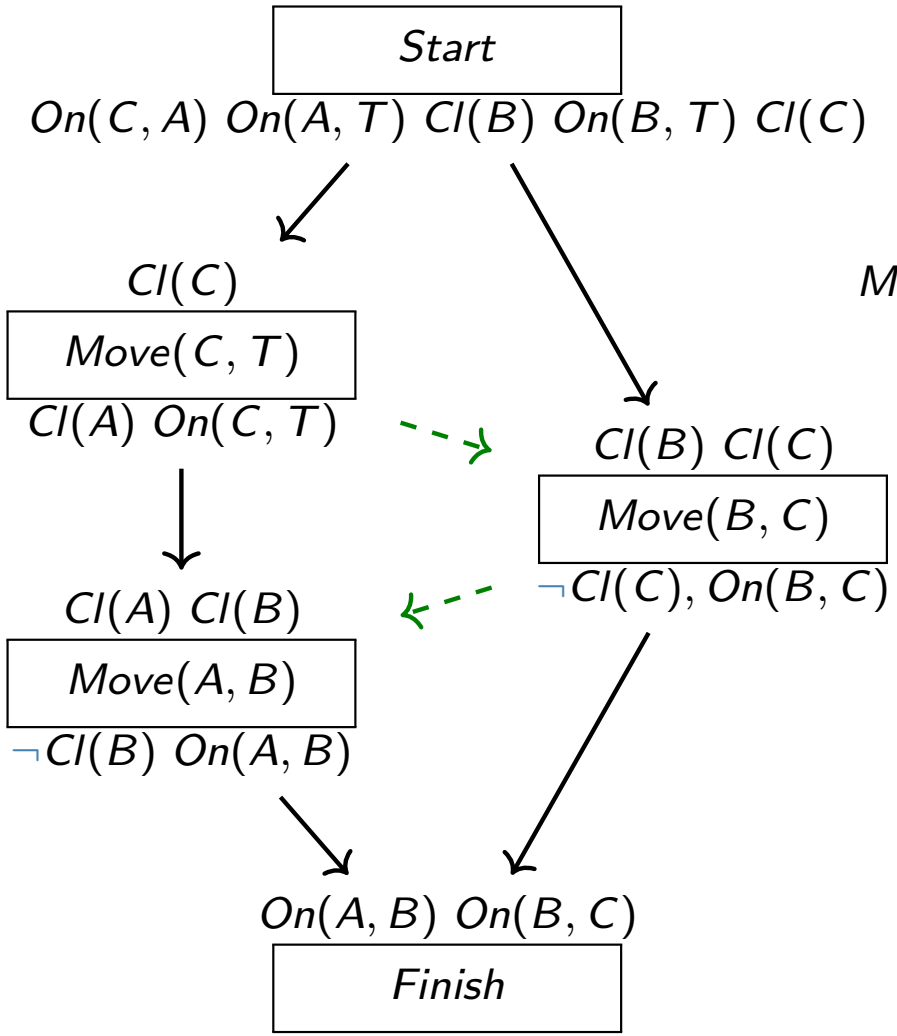
- **Example 5.35.** Solving the Sussman anomaly



$Move(A, B)$ clobbers $Cl(B) \rightsquigarrow$ demote.

Example: Solving the Sussman Anomaly (contd.)

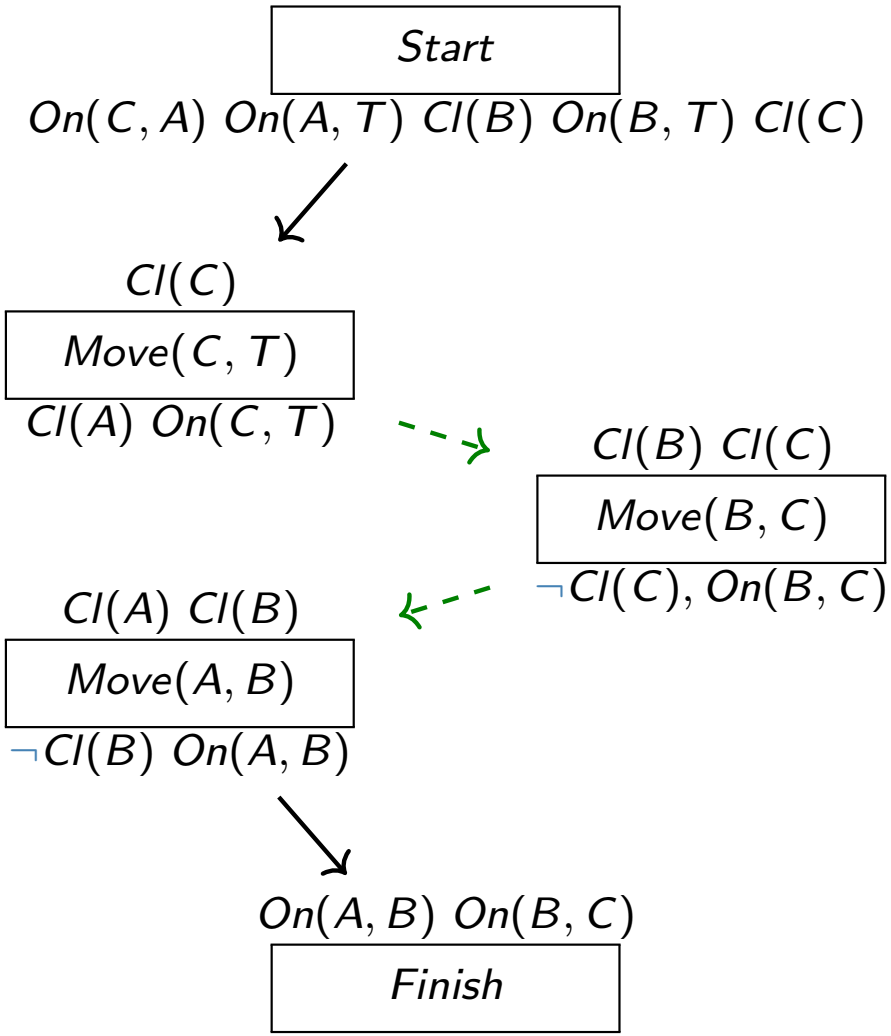
- **Example 5.36.** Solving the Sussman anomaly



$Move(B, C)$ clobbers $Cl(C) \rightsquigarrow$ demote.

Example: Solving the Sussman Anomaly (contd.)

- **Example 5.37.** Solving the Sussman anomaly



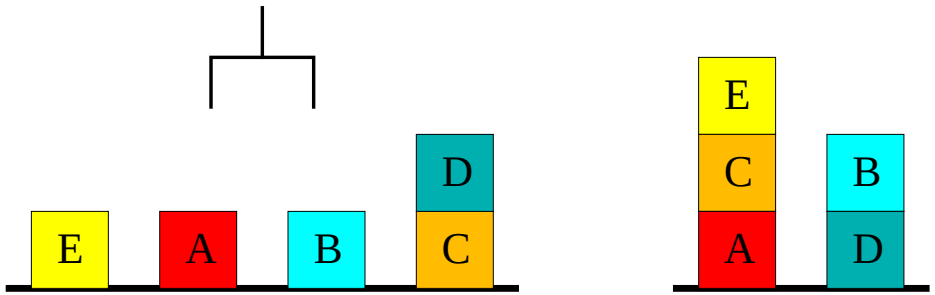
A totally ordered plan.

17.6 The PDDL Language

PDDL: Planning Domain Description Language

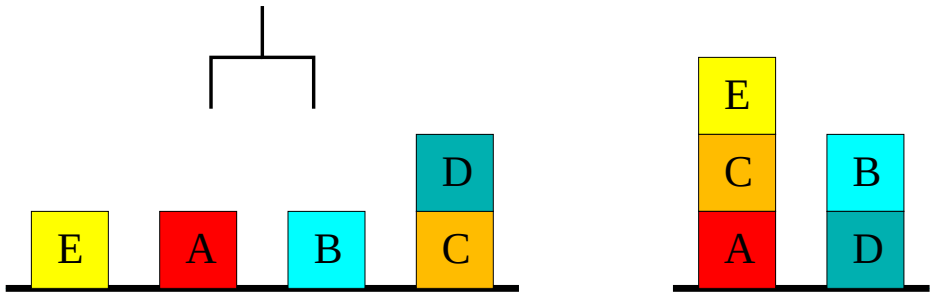
- **Definition 6.1.** The Planning Domain Description Language (PDDL) is a standardized representation language for planning benchmarks in various extensions of the STRIPS formalism.
- **Definition 6.2.** PDDL is not a propositional language
  - Representation is lifted, using object variables to be instantiated from a finite set of objects. (Similar to predicate logic)
  - Action schemas parameterized by objects.
  - Predicates to be instantiated with objects.
- **Definition 6.3.** A PDDL planning task comes in two pieces
  - The problem file gives the objects, the initial state, and the goal state.
  - The domain file gives the predicates and the actions.

The Blocksworld in PDDL: Domain File




```
(define (domain blocksworld)
  (:predicates (clear ?x) (holding ?x) (on ?x ?y)
               (on–table ?x) (arm–empty))
  (:action stack
   :parameters (?x ?y)
   :precondition (and (clear ?y) (holding ?x))
   :effect (and (arm–empty) (on ?x ?y)
                (not (clear ?y)) (not (holding ?x))))
  ...)
```

The Blocksworld in PDDL: Problem File




```
(define (problem bw-abcde)
  (:domain blocksworld)
  (:objects a b c d e)
  (:init (on-table a) (clear a)
    (on-table b) (clear b)
    (on-table e) (clear e)
    (on-table c) (on d c) (clear d)
    (arm-empty))
  (:goal (and (on e c) (on c a) (on b d))))
```

Miconic-ADL "Stop" Action Schema in PDDL

```
(:action stop
 :parameters (?f - floor)
 :precondition (and (lift-at ?f)
  (imply
   (exists
    (?p - conflict-A)
    (or (and (not (served ?p))
             (origin ?p ?f))
        (and (boarded ?p)
             (not (destin ?p ?f)))))
   (forall
    (?q - conflict-B)
    (and (or (destin ?q ?f)
             (not (boarded ?q)))
         (or (served ?q)
             (not (origin ?q ?f))))))
  (imply (exists
          (?p - conflict-B)
          (or (and (not (served ?p))
                   (origin ?p ?f))
              (and (boarded ?p)
                   (not (destin ?p ?f)))))
         (forall
          (?q - conflict-A)
          (and (or (destin ?q ?f)
                   (not (boarded ?q)))
               (or (served ?q)
                   (not (origin ?q ?f))))))
  (imply
   (exists
    (?p - never-alone)
    (or (and (origin ?p ?f)
             (not (served ?p)))
        (and (boarded ?p)
             (not (destin ?p ?f)))))
   (exists
    (?q - attendant)
    (or (and (boarded ?q)
             (not (destin ?q ?f)))
        (and (not (served ?q))
             (origin ?q ?f)))))
  (forall
   (?p - going-nonstop)
   (imply (boarded ?p) (destin ?p ?f)))
  (or (forall
       (?p - vip) (served ?p))
      (exists
       (?p - vip)
       (or (origin ?p ?f) (destin ?p ?f))))
  (forall
   (?p - passenger)
   (imply
    (no-access ?p ?f) (not (boarded ?p)))))
)
```

Planning Domain Description Language

- **Question:** What is PDDL good for?
  (A) Nothing.
  (B) Free beer.
  (C) Those AI planning guys.
  (D) Being lazy at work.

Planning Domain Description Language

- **Question:** What is PDDL good for?
  (A) Nothing.
  (B) Free beer.
  (C) Those AI planning guys.
  (D) Being lazy at work.
- **Answer:**
  (A) Nah, it's definitely good for *something* (see remaining answers)

Planning Domain Description Language

- **Question:** What is PDDL good for?
  (A) Nothing.
  (B) Free beer.
  (C) Those AI planning guys.
  (D) Being lazy at work.
- **Answer:**
  (A) Nah, it's definitely good for *something* (see remaining answers)
  (B) Generally, no. Sometimes, yes: PDDL is needed for the IPC, and if you win the IPC you get price money (= free beer).

Planning Domain Description Language

- **Question:** What is PDDL good for?
  (A) Nothing.
  (B) Free beer.
  (C) Those AI planning guys.
  (D) Being lazy at work.
- **Answer:**
  (A) Nah, it's definitely good for *something* (see remaining answers)
  (B) Generally, no. Sometimes, yes: PDDL is needed for the IPC, and if you win the IPC you get price money (= free beer).
  (C) Yep. (Initially, every system had its own language, so running experiments felt a lot like "Lost in Translation".)

Planning Domain Description Language

- **Question:** What is PDDL good for?
  (A) Nothing.
  (B) Free beer.
  (C) Those AI planning guys.
  (D) Being lazy at work.
- **Answer:**
  (A) Nah, it's definitely good for *something* (see remaining answers)
  (B) Generally, no. Sometimes, yes: PDDL is needed for the IPC, and if you win the IPC you get price money (= free beer).
  (C) Yep. (Initially, every system had its own language, so running experiments felt a lot like "Lost in Translation".)
  (D) Yep. You can be a busy bee, programming a solver yourself. Or you can be lazy and just write the PDDL. (I think I said that before . . .)

17.7 Conclusion

Summary

- General problem solving attempts to develop solvers that perform well across a large class of problems.
- Planning, as considered here, is a form of general problem solving dedicated to the class of classical search problems. (Actually, we also address inaccessible, stochastic, dynamic, continuous, and multi-agent settings.)
- Heuristic search planning has dominated the International Planning Competition (IPC). We focus on it here.
- STRIPS is the simplest possible, while reasonably expressive, language for our purposes. It uses Boolean variables (facts), and defines actions in terms of precondition, add list, and delete list.
- PDDL is the de-facto standard language for describing planning problems.
- Plan existence (bounded or not) is PSPACE-complete to decide for STRIPS. If we bound plans polynomially, we get down to NP-completeness.

Chapter 18
Planning II: Algorithms

18.1 Introduction

Reminder: Our Agenda for This Topic

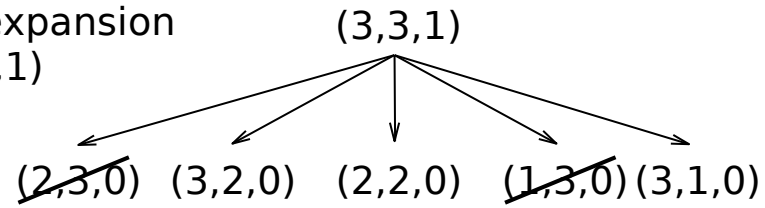
- **??**: Background, planning languages, complexity.
  - Sets up the framework. computational complexity is essential to distinguish different algorithmic problems, and for the design of heuristic functions.
- **This Chapter:** How to automatically generate a heuristic function, given planning language input?
  - Focussing on heuristic search as the solution method, this is the main question that needs to be answered.

Reminder: Search

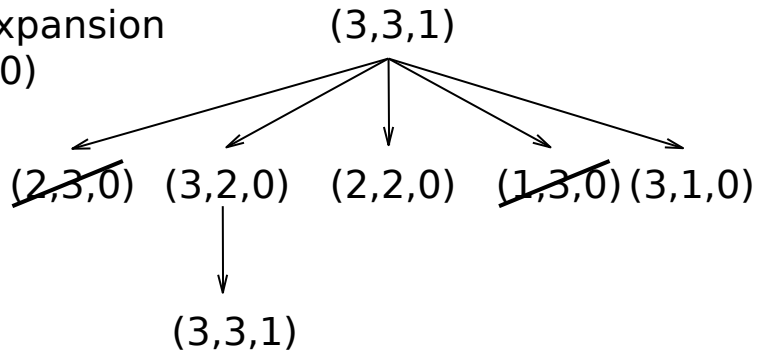
- Starting at initial state, produce all successor states step by step:

(a) initial state (3,3,1)

(b) after expansion of (3,3,1)

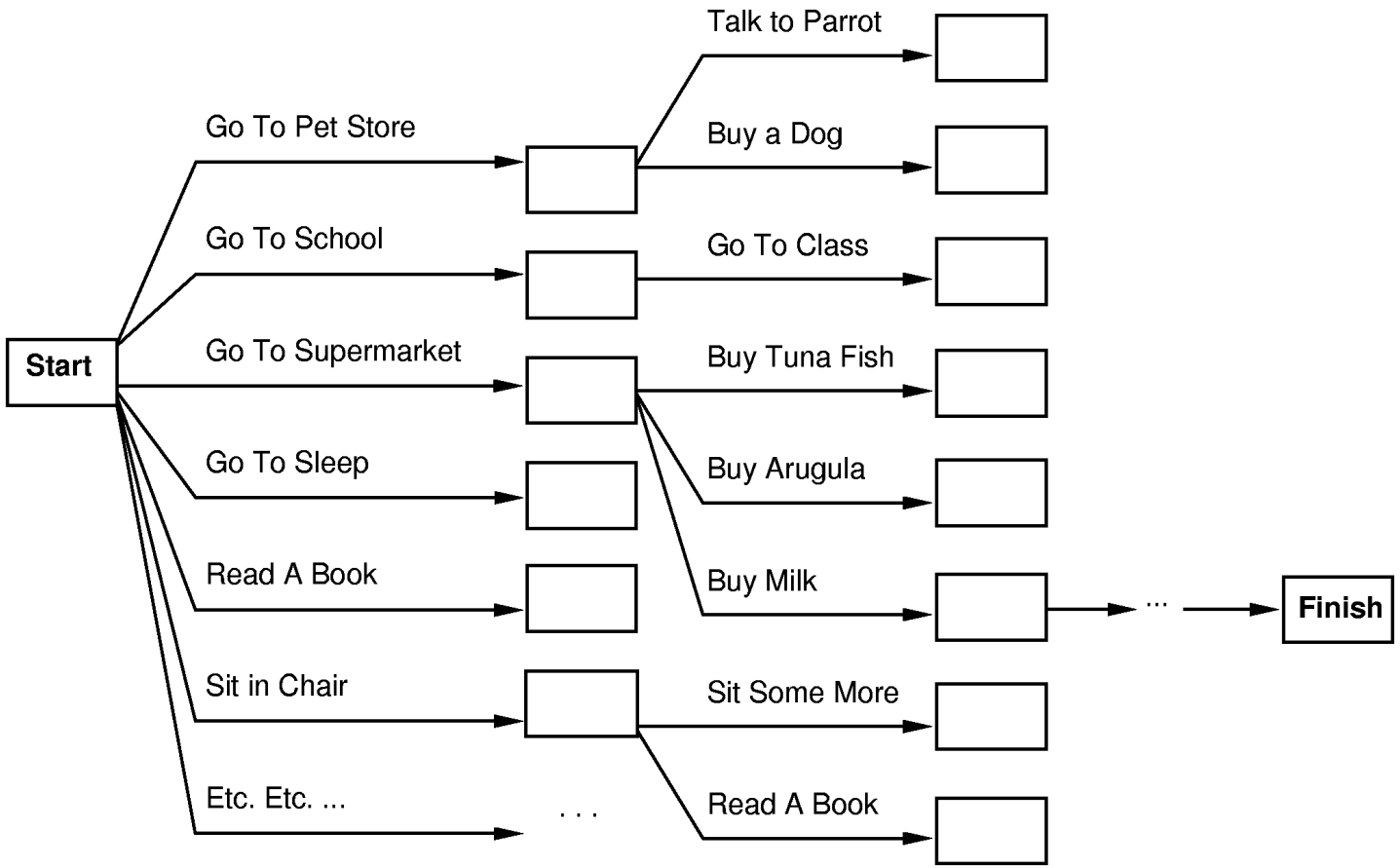


(c) after expansion of (3,2,0)



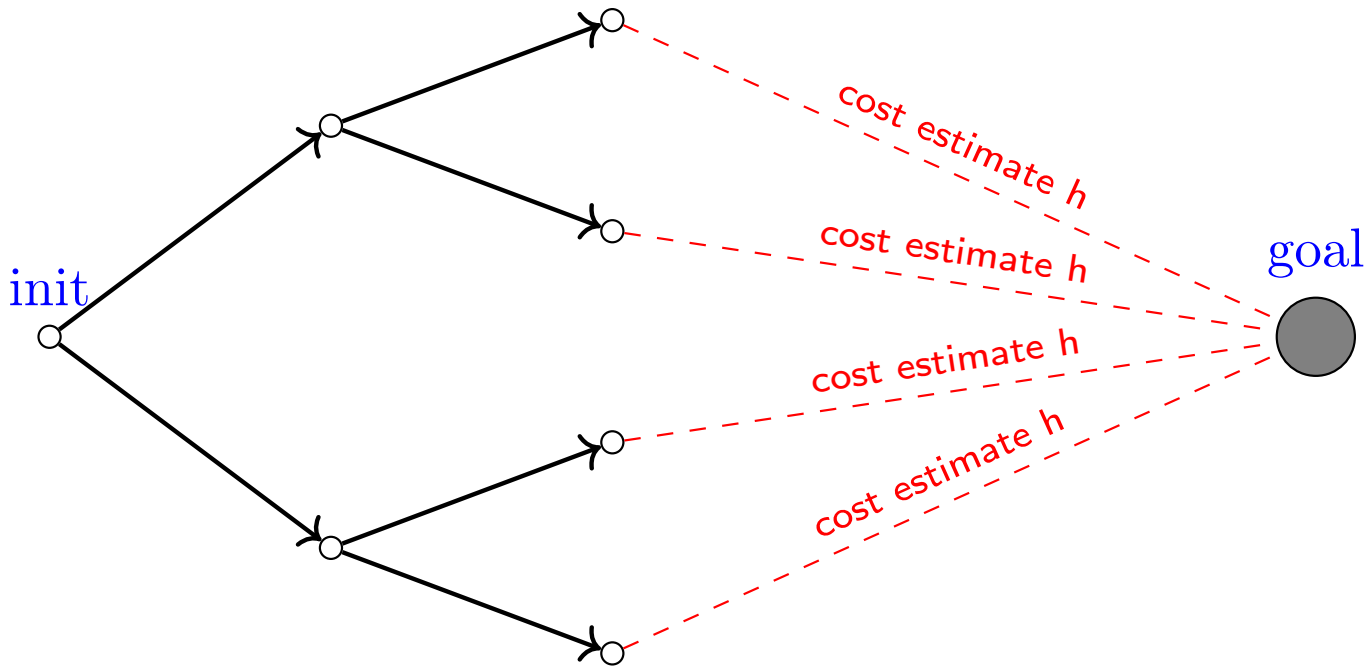
In planning, this is referred to as forward search, or forward state-space search.

Search in the State Space?



- Use heuristic function to guide the search towards the goal!

Reminder: Informed Search



- Heuristic function h estimates the cost of an optimal path from a state s to the goal state; search prefers to expand states s with small $h(s)$.
- Live Demo vs. Breadth-First Search:

  `http://qiao.github.io/PathFinding.js/visual/`

Reminder: Heuristic Functions

- **Definition 1.1.** Let Π be a STRIPS task with states S. A heuristic function, short heuristic, for Π is a function $h\colon S \to \mathbb{N} \cup \{\infty\}$ so that $h(s) = 0$ whenever s is a goal state.
- Exactly like our definition from **??**. Except, because we assume unit costs here, we use $\mathbb{N}$ instead of $\mathbb{R}^+$.
- **Definition 1.2.** Let Π be a STRIPS task with states S. The perfect heuristic h^* assigns every $s \in S$ the length of a shortest path from s to a goal state, or ∞ if no such path exists. A heuristic h for Π is admissible if, for all $s \in S$, we have $h(s) \leq h^*(s)$.
- Exactly like our definition from **??**, except for path *length* instead of path *cost* (cf. above).
- In all cases, we attempt to approximate $h^*(s)$, the length of an optimal plan for s. Some algorithms guarantee to lower bound $h^*(s)$.

Our (Refined) Agenda for This Chapter

- **How to Relax:** How to relax a problem?
  - Basic principle for generating heuristic functions.
- **The Delete Relaxation:** How to relax a planning problem?
  - The delete relaxation is the most successful method for the *automatic* generation of heuristic functions. It is a key ingredient to almost all IPC winners of the last decade. It relaxes STRIPS tasks by ignoring the delete lists.
- **The h^+ Heuristic:** What is the resulting heuristic function?
  - h^+ is the "ideal" delete relaxation heuristic.
- **Approximating h^+:** How to actually compute a heuristic?
  - Turns out that, in practice, we must approximate h^+.

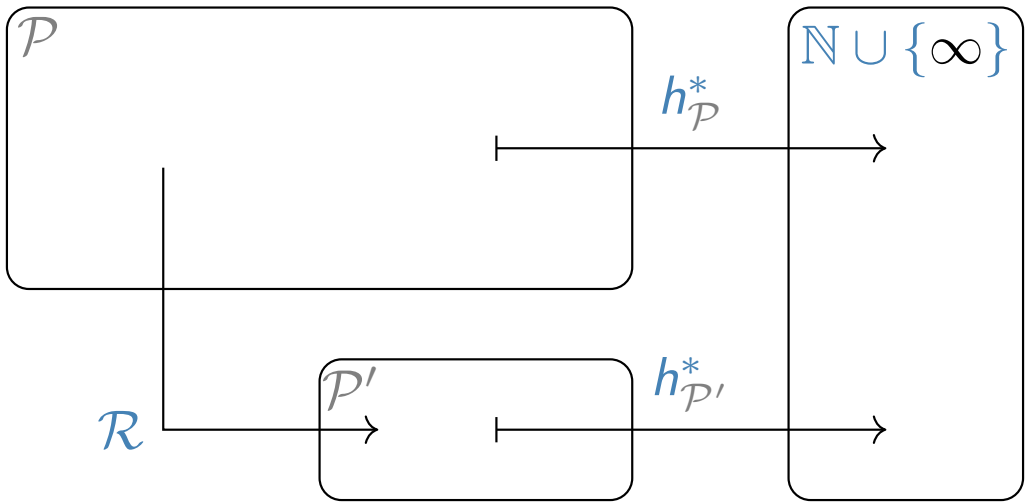
18.2 How to Relax in Planning

How to Relax

- **Recall:** We introduced the concept of a relaxed search problem (allow cheating) to derive heuristics from them.
- **Observation:** This can be generalized to arbitrary problem solving.

How to Relax

- **Recall:** We introduced the concept of a relaxed search problem (allow cheating) to derive heuristics from them.
- **Observation:** This can be generalized to arbitrary problem solving.
- **Definition 2.3 (The General Case).**



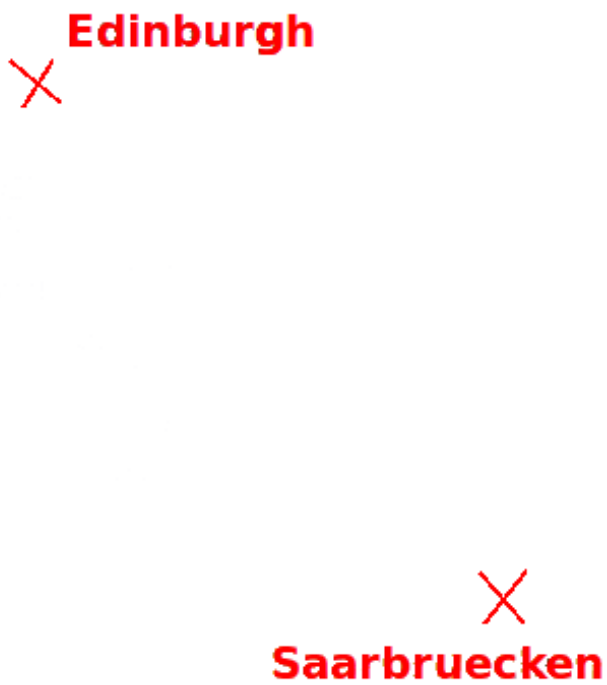
1. You have a class $\mathcal{P}$ of problems, whose perfect heuristic $h^*_{\mathcal{P}}$ you wish to estimate.
2. You define a class $\mathcal{P}'$ of *simpler problems*, whose perfect heuristic $h^*_{\mathcal{P}'}$ can be used to estimate $h^*_{\mathcal{P}}$.
3. You define a transformation – the relaxation mapping $\mathcal{R}$ – that maps instances $\Pi \in \mathcal{P}$ into instances $\Pi' \in \mathcal{P}'$.
4. Given $\Pi \in \mathcal{P}$, you let $\Pi' := \mathcal{R}(\Pi)$, and estimate $h^*{}_{\mathcal{P}}(\Pi)$ by $h^*{}_{\mathcal{P}'}(\Pi')$.

- **Definition 2.4.** For planning tasks, we speak of relaxed planning.

Reminder: Heuristic Functions from Relaxed Problems

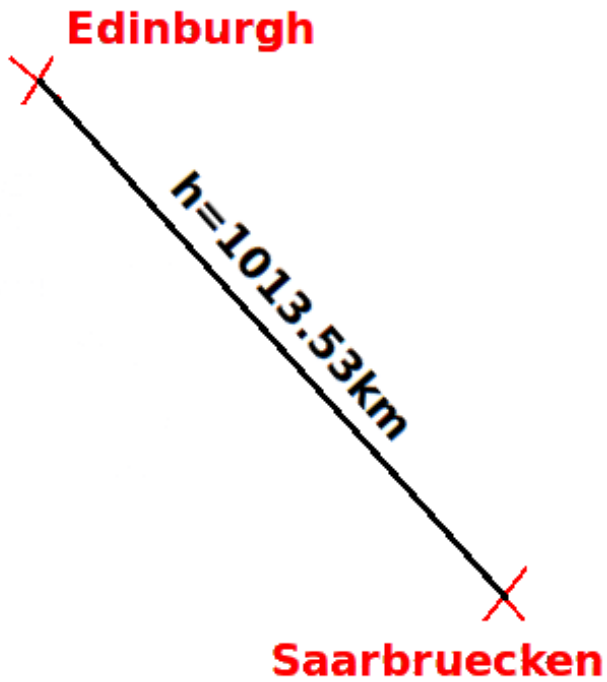
- Problem Π: Find a route from Saarbrücken to Edinburgh.

Reminder: Heuristic Functions from Relaxed Problems



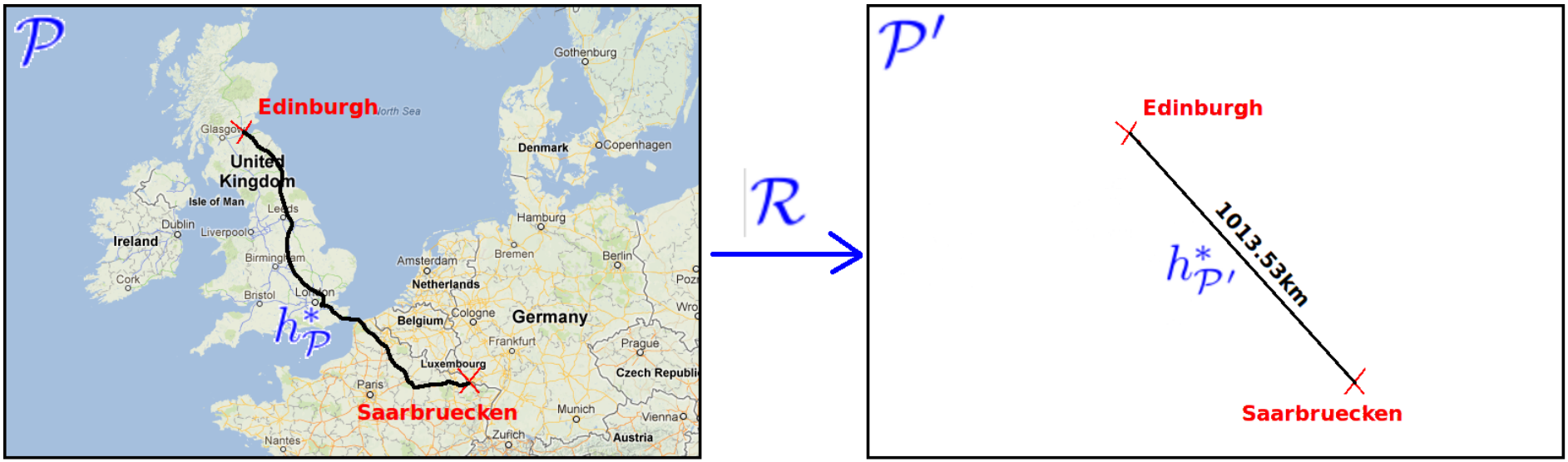
- Relaxed Problem Π': Throw away the map.

Reminder: Heuristic Functions from Relaxed Problems



- Heuristic function h: Straight line distance.

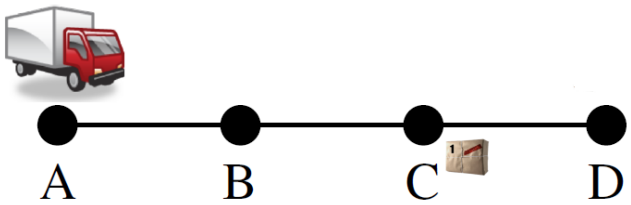
Relaxation in Route-Finding



- **Problem class** $\mathcal{P}$: Route finding.
- **Perfect heuristic** $h^*_{\mathcal{P}}$ **for** $\mathcal{P}$: Length of a shortest route.
- **Simpler problem class** $\mathcal{P}'$: Route finding on an empty map.
- **Perfect heuristic** $h^*_{\mathcal{P}'}$ **for** $\mathcal{P}'$: Straight-line distance.
- **Transformation** $\mathcal{R}$: Throw away the map.

How to Relax in Planning? (A Reminder!)

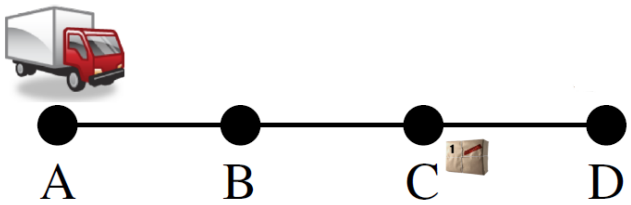
- **Example 2.5 (Logistics).**



- facts P: $\{\text{truck}(x) \mid x \in \{A, B, C, D\}\} \cup \{\text{pack}(x) \mid x \in \{A, B, C, D, T\}\}$.
- initial state I: $\{\text{truck}(A), \text{pack}(C)\}$.
- goal state G: $\{\text{truck}(A), \text{pack}(D)\}$.
- actions A: (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
  - $\text{drive}(x, y)$, where x and y have a road: "$\text{truck}(x) \Rightarrow \text{truck}(y), \neg\text{truck}(x)$".
  - $\text{load}(x)$: "$\text{truck}(x), \text{pack}(x) \Rightarrow \text{pack}(T), \neg\text{pack}(x)$".
  - $\text{unload}(x)$: "$\text{truck}(x), \text{pack}(T) \Rightarrow \text{pack}(x), \neg\text{pack}(T)$".

How to Relax in Planning? (A Reminder!)

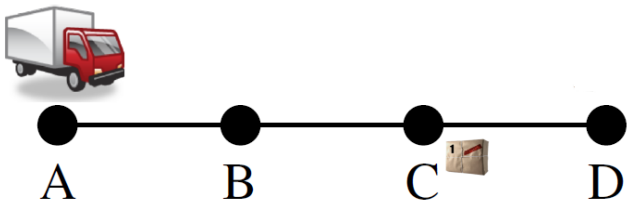
- **Example 2.7 (Logistics).**



  - facts P: $\{\text{truck}(x) \mid x \in \{A, B, C, D\}\} \cup \{\text{pack}(x) \mid x \in \{A, B, C, D, T\}\}$.
  - initial state I: $\{\text{truck}(A), \text{pack}(C)\}$.
  - goal state G: $\{\text{truck}(A), \text{pack}(D)\}$.
  - actions A: (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
    - $\text{drive}(x, y)$, where x and y have a road: "$\text{truck}(x) \Rightarrow \text{truck}(y), \neg\text{truck}(x)$".
    - $\text{load}(x)$: "$\text{truck}(x), \text{pack}(x) \Rightarrow \text{pack}(T), \neg\text{pack}(x)$".
    - $\text{unload}(x)$: "$\text{truck}(x), \text{pack}(T) \Rightarrow \text{pack}(x), \neg\text{pack}(T)$".
- **Example 2.8 ("Only-Adds" Relaxation).** Drop the preconditions and deletes.
  - "$\text{drive}(x, y)$: $\Rightarrow \text{truck}(y)$";
  - "$\text{load}(x)$: $\Rightarrow \text{pack}(T)$";
  - "$\text{unload}(x)$: $\Rightarrow \text{pack}(x)$".
- Heuristics value for I is?

How to Relax in Planning? (A Reminder!)

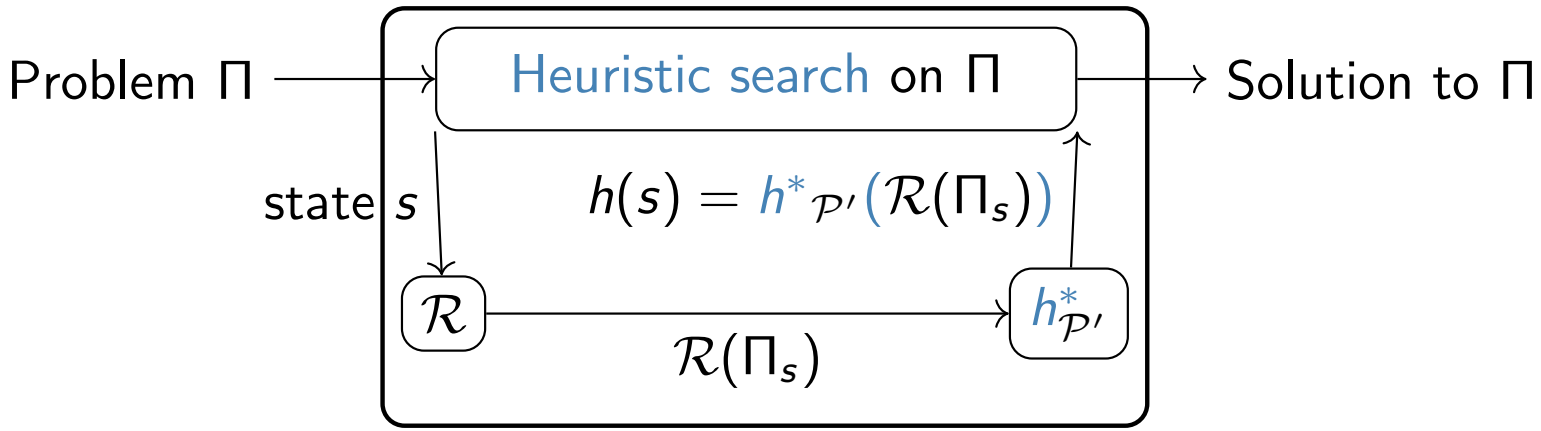
- **Example 2.9 (Logistics).**



  - facts P: $\{\text{truck}(x) \mid x \in \{A, B, C, D\}\} \cup \{\text{pack}(x) \mid x \in \{A, B, C, D, T\}\}$.
  - initial state I: $\{\text{truck}(A), \text{pack}(C)\}$.
  - goal state G: $\{\text{truck}(A), \text{pack}(D)\}$.
  - actions A: (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
    - $\text{drive}(x, y)$, where x and y have a road: "$\text{truck}(x) \Rightarrow \text{truck}(y), \neg\text{truck}(x)$".
    - $\text{load}(x)$: "$\text{truck}(x), \text{pack}(x) \Rightarrow \text{pack}(T), \neg\text{pack}(x)$".
    - $\text{unload}(x)$: "$\text{truck}(x), \text{pack}(T) \Rightarrow \text{pack}(x), \neg\text{pack}(T)$".
- **Example 2.10 ("Only-Adds" Relaxation).** Drop the preconditions and deletes.
  - "$\text{drive}(x, y)$: $\Rightarrow \text{truck}(y)$";
  - "$\text{load}(x)$: $\Rightarrow \text{pack}(T)$";
  - "$\text{unload}(x)$: $\Rightarrow \text{pack}(x)$".
- Heuristics value for I is?
- $h^{\mathcal{R}}(I) = 1$: A plan for the relaxed task is $\langle \text{unload}(D) \rangle$.

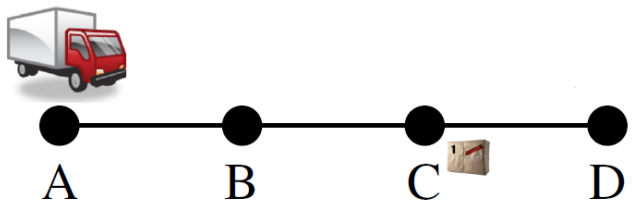
How to Relax During Search: Overview

- **Attention:** Search uses the real (un-relaxed) Π. The relaxation is applied (e.g., in Only-Adds, the simplified actions are used) **only within the call to $h(s)$!!!**



- Here, Π_s is Π with initial state replaced by s, i.e., $\Pi := \langle P, A, I, G \rangle$ changed to $\Pi^s := \langle P, A, \{s\}, G \rangle$: The task of finding a plan for search state s.
- A common student error is to instead apply the relaxation once to the whole problem, then doing the whole search "within the relaxation".
- The next slide illustrates the correct search process in detail.

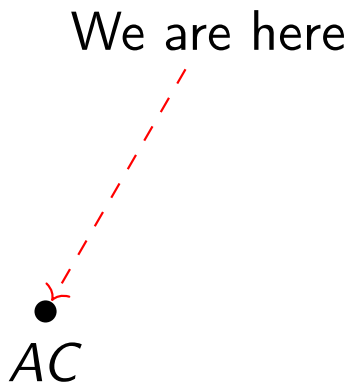
How to Relax During Search: Only-Adds



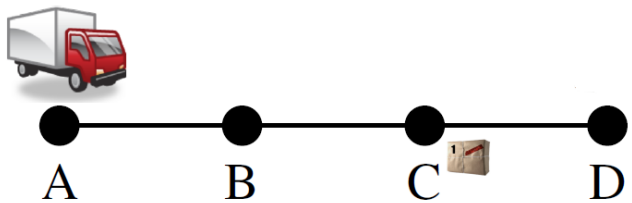
Real problem:

- Initial state I: AC; goal G: AD.
- Actions A: pre, add, del.
- $drXY$, loX, ulX.

Greedy best-first search: (tie-breaking: alphabetic)



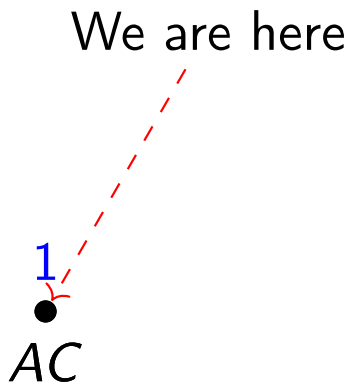
How to Relax During Search: Only-Adds



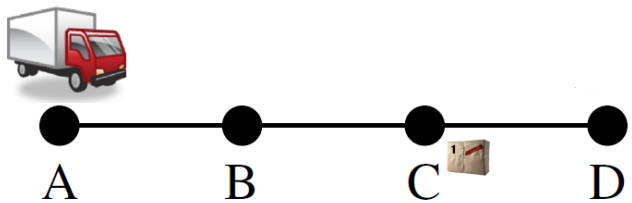
Relaxed problem:

- State s: AC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



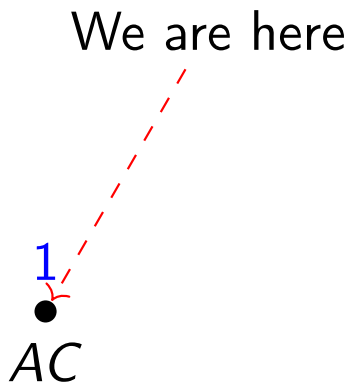
How to Relax During Search: Only-Adds



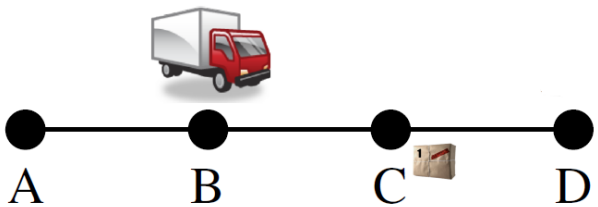
Relaxed problem:

- State s: AC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 1$: $\langle uID \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



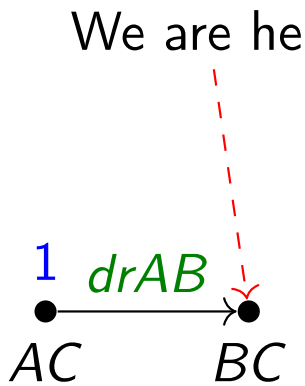
How to Relax During Search: Only-Adds



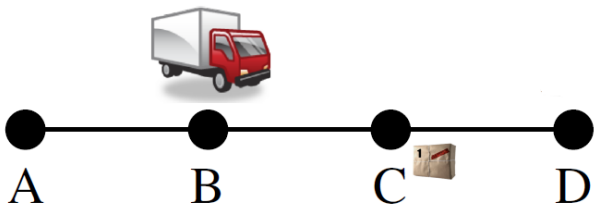
Real problem:

- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- $AC \xrightarrow{drAB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)



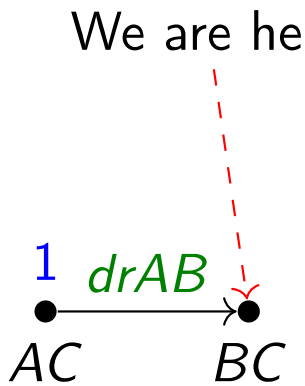
How to Relax During Search: Only-Adds



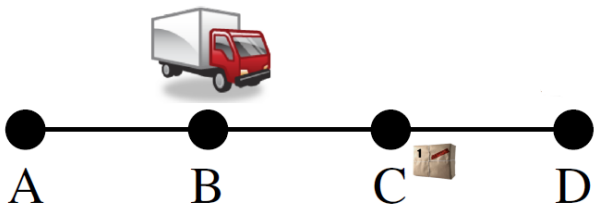
Relaxed problem:

- State s: BC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



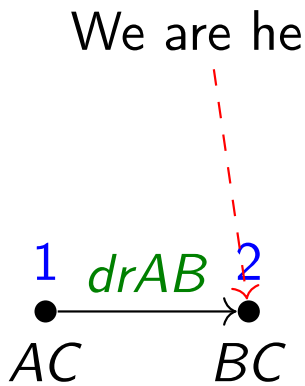
How to Relax During Search: Only-Adds



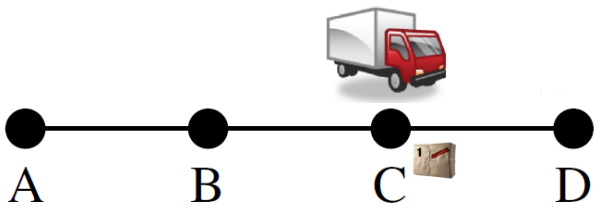
Relaxed problem:

- State s: BC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: $\langle drBA, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



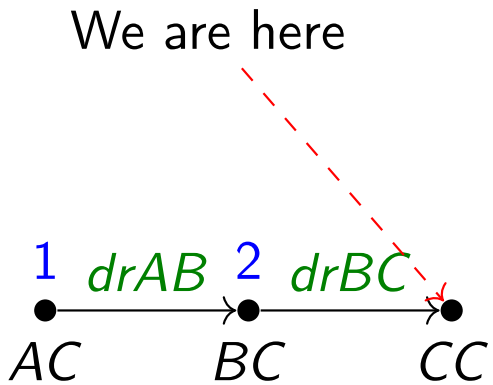
How to Relax During Search: Only-Adds



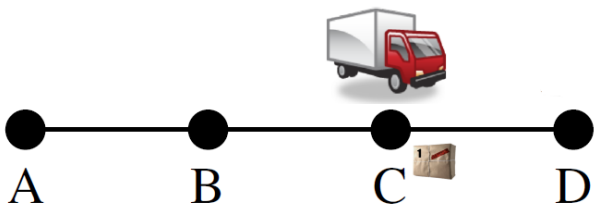
Real problem:

- State s: CC; goal G: AD.
- Actions A: pre, add, del.
- $BC \xrightarrow{drBC} CC$.

Greedy best-first search: (tie-breaking: alphabetic)



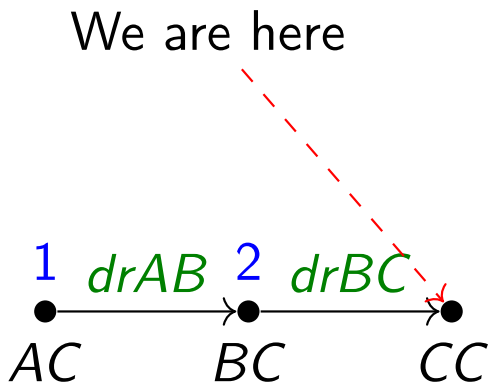
How to Relax During Search: Only-Adds



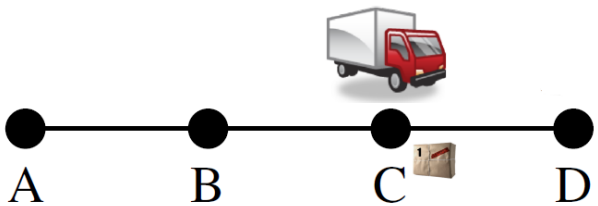
Relaxed problem:

- State s: CC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



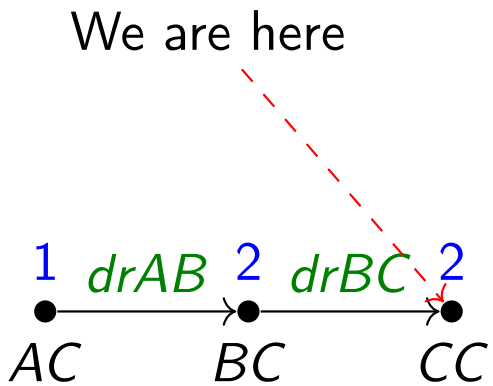
How to Relax During Search: Only-Adds



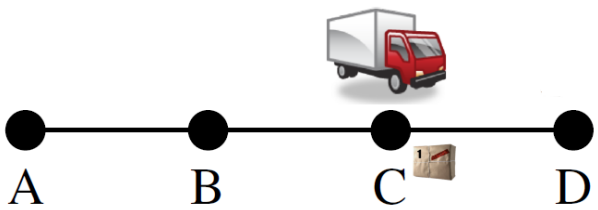
Relaxed problem:

- State s: CC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: $\langle drBA, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



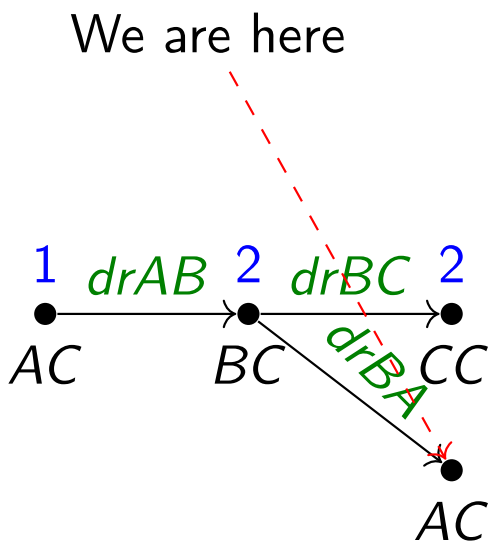
How to Relax During Search: Only-Adds



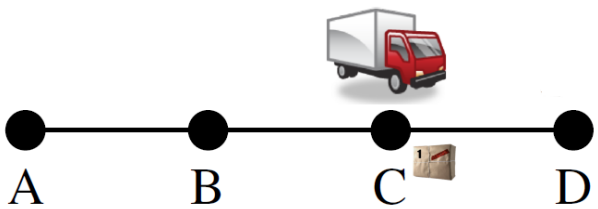
Real problem:

- State s: AC; goal G: AD.
- Actions A: pre, add, del.
- $BC \xrightarrow{drBA} AC$.

Greedy best-first search: (tie-breaking: alphabetic)



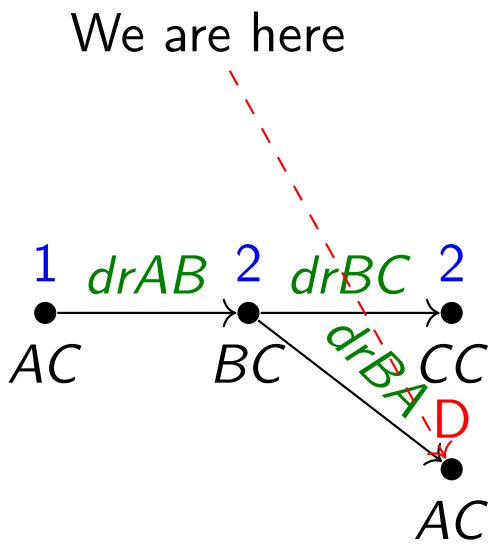
How to Relax During Search: Only-Adds



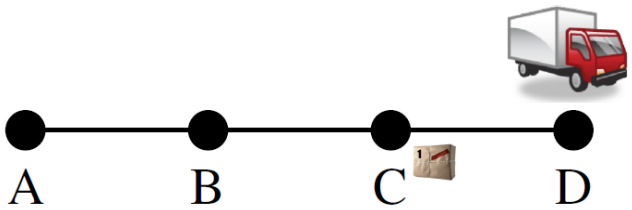
Real problem:

- State s: AC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)



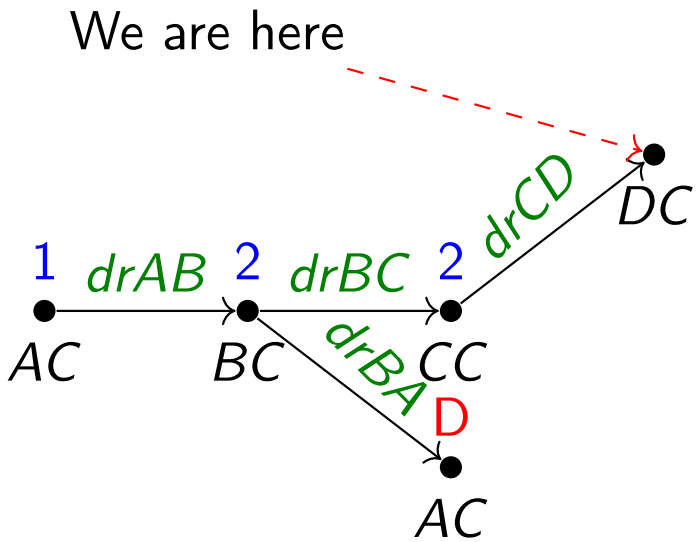
How to Relax During Search: Only-Adds



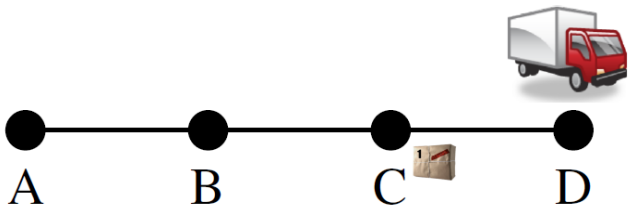
Real problem:

- State s: DC; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{drCD} DC$.

Greedy best-first search: (tie-breaking: alphabetic)



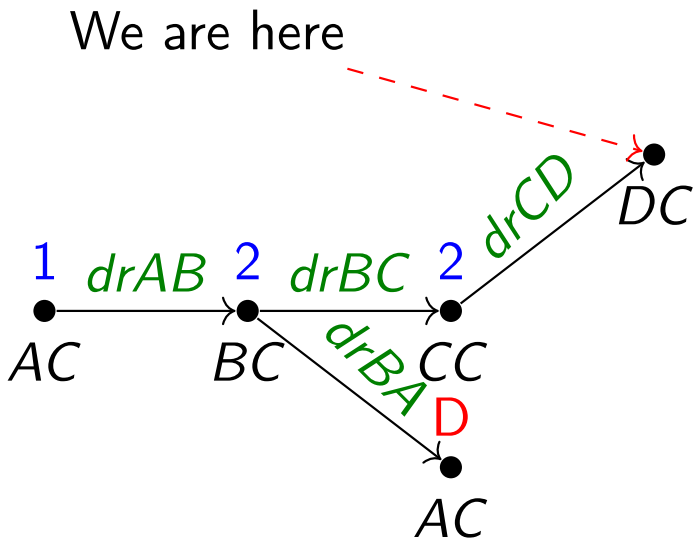
How to Relax During Search: Only-Adds



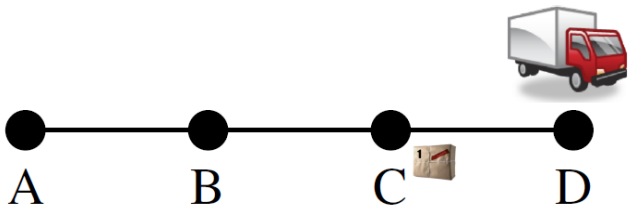
Relaxed problem:

- State s: DC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



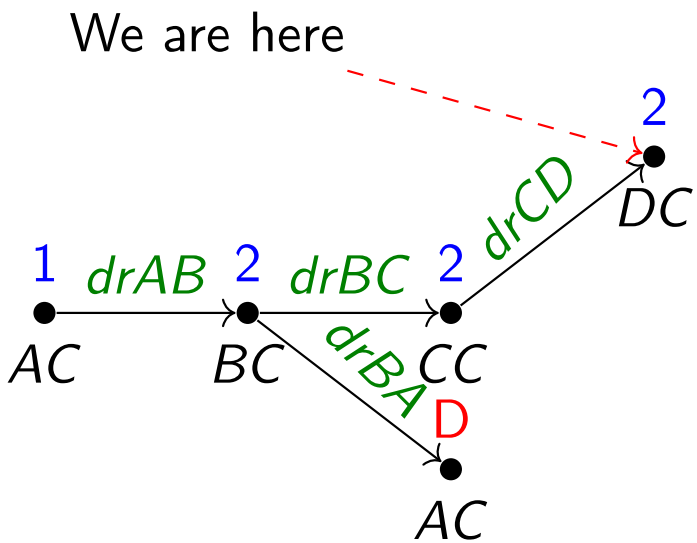
How to Relax During Search: Only-Adds



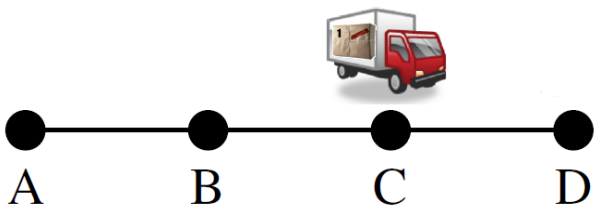
Relaxed problem:

- State s: DC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: $\langle drBA, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

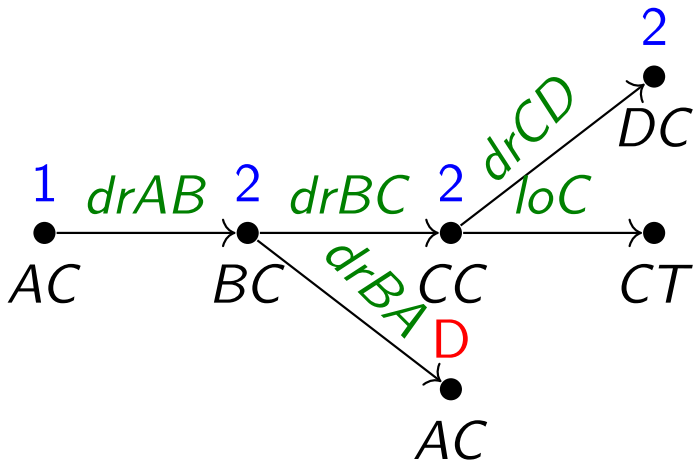


Real problem:

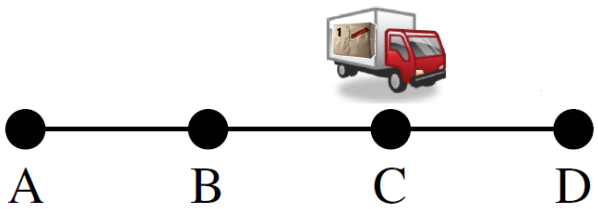
- State s: CT; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{loC} CT$.

Greedy best-first search: (tie-breaking: alphabetic)

We are here



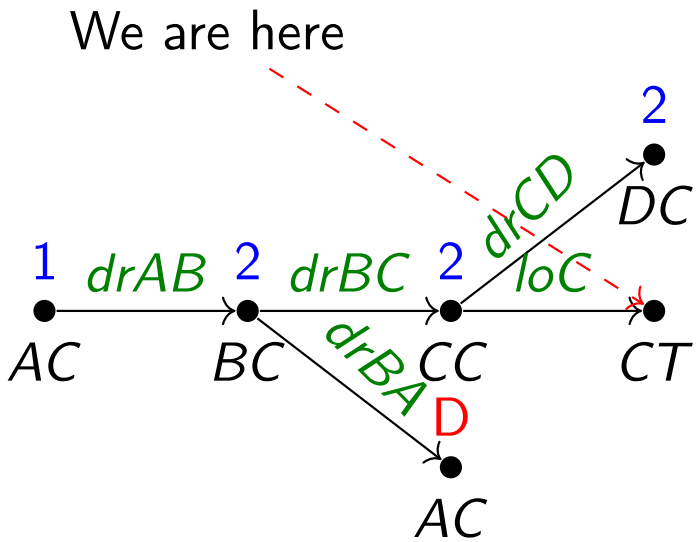
How to Relax During Search: Only-Adds



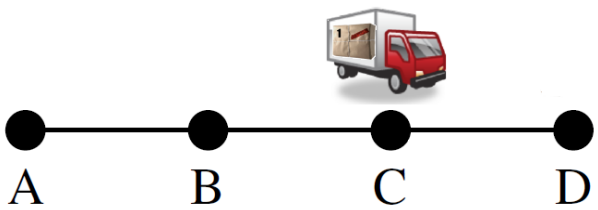
Relaxed problem:

- State s: CT; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



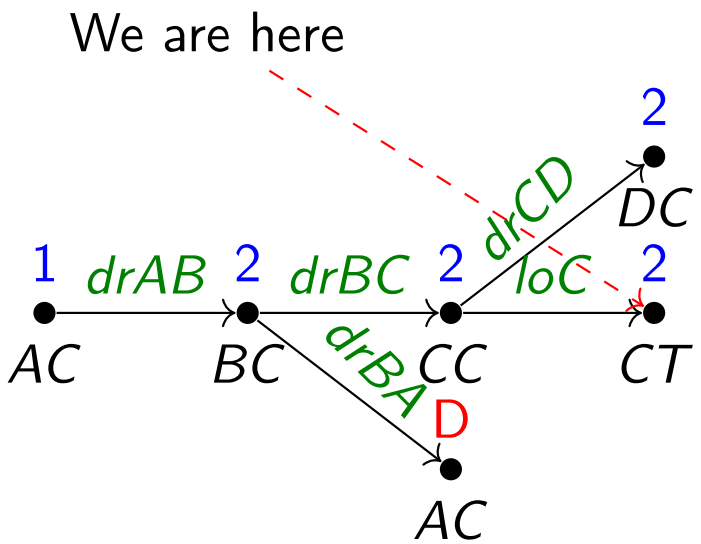
How to Relax During Search: Only-Adds



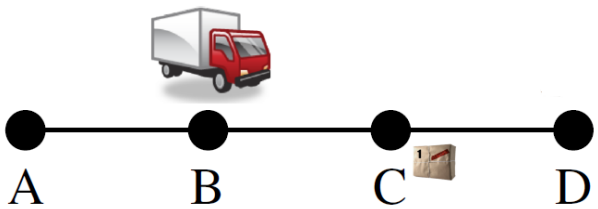
Relaxed problem:

- State s: CT; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: $\langle drBA, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



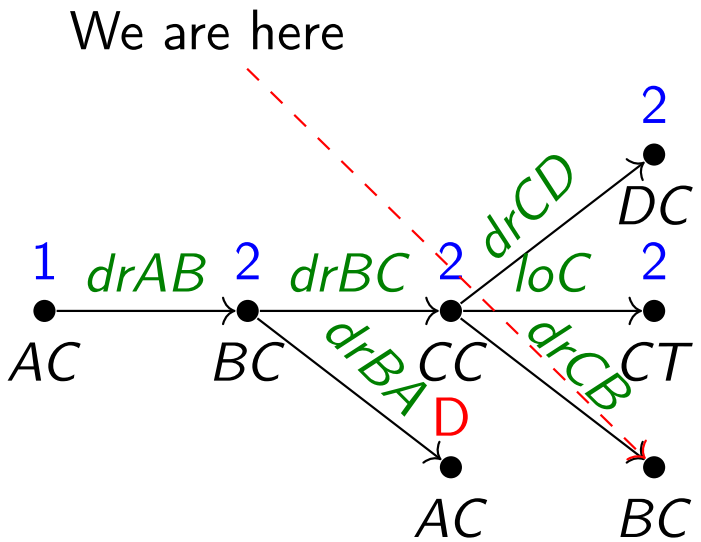
How to Relax During Search: Only-Adds



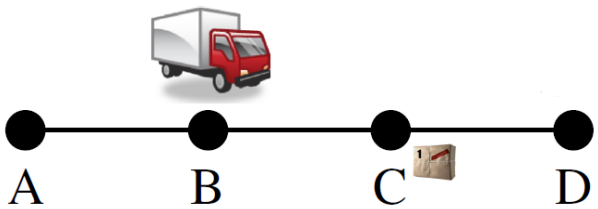
Real problem:

- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{drCB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)



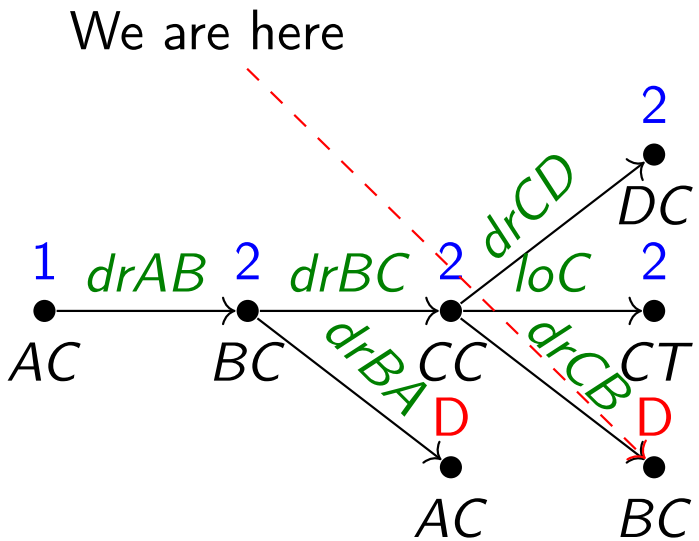
How to Relax During Search: Only-Adds



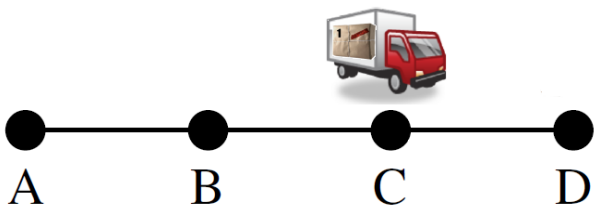
Real problem:

- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)



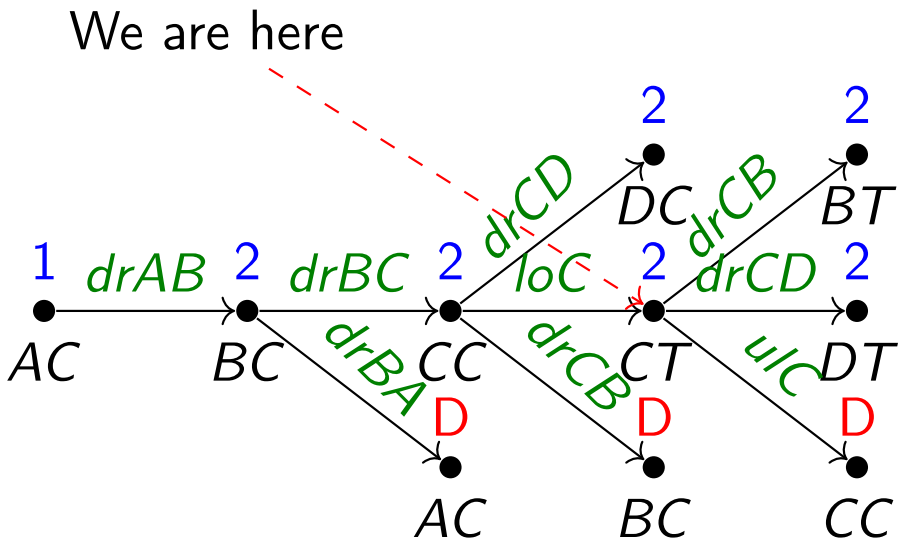
How to Relax During Search: Only-Adds



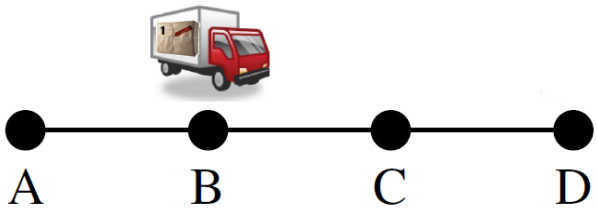
Real problem:

- State s: CT; goal G: AD.
- Actions A: pre, add, del.
- Successors: BT, DT, CC.

Greedy best-first search: (tie-breaking: alphabetic)



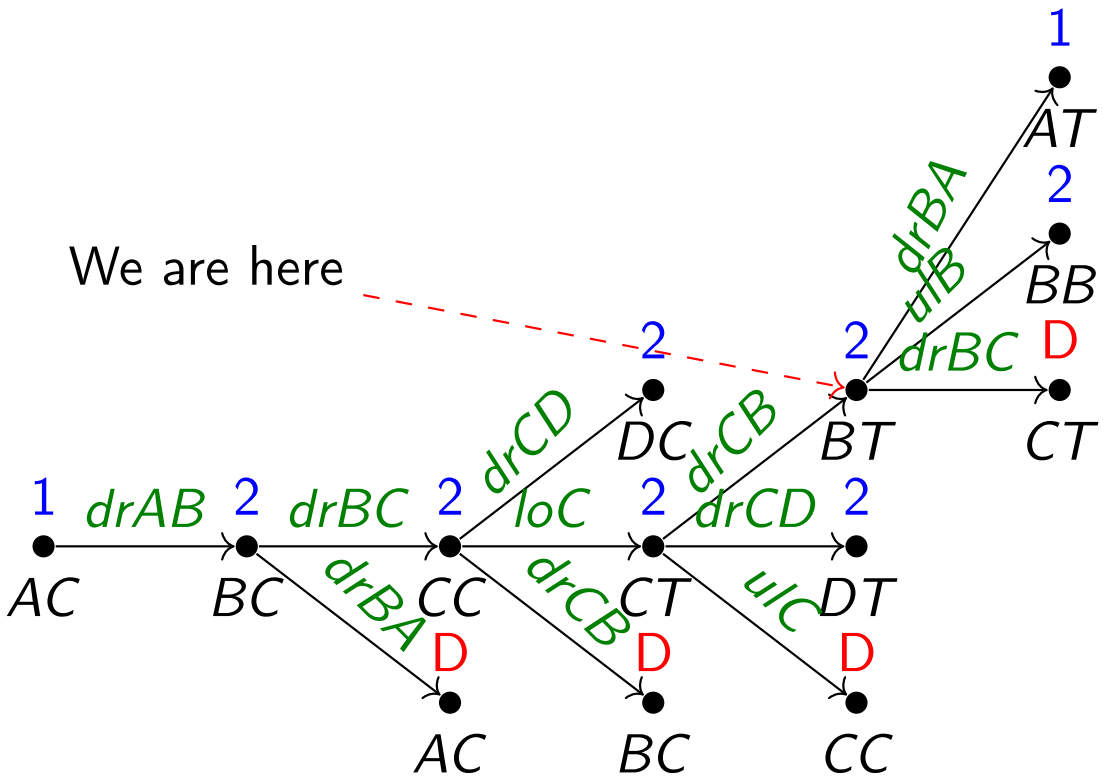
How to Relax During Search: Only-Adds



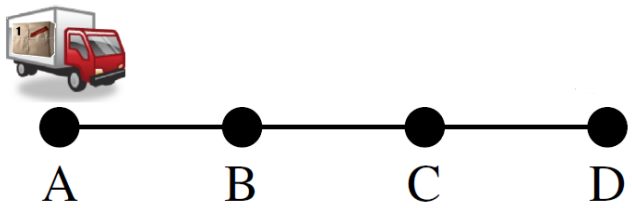
Real problem:

- State s: BT; goal G: AD.
- Actions A: pre, add, del.
- Successors: AT, BB, CT.

Greedy best-first search: (tie-breaking: alphabetic)



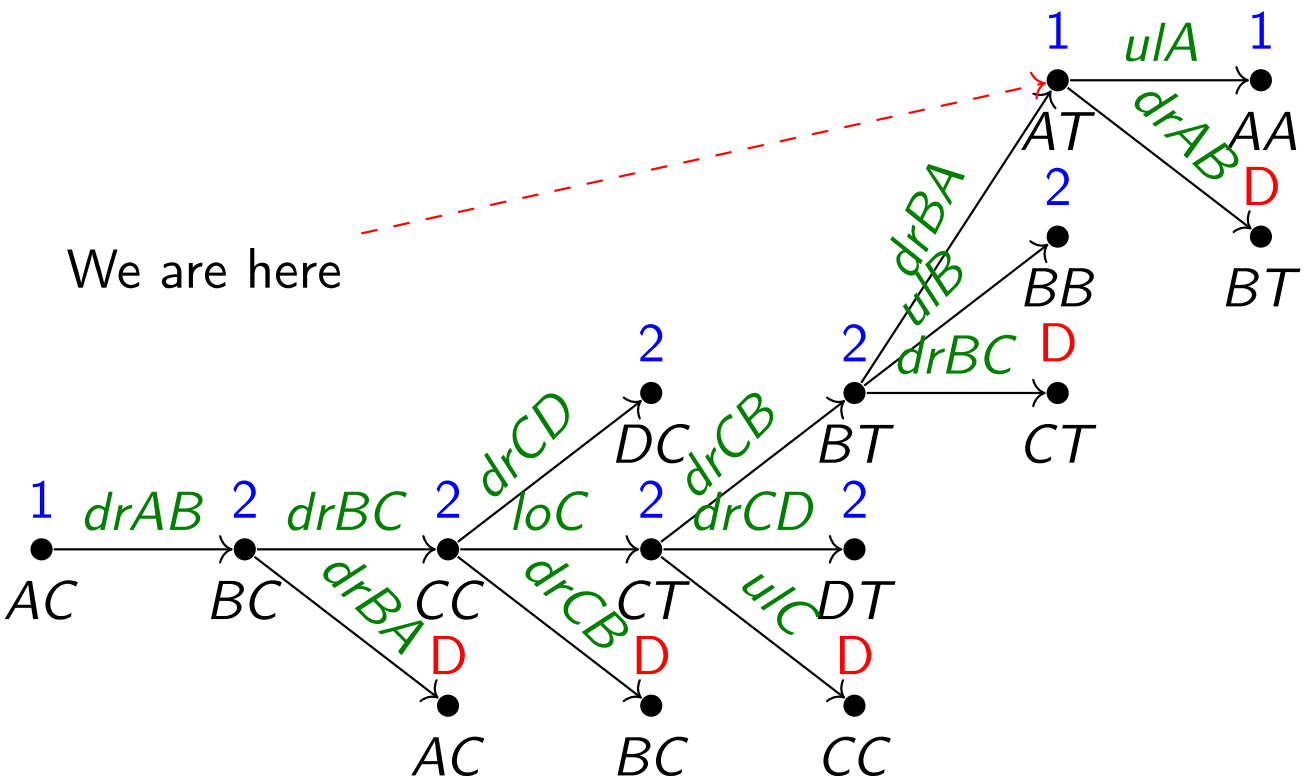
How to Relax During Search: Only-Adds



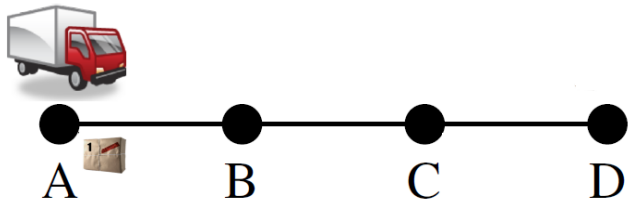
Real problem:

- State s: AT; goal G: AD.
- Actions A: pre, add, del.
- Successors: AA, BT.

Greedy best-first search: (tie-breaking: alphabetic)



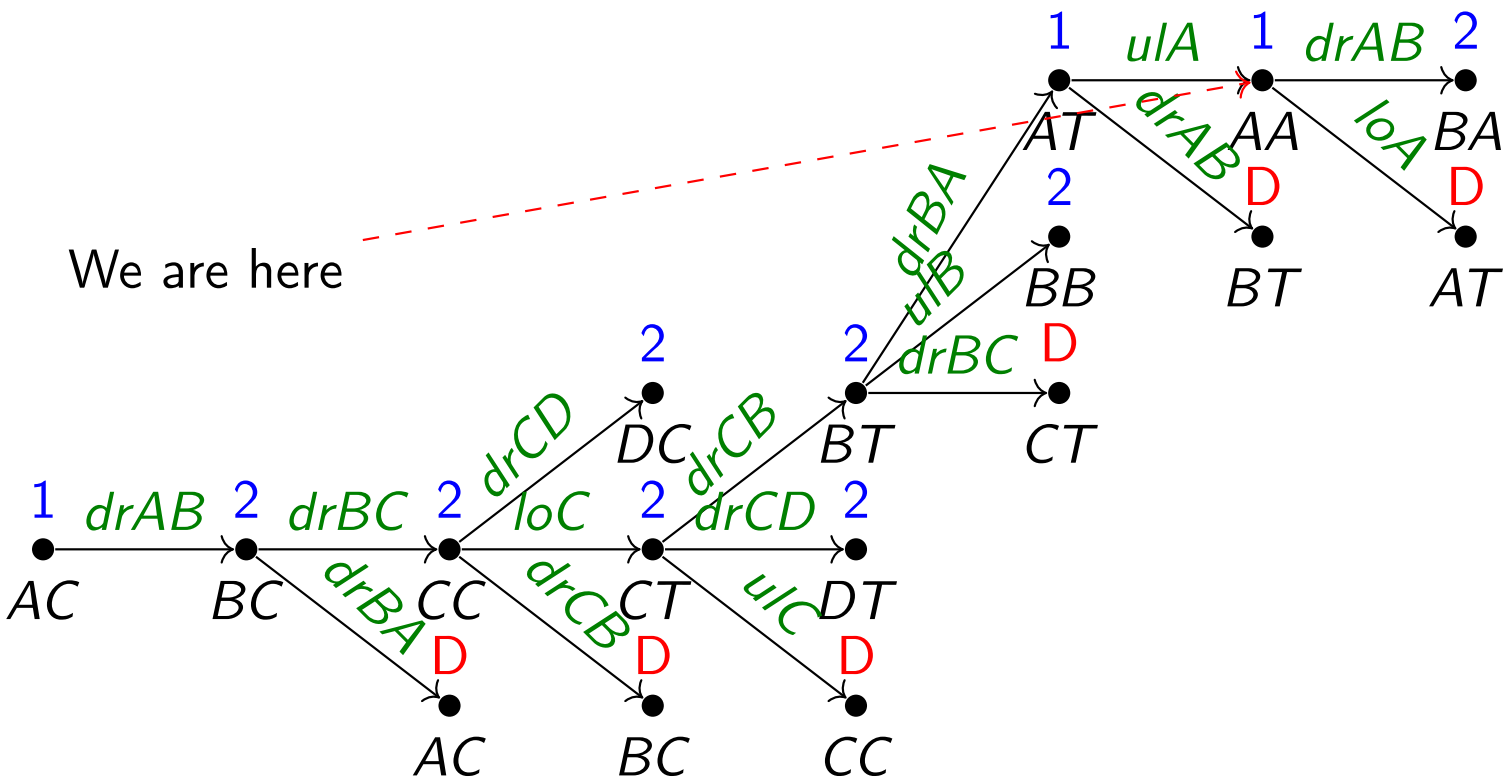
How to Relax During Search: Only-Adds



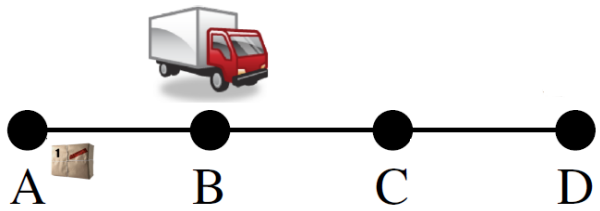
Real problem:

- State s: AA; goal G: AD.
- Actions A: pre, add, del.
- Successors: BA, AT.

Greedy best-first search: (tie-breaking: alphabetic)



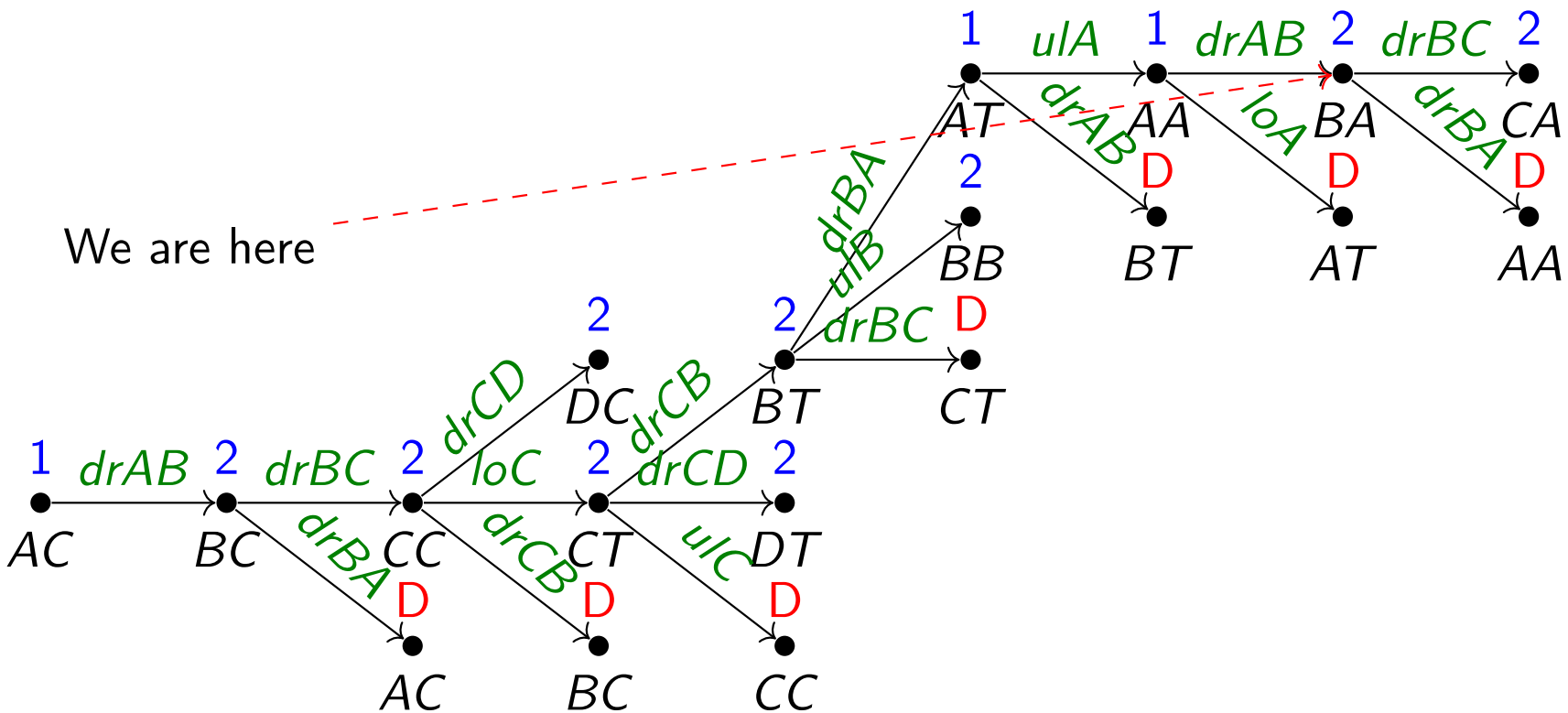
How to Relax During Search: Only-Adds



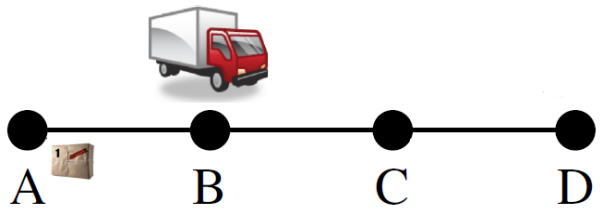
Real problem:

- State s: BA; goal G: AD.
- Actions A: pre, add, del.
- Successors: CA, AA.

Greedy best-first search: (tie-breaking: alphabetic)



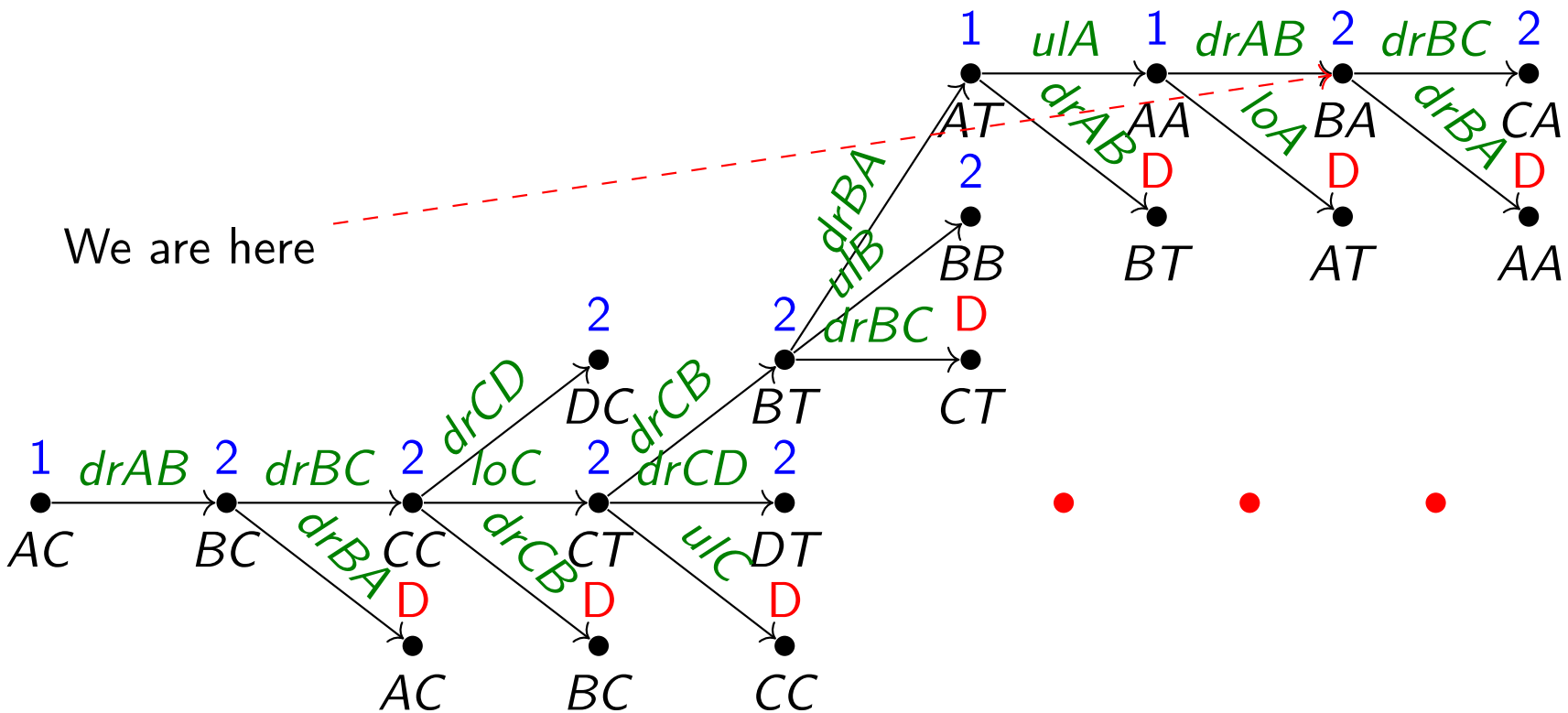
How to Relax During Search: Only-Adds



Real problem:

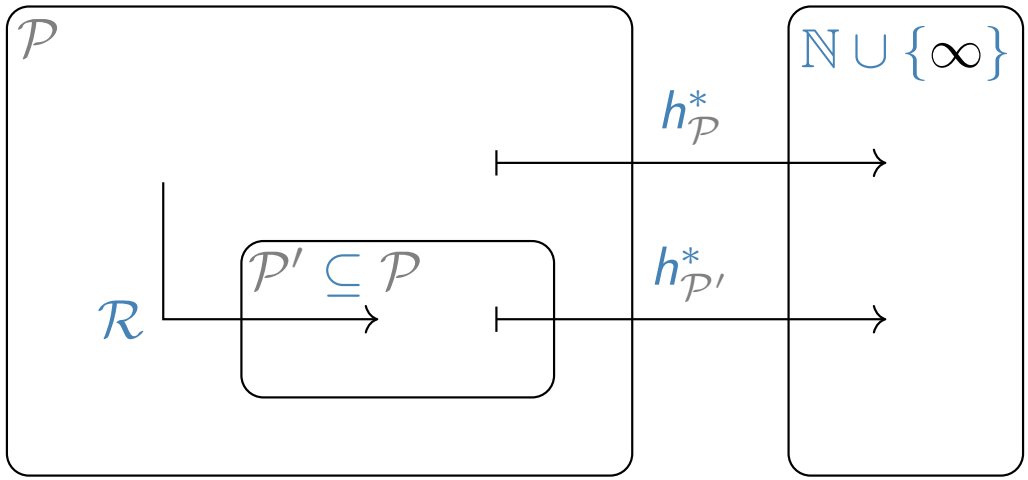
- State s: BA; goal G: AD.
- Actions A: pre, add, del.
- Successors: CA, AA.

Greedy best-first search: (tie-breaking: alphabetic)



Only-Adds is a "Native" Relaxation

- **Definition 2.11 (Native Relaxations).** Confusing special case where $\mathcal{P}' \subseteq \mathcal{P}$.



- **Problem class $\mathcal{P}$**: STRIPS tasks.
- **Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$**: Length h^* of a shortest plan.
- **Transformation $\mathcal{R}$:** Drop the preconditions and delete lists.
- **Simpler problem class $\mathcal{P}'$** is a special case of $\mathcal{P}$, $\mathcal{P}' \subseteq \mathcal{P}$: STRIPS tasks with empty preconditions and delete lists.
- Perfect heuristic for $\mathcal{P}'$: Shortest plan for only-adds STRIPS task.

18.3 The Delete Relaxation

How the Delete Relaxation Changes the World (I)

- Relaxation mapping $\mathcal{R}$ saying that:

  "When the world changes, its previous state remains true as well."

Real world: (before)

How the Delete Relaxation Changes the World (I)

- Relaxation mapping $\mathcal{R}$ saying that:

  "When the world changes, its previous state remains true as well."

Real world: (after)

How the Delete Relaxation Changes the World (I)

- Relaxation mapping $\mathcal{R}$ saying that:

  "When the world changes, its previous state remains true as well."

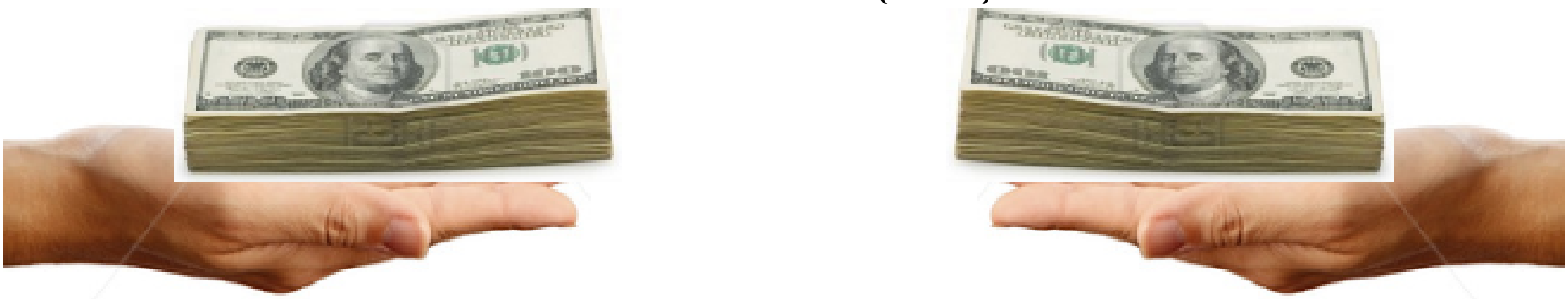
  Relaxed world: (before)

How the Delete Relaxation Changes the World (I)

- Relaxation mapping $\mathcal{R}$ saying that:

  "When the world changes, its previous state remains true as well."

  Relaxed world: (after)



How the Delete Relaxation Changes the World (II)

- Relaxation mapping $\mathcal{R}$ saying that:

Real world: (before)

How the Delete Relaxation Changes the World (II)

- Relaxation mapping $\mathcal{R}$ saying that:

Real world: (after)

How the Delete Relaxation Changes the World (II)

- Relaxation mapping $\mathcal{R}$ saying that:

Relaxed world: (before)

How the Delete Relaxation Changes the World (II)

- Relaxation mapping $\mathcal{R}$ saying that:

Relaxed world: (after)

How the Delete Relaxation Changes the World (III)

- Relaxation mapping $\mathcal{R}$ saying that:

Real world:

How the Delete Relaxation Changes the World (III)

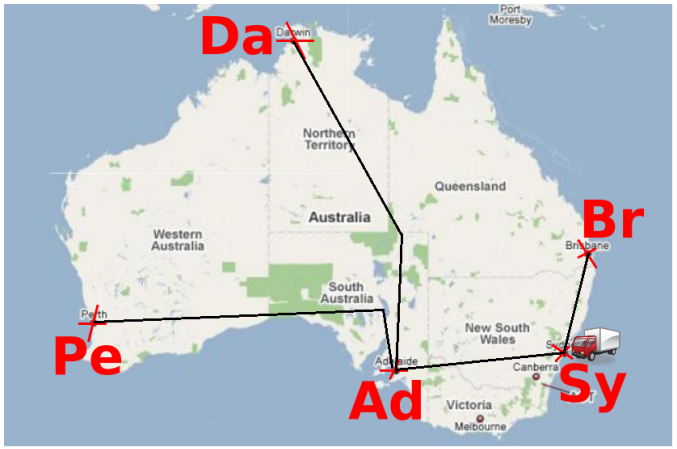
- Relaxation mapping $\mathcal{R}$ saying that:

Relaxed world:

The Delete Relaxation

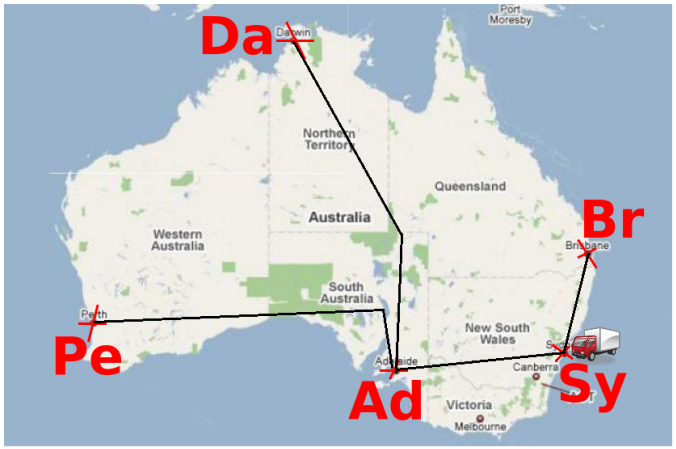
- **Definition 3.1 (Delete Relaxation).** Let $\Pi := \langle P, A, I, G \rangle$ be a STRIPS task. The delete relaxation of Π is the task $\Pi^+ = \langle P, A^+, I, G \rangle$ where $A^+ := \{a^+ \mid a \in A\}$ with $\mathrm{pre}_{a^+} := \mathrm{pre}_a$, $\mathrm{add}_{a^+} := \mathrm{add}_a$, and $\mathrm{del}_{a^+} := \emptyset$.
- In other words, the class of simpler problems $\mathcal{P}'$ is the set of all STRIPS tasks with empty delete lists, and the relaxation mapping $\mathcal{R}$ drops the delete lists.
- **Definition 3.2 (Relaxed Plan).** Let $\Pi := \langle P, A, I, G \rangle$ be a STRIPS task, and let s be a state. A relaxed plan for s is a plan for $\langle P, A, s, G \rangle^+$. A relaxed plan for I is called a relaxed plan for Π.
- A relaxed plan for s is an action sequence that solves s when pretending that all delete lists are empty.
- Also called delete-relaxed plan: "relaxation" is often used to mean delete relaxation by default.

A Relaxed Plan for "TSP" in Australia



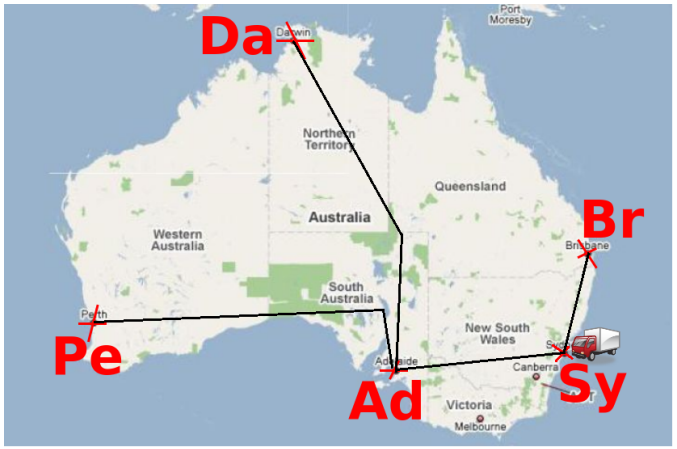
1. **Initial state:** $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.

A Relaxed Plan for "TSP" in Australia



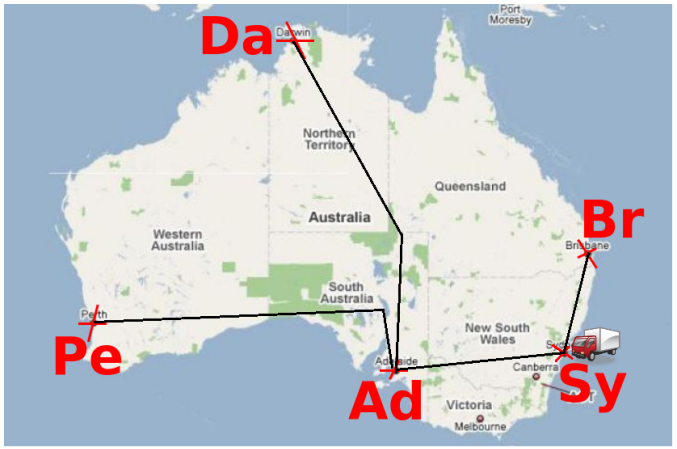
1. **Initial state:** $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
2. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Br})^+$**:** $\{\mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.

A Relaxed Plan for "TSP" in Australia



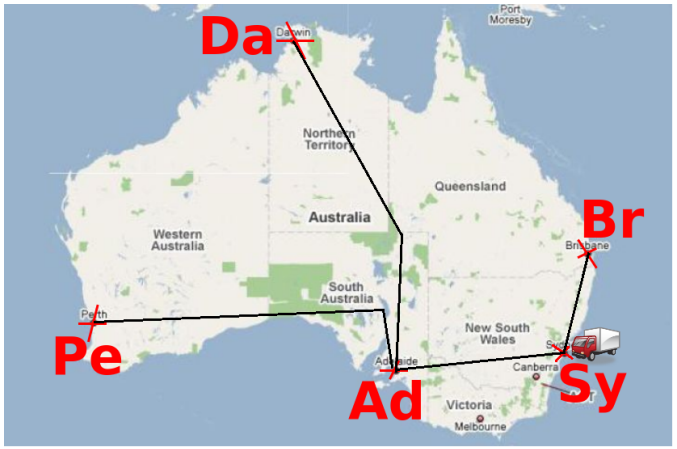
1. **Initial state:** $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
2. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Br})^+$**:** $\{\mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
3. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Ad})^+$**:** $\{\mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.

A Relaxed Plan for "TSP" in Australia



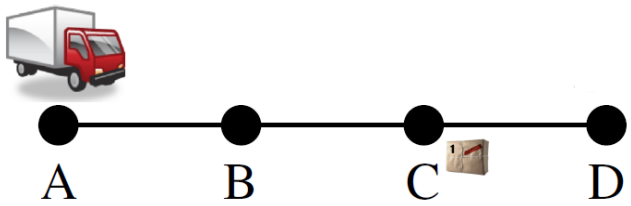
1. **Initial state:** $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
2. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Br})^+$: $\{\mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
3. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Ad})^+$: $\{\mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
4. $\mathrm{drv}(\mathrm{Ad}, \mathrm{Pe})^+$:
   $\{\mathrm{at}(\mathrm{Pe}), \mathrm{vis}(\mathrm{Pe}), \mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.

A Relaxed Plan for "TSP" in Australia



1. **Initial state:** $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
2. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Br})^+$: $\{\mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
3. $\mathrm{drv}(\mathrm{Sy}, \mathrm{Ad})^+$: $\{\mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
4. $\mathrm{drv}(\mathrm{Ad}, \mathrm{Pe})^+$:
   $\{\mathrm{at}(\mathrm{Pe}), \mathrm{vis}(\mathrm{Pe}), \mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.
5. $\mathrm{drv}(\mathrm{Ad}, \mathrm{Da})^+$:
   $\{\mathrm{at}(\mathrm{Da}), \mathrm{vis}(\mathrm{Da}), \mathrm{at}(\mathrm{Pe}), \mathrm{vis}(\mathrm{Pe}), \mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br}), \mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$.

A Relaxed Plan for "Logistics"



- Facts P: $\{\text{truck}(x) \mid x \in \{A, B, C, D\}\} \cup \{\text{pack}(x) \mid x \in \{A, B, C, D, T\}\}$.
- Initial state I: $\{\text{truck}(A), \text{pack}(C)\}$.
- Goal G: $\{\text{truck}(A), \text{pack}(D)\}$.
- Relaxed actions A^+: (Notated as "precondition $\Rightarrow$ adds")
  - $\text{drive}(x, y)^+$: "$\text{truck}(x) \Rightarrow \text{truck}(y)$".
  - $\text{load}(x)^+$: "$\text{truck}(x), \text{pack}(x) \Rightarrow \text{pack}(T)$".
  - $\text{unload}(x)^+$: "$\text{truck}(x), \text{pack}(T) \Rightarrow \text{pack}(x)$".

  Relaxed plan:

$$\langle \text{drive}(A, B)^+, \text{drive}(B, C)^+, \text{load}(C)^+, \text{drive}(C, D)^+, \text{unload}(D)^+ \rangle$$

- We don't need to drive the truck back, because "it is still at A".

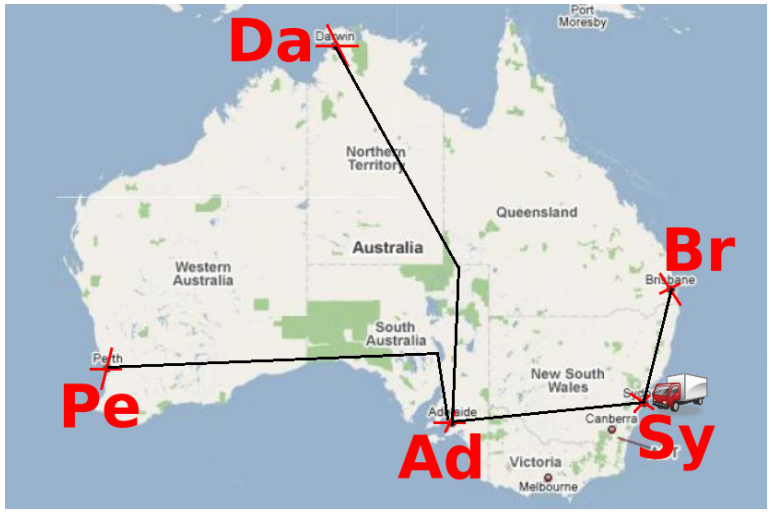
PlanEx$^+$

- **Definition 3.3 (Relaxed Plan Existence Problem).** By PlanEx$^+$, we denote the problem of deciding, given a STRIPS task $\Pi := \langle P, A, I, G \rangle$, whether or not there exists a relaxed plan for Π.
- This is easier than PlanEx for general STRIPS!
- PlanEx$^+$ is in P.
- *Proof:* The following algorithm decides PlanEx$^+$
  1.

```
var F := I
while G ⊈ F do
    F' := F ∪ ⋃_{a∈A:pre_a⊆F} add_a
    if F' = F then return "unsolvable" endif        (*)
    F := F'
endwhile
return "solvable"
```

  2. The algorithm terminates after at most $|P|$ iterations, and thus runs in polynomial time.
  3. Correctness: See slide 634

Deciding PlanEx$^+$ in "TSP" in Australia

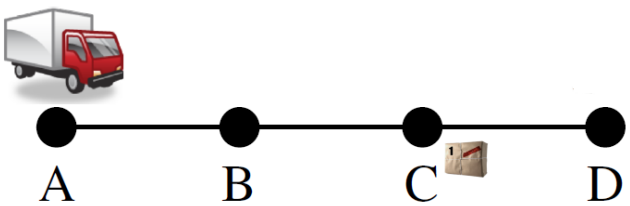


Iterations on F:

1. $\{\mathrm{at}(\mathrm{Sy}), \mathrm{vis}(\mathrm{Sy})\}$
2. $\cup\, \{\mathrm{at}(\mathrm{Ad}), \mathrm{vis}(\mathrm{Ad}), \mathrm{at}(\mathrm{Br}), \mathrm{vis}(\mathrm{Br})\}$
3. $\cup\, \{\mathrm{at}(\mathrm{Da}), \mathrm{vis}(\mathrm{Da}), \mathrm{at}(\mathrm{Pe}), \mathrm{vis}(\mathrm{Pe})\}$

Deciding PlanEx$^+$ in "Logistics"

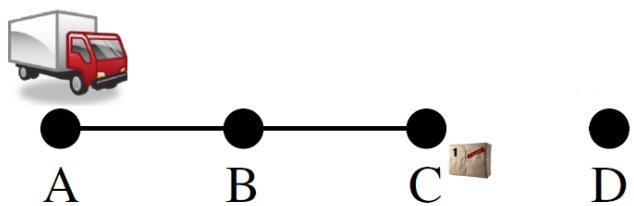
- **Example 3.4 (The solvable Case).**



Iterations on F:

1. $\{\text{truck}(A), \text{pack}(C)\}$
2. $\cup\{\text{truck}(B)\}$
3. $\cup\{\text{truck}(C)\}$
4. $\cup\{\text{truck}(D), \text{pack}(T)\}$
5. $\cup\{\text{pack}(A), \text{pack}(B), \text{pack}(D)\}$

- **Example 3.5 (The unsolvable Case).**



Iterations on F:

1. $\{\text{truck}(A), \text{pack}(C)\}$
2. $\cup\{\text{truck}(B)\}$
3. $\cup\{\text{truck}(C)\}$
4. $\cup\{\text{pack}(T)\}$
5. $\cup\{\text{pack}(A), \text{pack}(B)\}$
6. $\cup\emptyset$

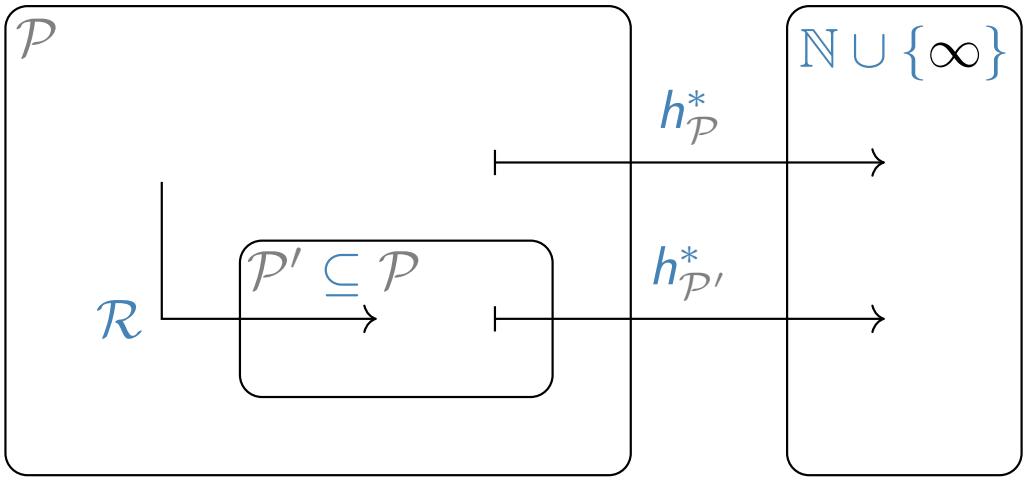
PlanEx$^+$ Algorithm: Proof

Proof: To show: The algorithm returns "solvable" iff there is a relaxed plan for Π.

1. Denote by F_i the content of F after the ith iteration of the while-loop,
2. All $a \in A_0$ are applicable in I, all $a \in A_1$ are applicable in $\mathrm{apply}(I, A_0^+)$, and so forth.
3. Thus $F_i = \mathrm{apply}(I, \langle A_0^+, \ldots, A_{i-1}^+ \rangle)$. (Within each A_j^+, we can sequence the actions in any order.)
4. Direction "$\Rightarrow$" *If "solvable" is returned after iteration n then $G \subseteq F_n = \mathrm{apply}(I, \langle A_0^+, \ldots, A_{n-1}^+ \rangle)$ so $\langle A_0^+, \ldots, A_{n-1}^+ \rangle$ can be sequenced to a relaxed plan which shows the claim.*
5. Direction "$\Leftarrow$"
   - 5.1. Let $\langle a_0^+, \ldots, a_{n-1}^+ \rangle$ be a relaxed plan, hence $G \subseteq \mathrm{apply}(I, \langle a_0^+, \ldots, a_{n-1}^+ \rangle)$.
   - 5.2. Assume, for the moment, that we drop line (*) from the algorithm. It is then easy to see that $a_i \in A_i$ and $\mathrm{apply}(I, \langle a_0^+, \ldots, a_{i-1}^+ \rangle) \subseteq F_i$, for all i.
   - 5.3. We get $G \subseteq \mathrm{apply}(I, \langle a_0^+, \ldots, a_{n-1}^+ \rangle) \subseteq F_n$, and the algorithm returns "solvable" as desired.
   - 5.4. Assume to the contrary of the claim that, in an iteration $i < n$, (*) fires. Then $G \not\subseteq F$ and $F = F'$. But, with $F = F'$, $F = F_j$ for all $j > i$, and we get $G \not\subseteq F_n$ in contradiction.

18.4 The h^+ Heuristic

Hold on a Sec – Where are we?



- $\mathcal{P}$: STRIPS tasks; $h^*_{\mathcal{P}}$: Length h^* of a shortest plan.
- $\mathcal{P}' \subseteq \mathcal{P}$: STRIPS tasks with empty delete lists.
- $\mathcal{R}$: Drop the delete lists.
- Heuristic function: Length of a shortest relaxed plan ($h^* \circ \mathcal{R}$).
- PlanEx$^+$ is not actually what we're looking for. PlanEx$^+$ $\widehat{=}$ relaxed plan *existence*; we want relaxed plan *length* $h^* \circ \mathcal{R}$.

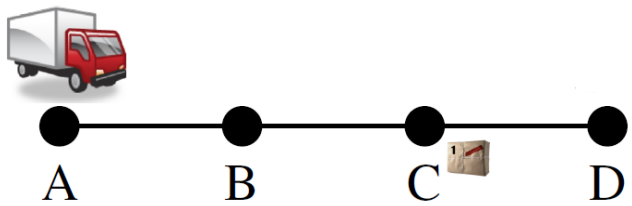
h^+: The Ideal Delete Relaxation Heuristic

- **Definition 4.1 (Optimal Relaxed Plan).** Let $\langle P, A, I, G\rangle$ be a STRIPS task, and let s be a state. A optimal relaxed plan for s is an optimal plan for $\langle P, A, \{s\}, G\rangle^+$.
- Same as slide 628, just adding the word "optimal".
- Here's what we're looking for:
- **Definition 4.2.** Let $\Pi := \langle P, A, I, G\rangle$ be a STRIPS task with states S. The ideal delete relaxation heuristic h^+ for Π is the function $h^+ : S \to \mathbb{N} \cup \{\infty\}$ where $h^+(s)$ is the length of an optimal relaxed plan for s if a relaxed plan for s exists, and $h^+(s) = \infty$ otherwise.
- In other words, $h^+ = h^* \circ \mathcal{R}$, cf. previous slide.

h^+ is Admissible

- **Lemma 4.3.** *Let $\Pi := \langle P, A, I, G \rangle$ be a STRIPS task, and let s be a state. If $\langle a_1, \ldots, a_n \rangle$ is a plan for $\Pi_s := \langle P, A, \{s\}, G \rangle$, then $\langle a_1^+, \ldots, a_n^+ \rangle$ is a plan for Π^+.*
- *Proof sketch:* Show by induction over $0 \leq i \leq n$ that $\mathrm{apply}(s, \langle a_1, \ldots, a_i \rangle) \subseteq \mathrm{apply}(s, \langle a_1^+, \ldots, a_i^+ \rangle)$.
- If we ignore deletes, the states along the plan can only get bigger.
- **Theorem 4.4.** *h^+ is Admissible.*
- *Proof:*
  1. Let $\Pi := \langle P, A, I, G \rangle$ be a STRIPS task with states P, and let $s \in P$.
  2. $h^+(s)$ is defined as optimal plan length in Π_s^+.
  3. With the lemma above, any plan for Π also constitutes a plan for Π_s^+.
  4. Thus optimal plan length in Π_s^+ can only be shorter than that in $\Pi_s i$, and the claim follows.

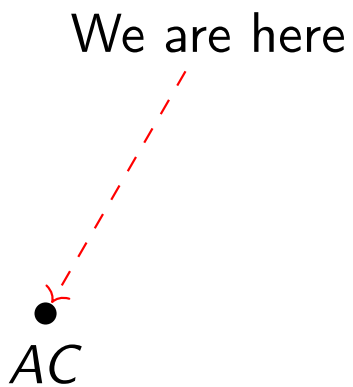
How to Relax During Search: Ignoring Deletes



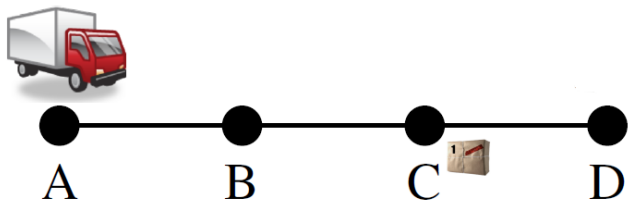
Real problem:

- Initial state I: AC; goal G: AD.
- Actions A: pre, add, del.
- $drXY$, loX, ulX.

Greedy best-first search: (tie-breaking: alphabetic)



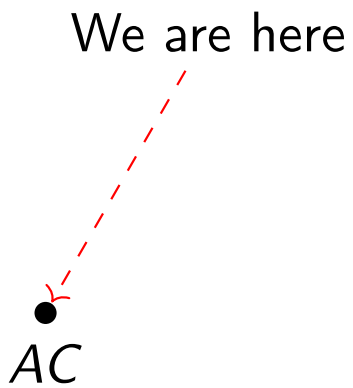
How to Relax During Search: Ignoring Deletes



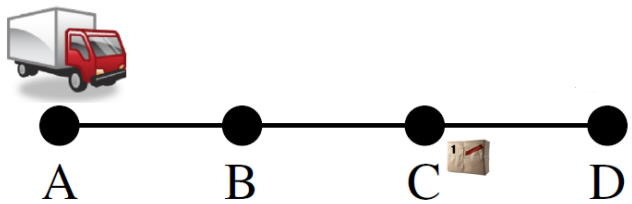
Relaxed problem:

- State s: AC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

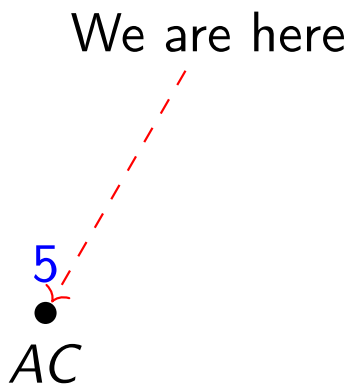


Relaxed problem:

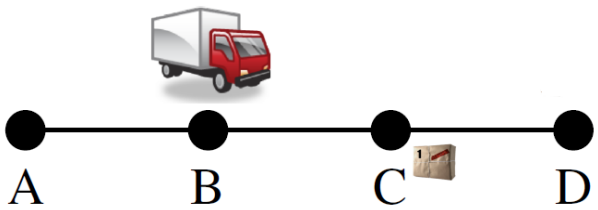
- State s: AC; goal G: AD.
- Actions A: pre, add.
- $h^+(s)$ =5: e.g. $\langle drAB, drBC, drCD, loC, ulD \rangle$.

(tie-breaking: alphabetic)

Greedy best-first search:



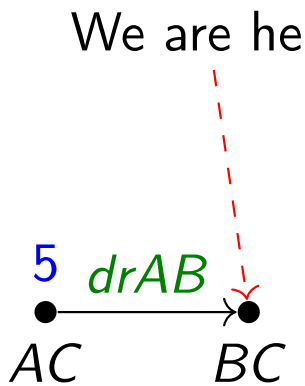
How to Relax During Search: Ignoring Deletes



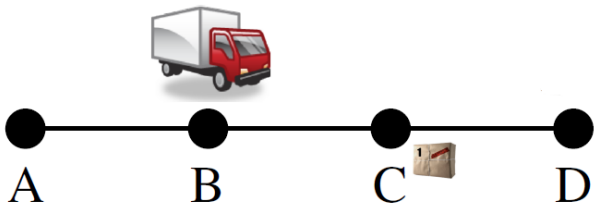
Real problem:

- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- $AC \xrightarrow{drAB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)



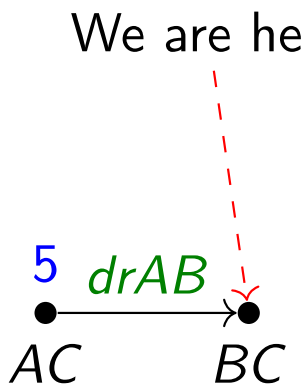
How to Relax During Search: Ignoring Deletes



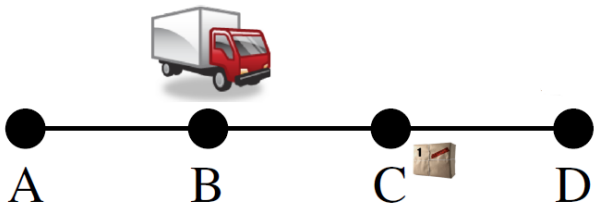
Relaxed problem:

- State s: BC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



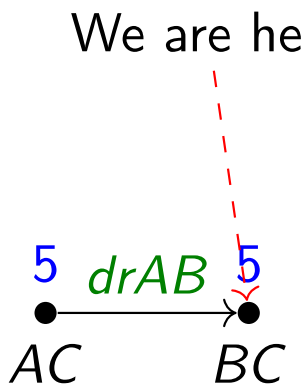
How to Relax During Search: Ignoring Deletes



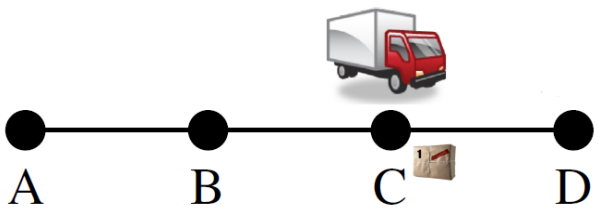
Relaxed problem:

- State s: BC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s) = 5$: e.g. $\langle drBA, drBC, drCD, loC, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



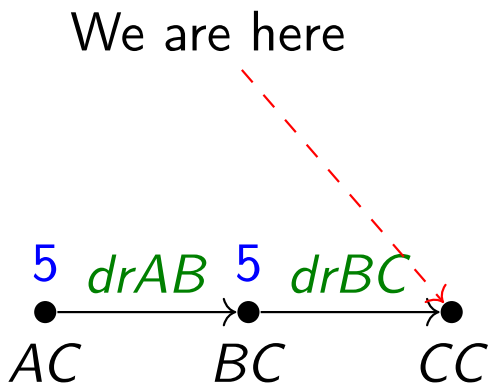
How to Relax During Search: Ignoring Deletes



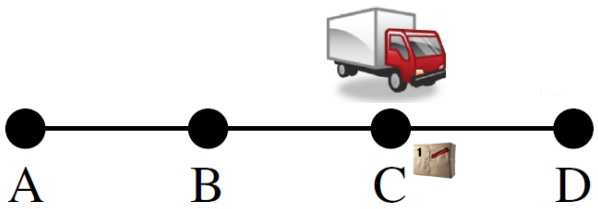
Real problem:

- State s: CC; goal G: AD.
- Actions A: pre, add, del.
- $BC \xrightarrow{drBC} CC$.

Greedy best-first search: (tie-breaking: alphabetic)



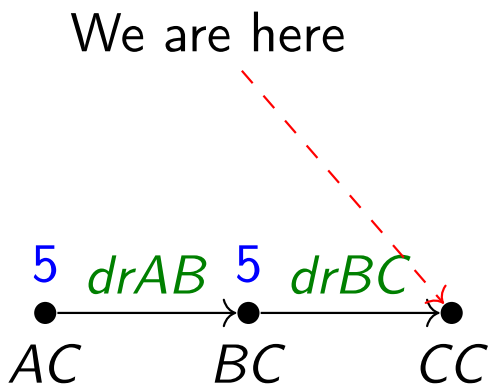
How to Relax During Search: Ignoring Deletes



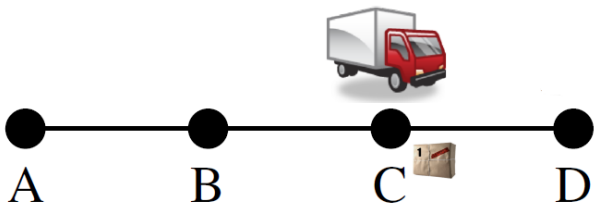
Relaxed problem:

- State s: CC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



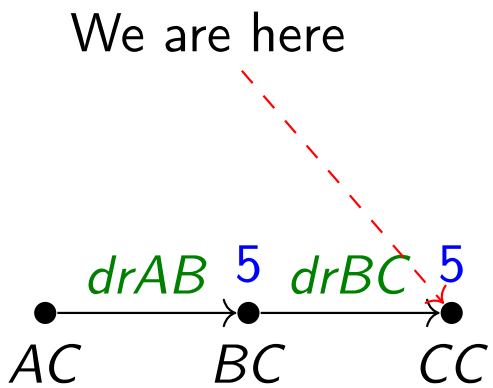
How to Relax During Search: Ignoring Deletes



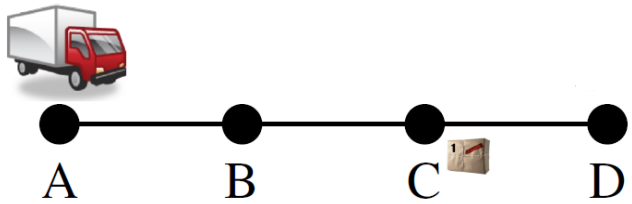
Relaxed problem:

- State s: CC; goal G: AD.
- Actions A: pre, add.
- $h^+(s) = 5$: e.g. $\langle drCB, drBA, drCD, loC, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



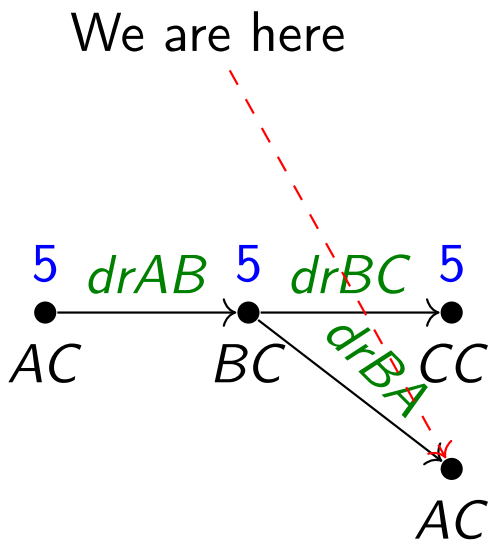
How to Relax During Search: Ignoring Deletes



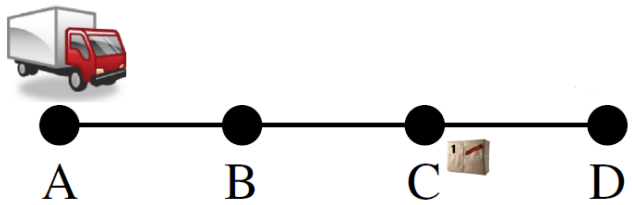
Real problem:

- State s: AC; goal G: AD.
- Actions A: pre, add, del.
- $BC \xrightarrow{drBA} AC$.

Greedy best-first search: (tie-breaking: alphabetic)



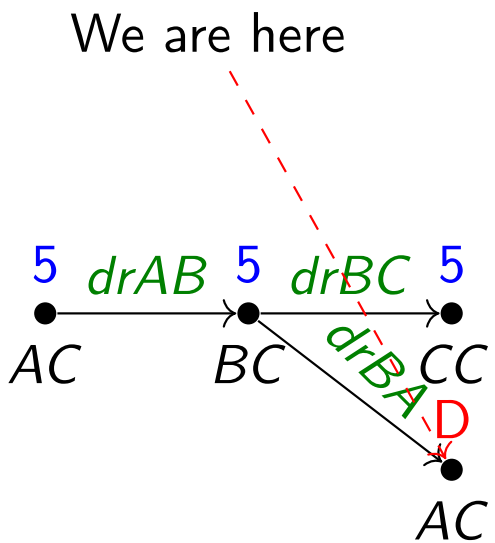
How to Relax During Search: Ignoring Deletes



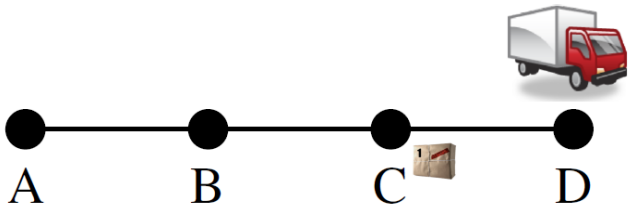
Real problem:

- State s: AC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)



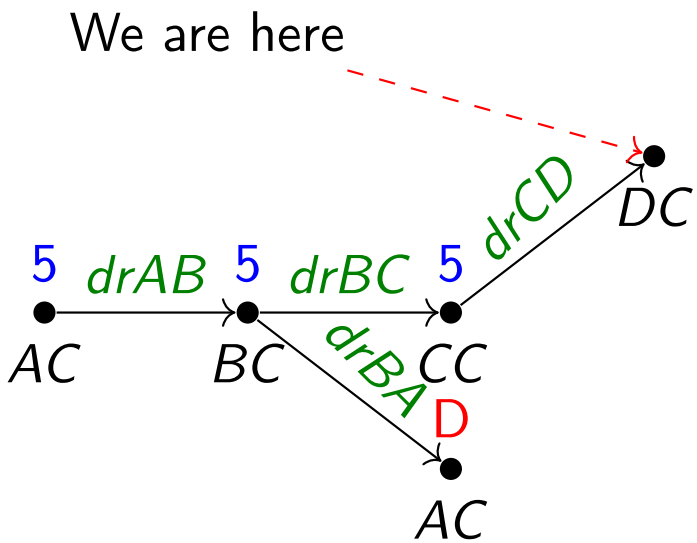
How to Relax During Search: Ignoring Deletes



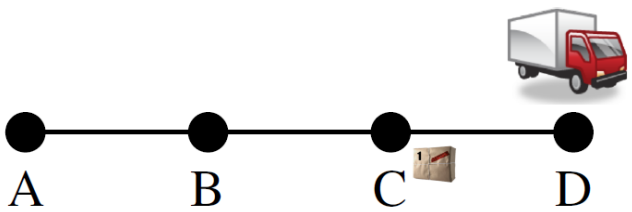
Real problem:

- State s: DC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $CC \xrightarrow{drCD} DC$.

Greedy best-first search: (tie-breaking: alphabetic)



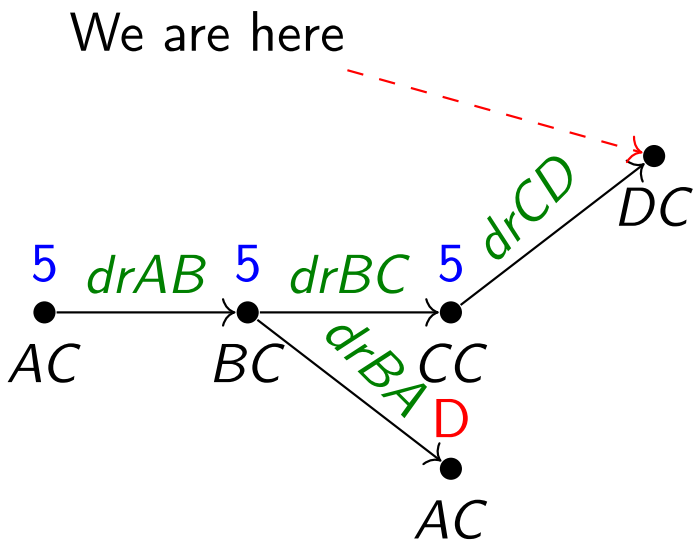
How to Relax During Search: Ignoring Deletes



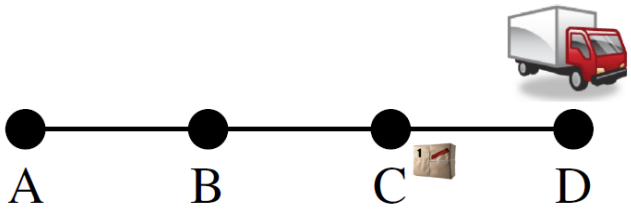
Relaxed problem:

- State s: DC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



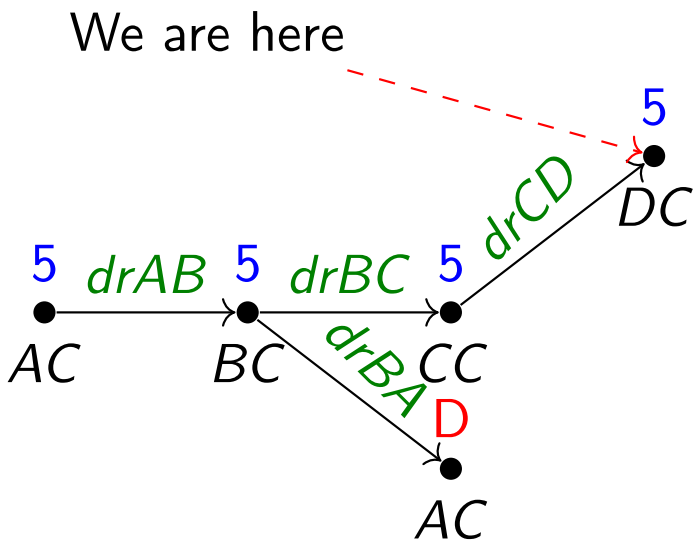
How to Relax During Search: Ignoring Deletes



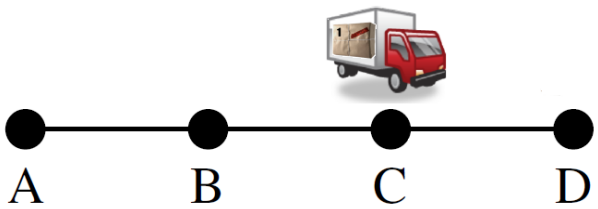
Relaxed problem:

- State s: DC; goal G: AD.
- Actions A: pre, add.
- $h^+(s)$ =5: e.g. $\langle drDC, drCB, drBA, loC, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



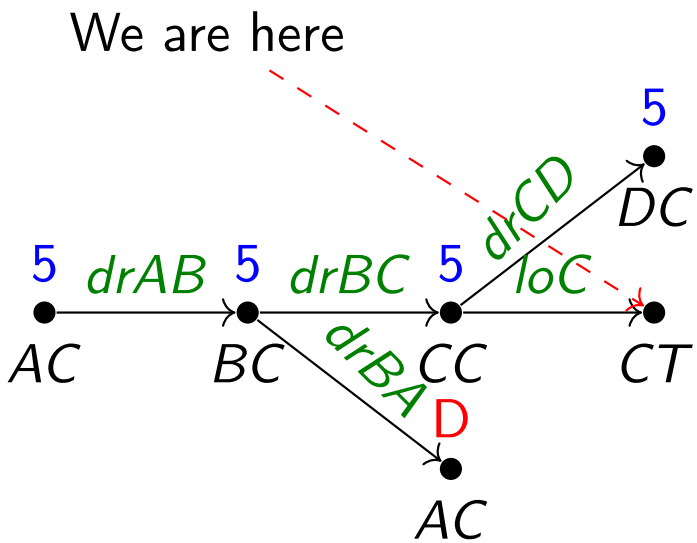
How to Relax During Search: Ignoring Deletes



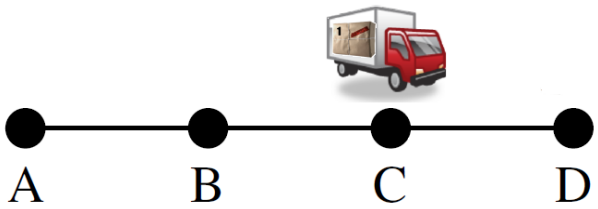
Real problem:

- State s: CT; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $CC \xrightarrow{loC} CT$.

Greedy best-first search: (tie-breaking: alphabetic)



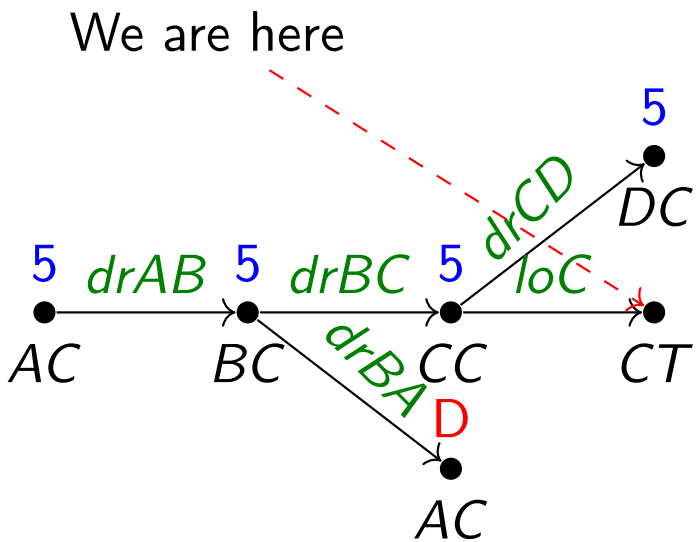
How to Relax During Search: Ignoring Deletes



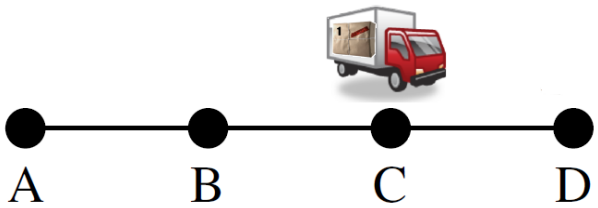
Relaxed problem:

- State s: CT; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}$.
- $h^+(s) =$

Greedy best-first search: (tie-breaking: alphabetic)



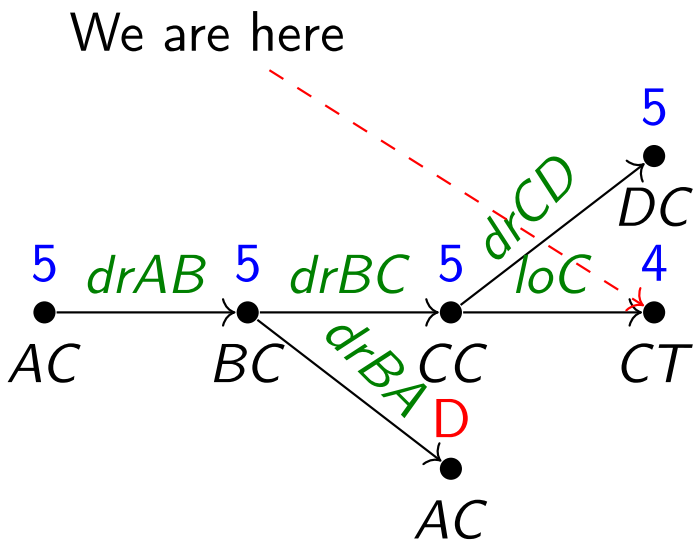
How to Relax During Search: Ignoring Deletes



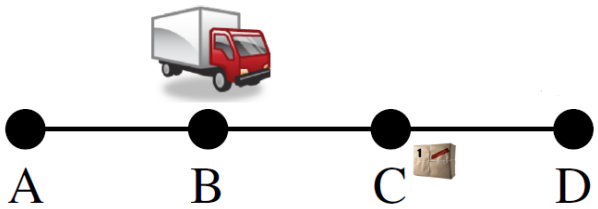
Relaxed problem:

- State s: CT; goal G: AD.
- Actions A: pre, add.
- $h^+(s) = 4$: e.g. $\langle drCB, drBA, drCD, ulD \rangle$.

Greedy best-first search: (tie-breaking: alphabetic)



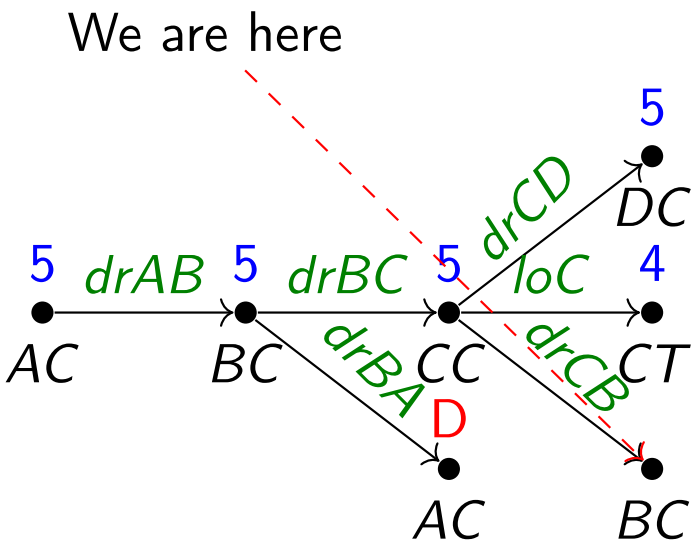
How to Relax During Search: Ignoring Deletes



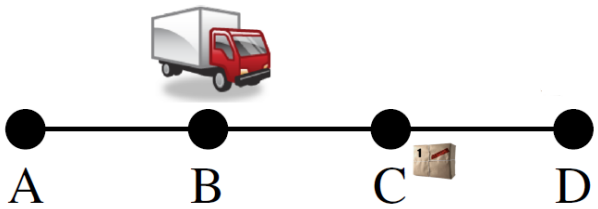
Real problem:

- State s: BC; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- $CC \xrightarrow{drCB} BC$.

Greedy best-first search: (tie-breaking: alphabetic)



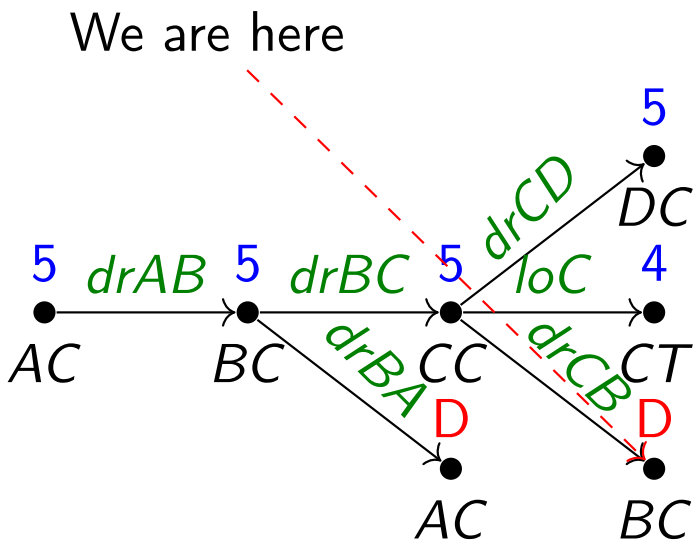
How to Relax During Search: Ignoring Deletes



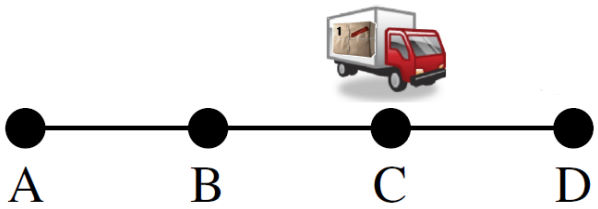
Real problem:

- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search: (tie-breaking: alphabetic)



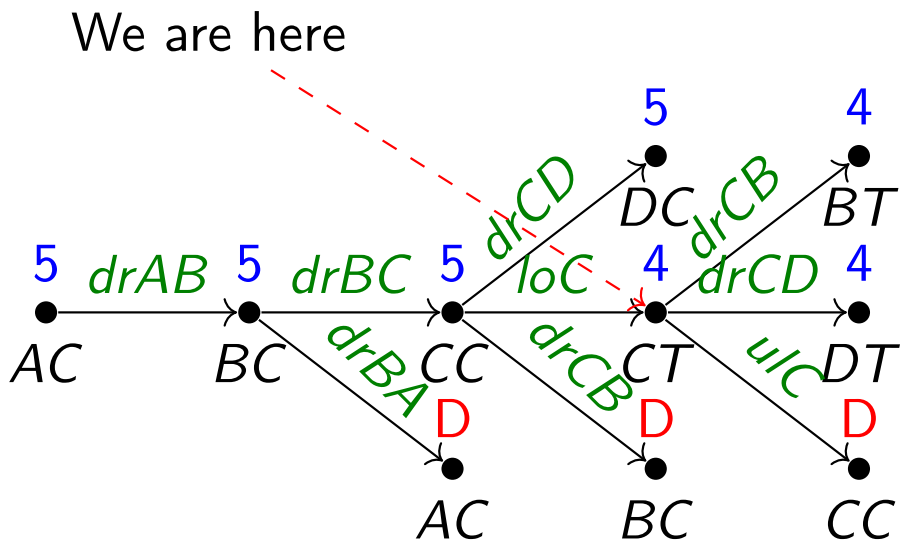
How to Relax During Search: Ignoring Deletes



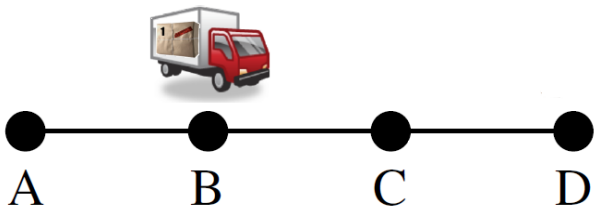
Real problem:

- State s: CT; goal G: AD.
- Actions A: pre, add, del.
- Successors: BT, DT, CC.

Greedy best-first search: (tie-breaking: alphabetic)



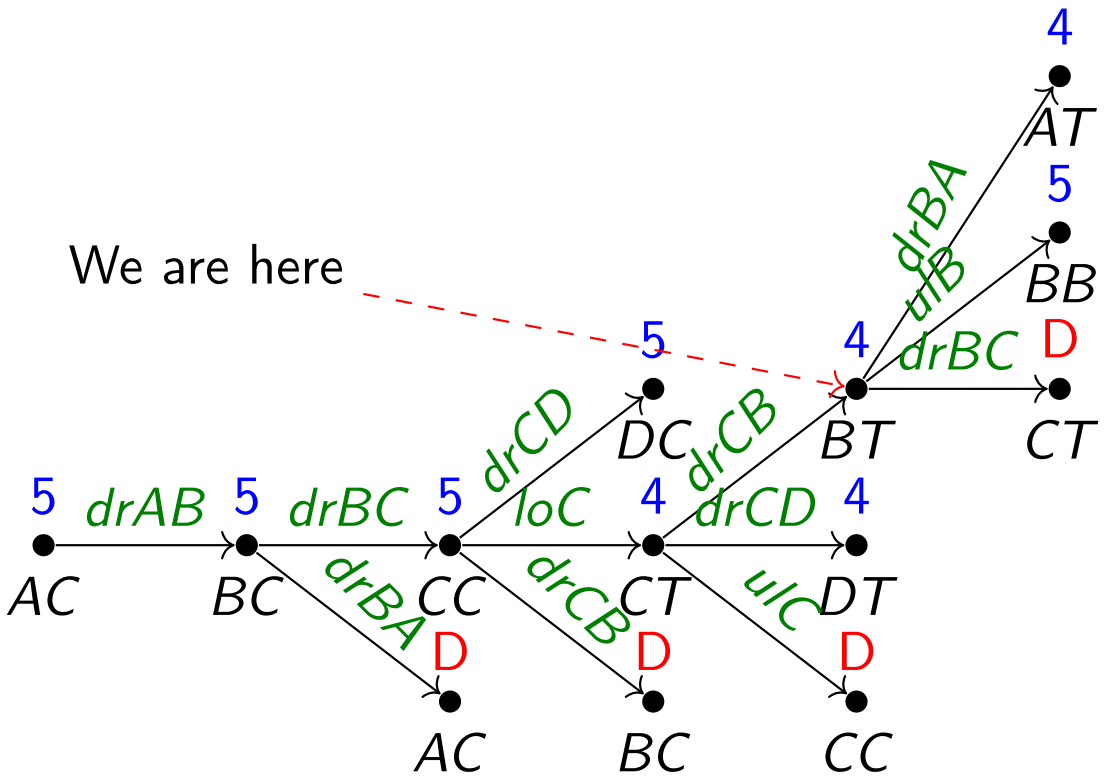
How to Relax During Search: Ignoring Deletes



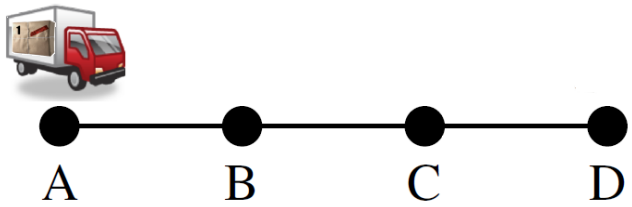
Real problem:

- State s: BT; goal G: AD.
- Actions A: pre, add, del.
- Successors: AT, BB, CT.

Greedy best-first search: (tie-breaking: alphabetic)



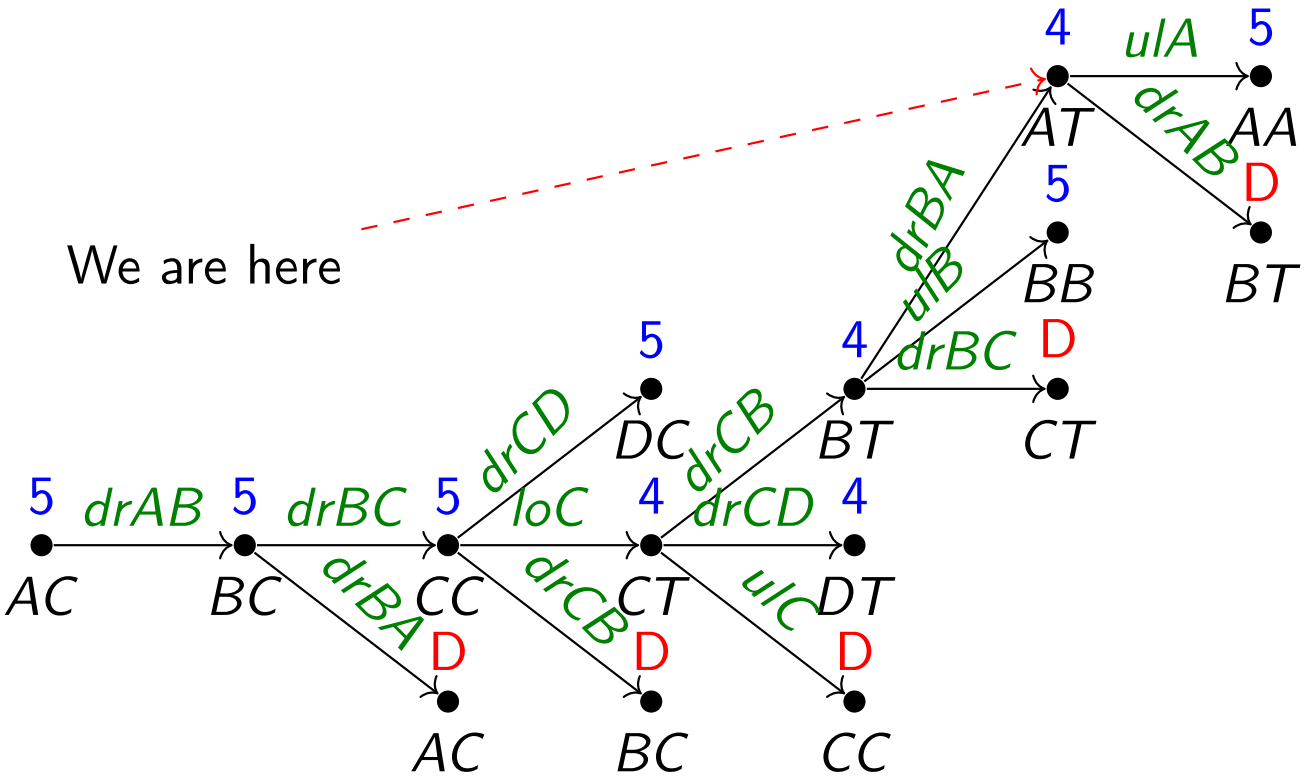
How to Relax During Search: Ignoring Deletes



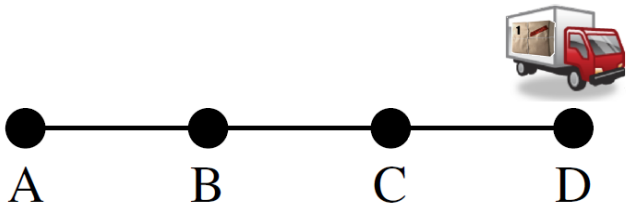
Real problem:

- State s: AT; goal G: AD.
- Actions A: pre, add, del.
- Successors: AA, BT.

Greedy best-first search: (tie-breaking: alphabetic)



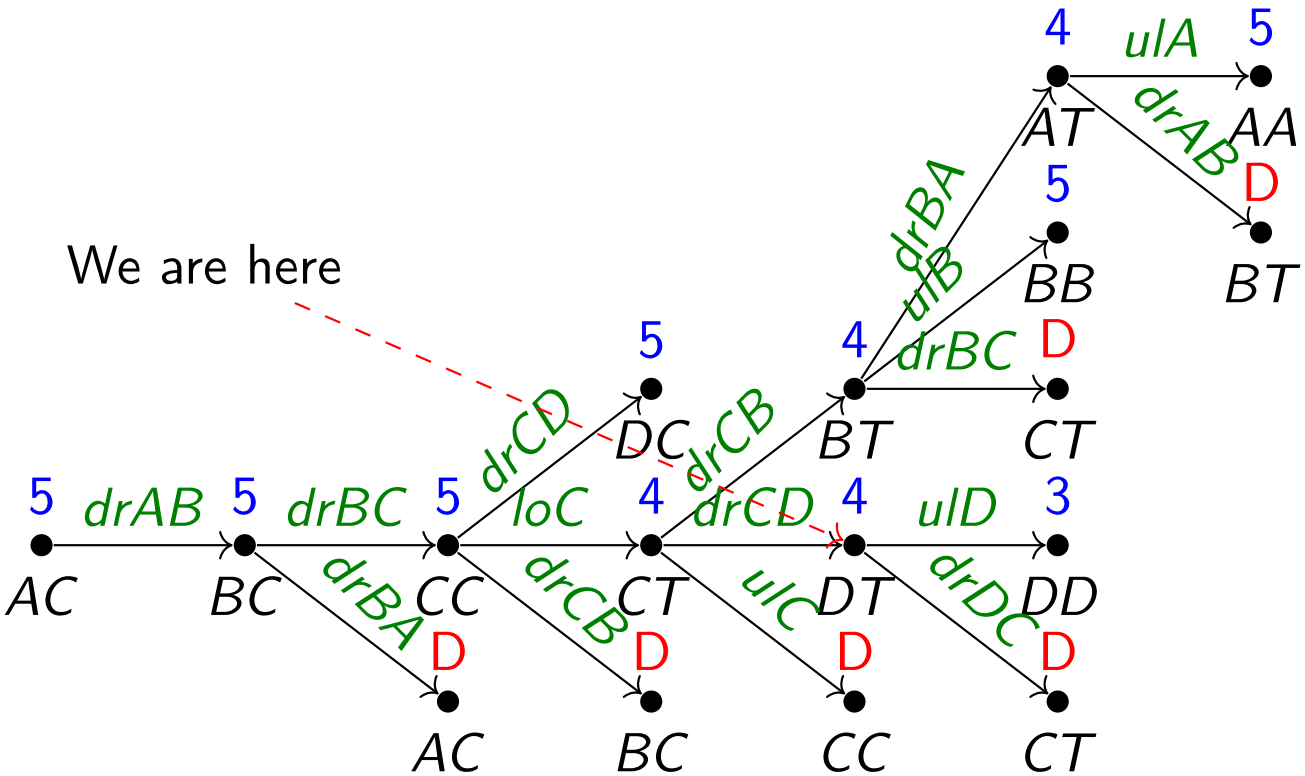
How to Relax During Search: Ignoring Deletes



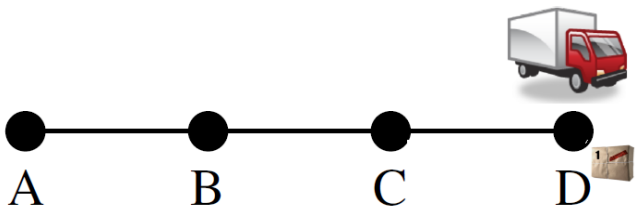
Real problem:

- State s: DT; goal G: AD.
- Actions A: pre, add, del.
- Successors: DD, CT.

Greedy best-first search: (tie-breaking: alphabetic)



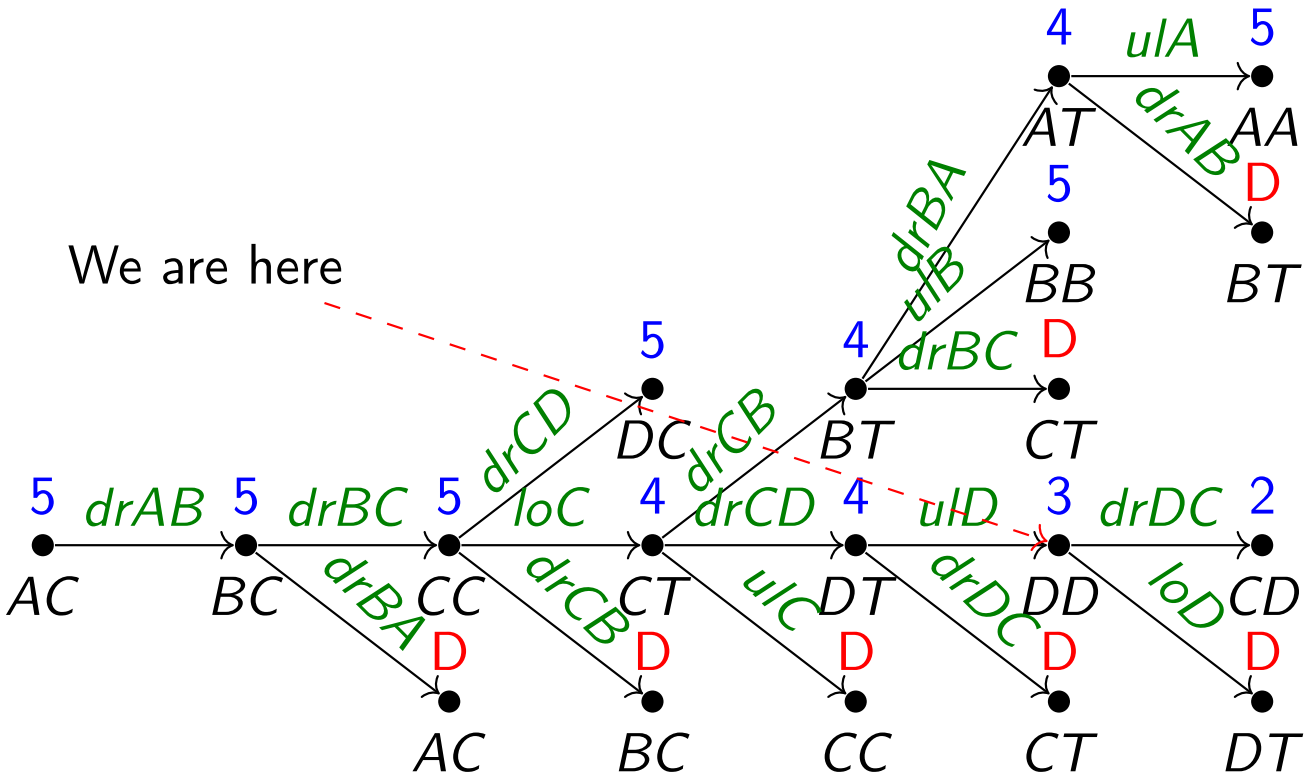
How to Relax During Search: Ignoring Deletes



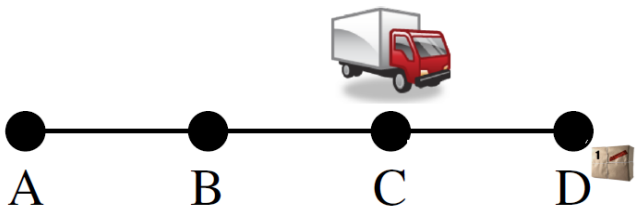
Real problem:

- State s: DD; goal G: AD.
- Actions A: pre, add, del.
- Successors: CD, DT.

Greedy best-first search: (tie-breaking: alphabetic)



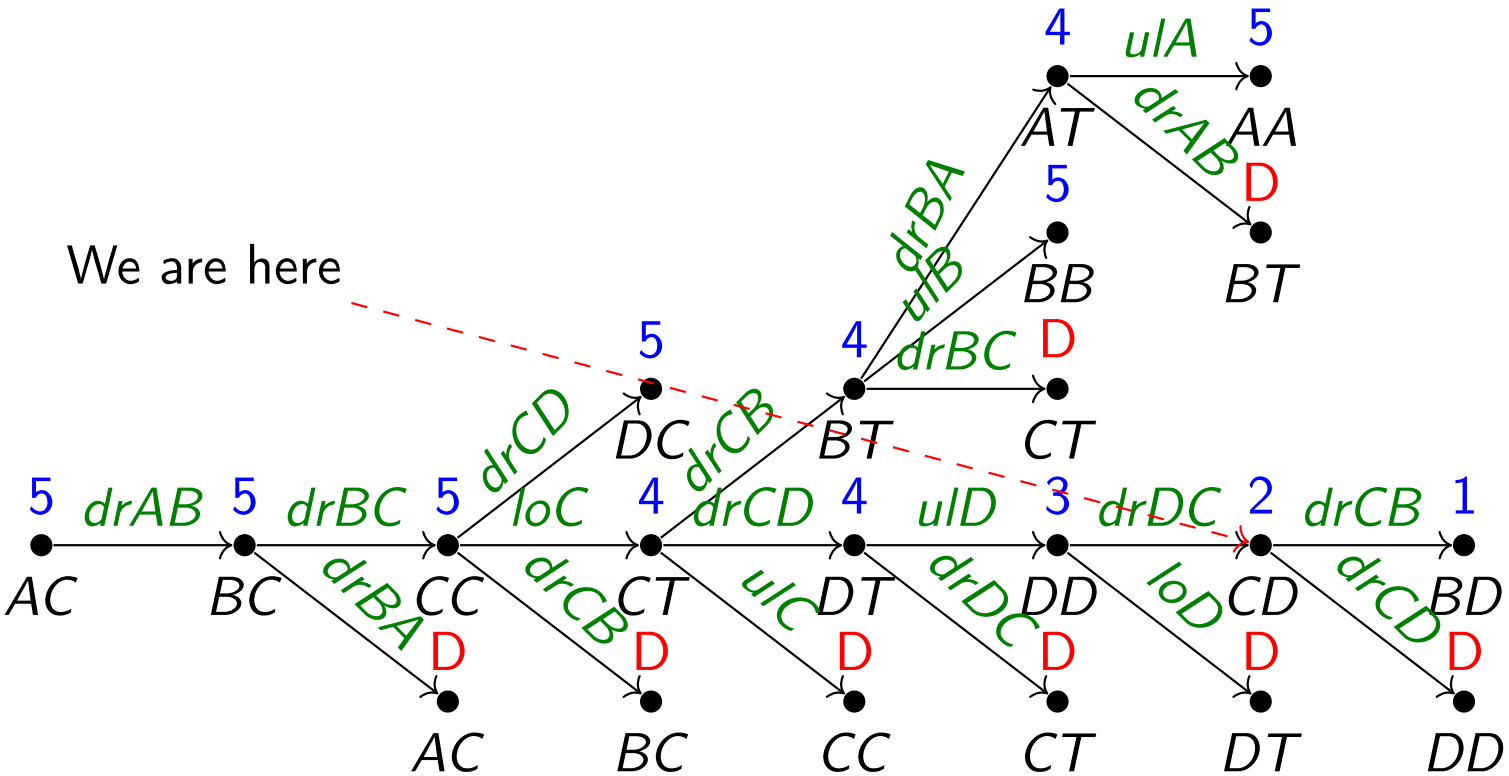
How to Relax During Search: Ignoring Deletes



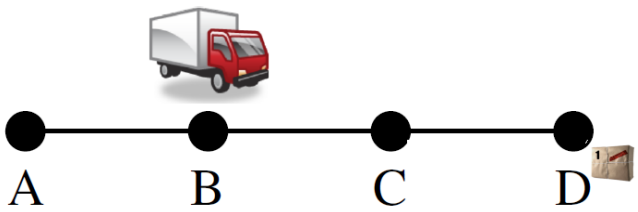
Real problem:

- State s: CD; goal G: AD.
- Actions A: pre, add, del.
- Successors: BD, DD.

Greedy best-first search: (tie-breaking: alphabetic)



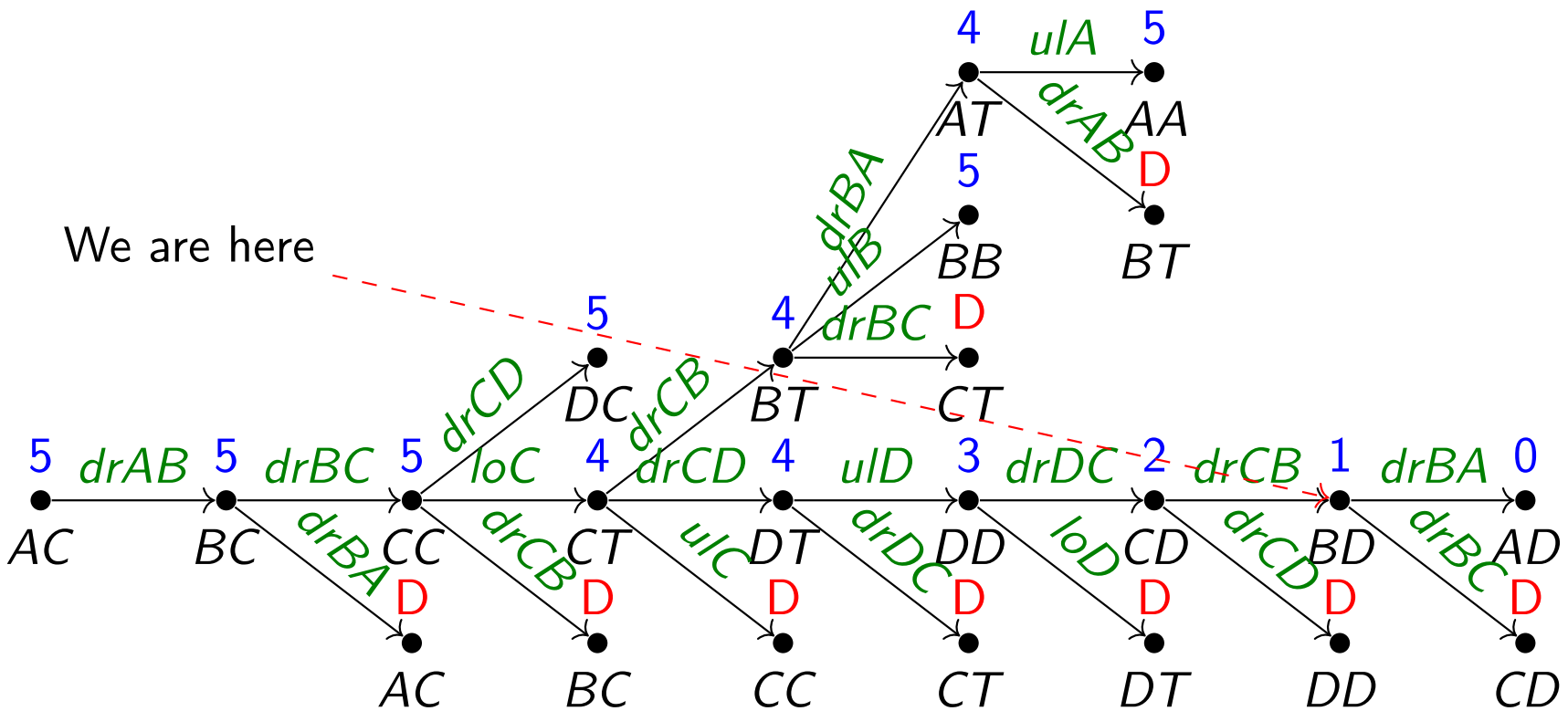
How to Relax During Search: Ignoring Deletes



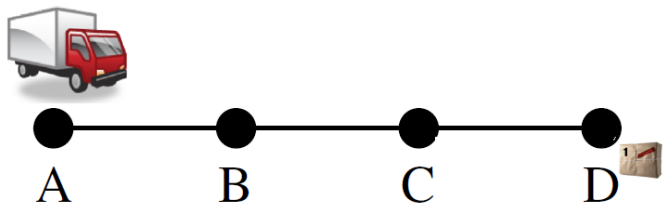
Real problem:

- State s: BD; goal G: AD.
- Actions A: pre, add, del.
- Successors: AD, CD.

Greedy best-first search: (tie-breaking: alphabetic)



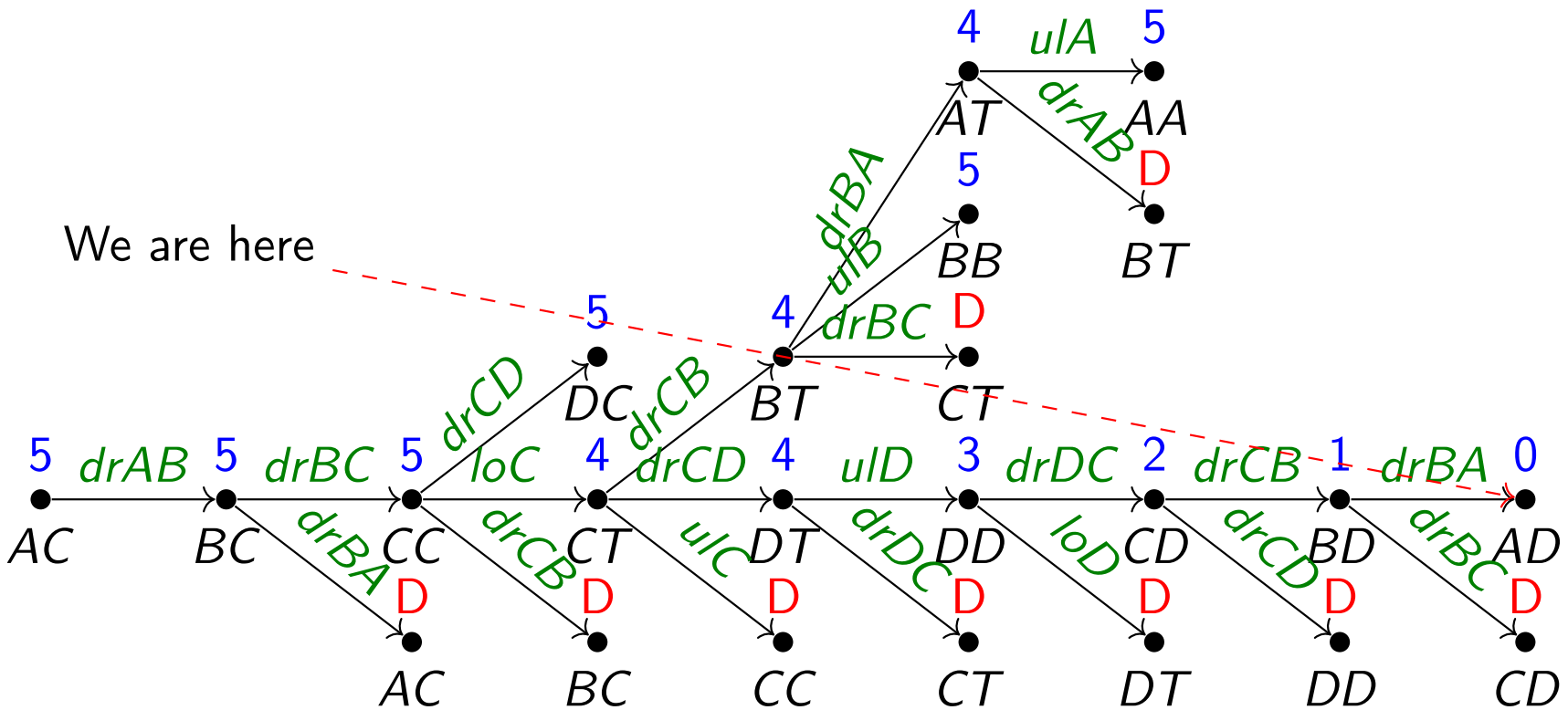
How to Relax During Search: Ignoring Deletes



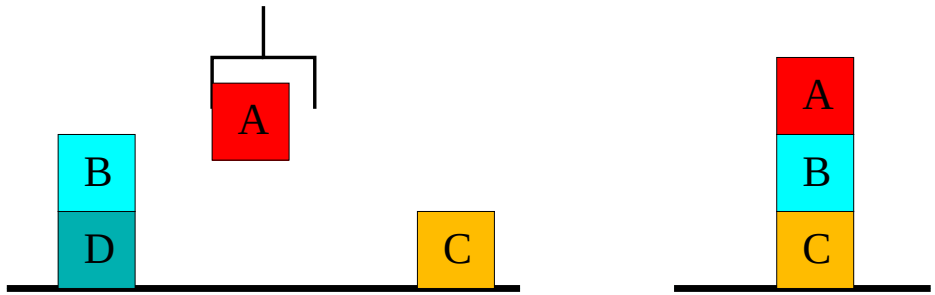
Real problem:

- State s: AD; goal G: AD.
- Actions A: $\mathrm{pre}, \mathrm{add}, \mathrm{del}$.
- Goal state!

Greedy best-first search: (tie-breaking: alphabetic)



h^+ in the Blocksworld



- Initial State
- Goal State

- **Optimal plan**: $\langle \text{putdown}(A), \text{unstack}(B, D), \text{stack}(B, C), \text{pickup}(A), \text{stack}(A, B) \rangle$.
- **Optimal relaxed plan**: $\langle \text{stack}(A, B), \text{unstack}(B, D), \text{stack}(B, C) \rangle$.
- **Observation:** What can we say about the "search space surface" at the initial state here?
- The initial state lies on a local minimum under h^+, together with the successor state s where we stacked A onto B. All direct other neighbors of these two states have a strictly higher h^+ value.

18.5 Conclusion

Summary

- Heuristic search on classical search problems relies on a function h mapping states s to an estimate $h(s)$ of their goal state distance. Such functions h are derived by solving relaxed problems.
- In planning, the relaxed problems are generated and solved automatically. There are four known families of suitable relaxation methods: *abstractions*, *landmarks*, *critical paths*, and *ignoring deletes* (aka delete relaxation).
- The delete relaxation consists in dropping the deletes from STRIPS tasks. A relaxed plan is a plan for such a relaxed task. $h^+(s)$ is the length of an optimal relaxed plan for state s. h^+ is NP-hard to compute.
- h^{FF} approximates h^+ by computing some, not necessarily optimal, relaxed plan. That is done by a forward pass (building a *relaxed planning graph*), followed by a backward pass (*extracting a relaxed plan*).

Topics We Didn't Cover Here

- **Abstractions, Landmarks, Critical-Path Heuristics, Cost Partitions, Compilability between Heuristic Functions, Planning Competitions:**
- **Tractable fragments:** Planning sub-classes that can be solved in polynomial time. Often identified by properties of the "causal graph" and "domain transition graphs".
- **Planning as SAT:** Compile length-k bounded plan existence into satisfiability of a CNF formula φ. Extensive literature on how to obtain small φ, how to schedule different values of k, how to modify the underlying SAT solver.
- **Compilations:** Formal framework for determining whether planning formalism X is (or is not) at least as expressive as planning formalism Y.
- **Admissible pruning/decomposition methods:** Partial-order reduction, symmetry reduction, simulation-based dominance pruning, factored planning, decoupled search.
- **Hand-tailored planning:** Automatic planning is the extreme case where the computer is given no domain knowledge other than "physics". We can instead allow the user to provide search control knowledge, trading off modeling effort against search performance.
- **Numeric planning**, **temporal planning**, **planning under uncertainty** ...

Chapter 19
Searching, Planning, and Acting in the Real World

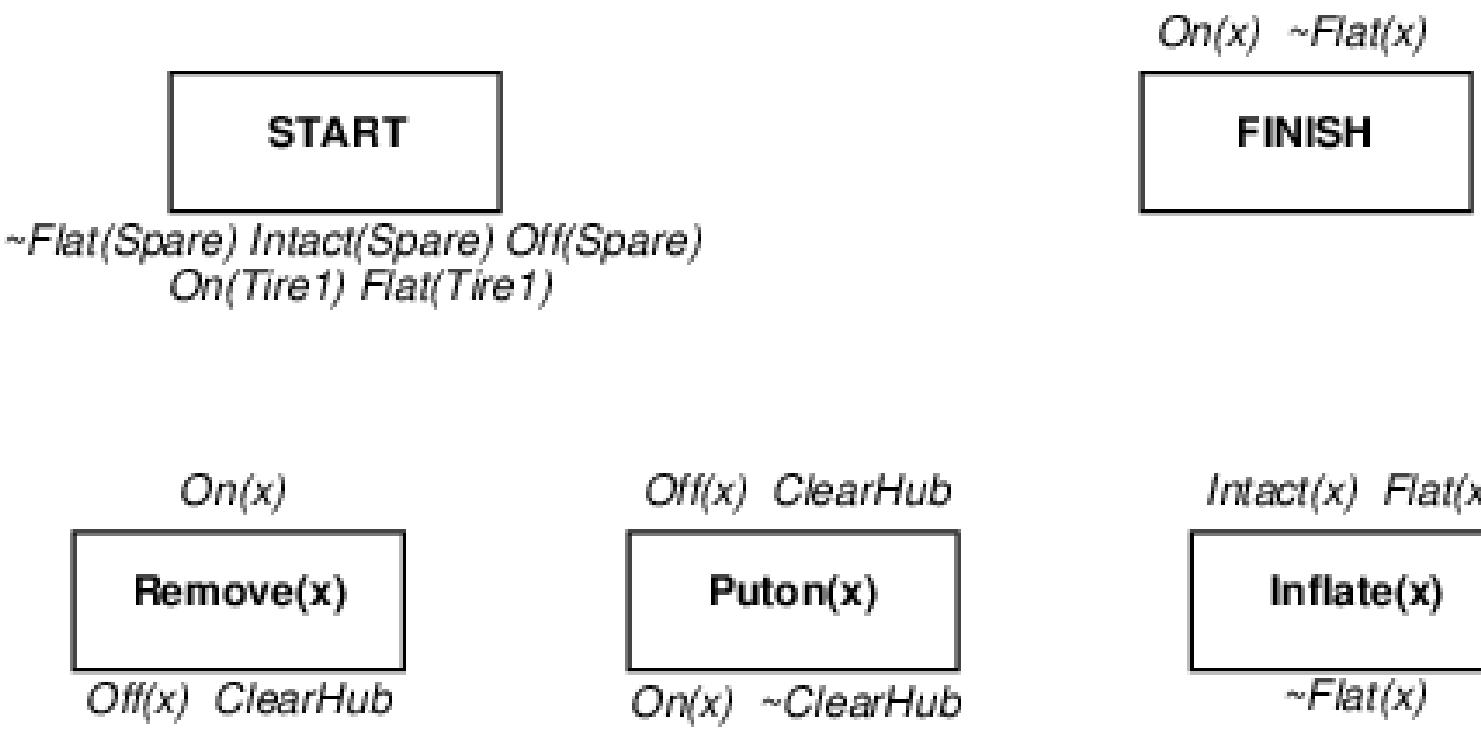
Outline

- **So Far:** we made idealizing/simplifying assumptions:
  The environment is fully observable and deterministic.
- **Outline:** In this chapter we will lift some of them
  - The real world (things go wrong)
  - Agents and Belief States
  - Conditional planning
  - Monitoring and replanning
- **Note:** The considerations in this chapter apply to both search and planning.

19.1 Introduction

The real world

- **Example 1.1.** We have a flat tire – what to do?



Generally: Things go wrong (in the real world)

- **Example 1.2 (Incomplete Information).**
  - Unknown preconditions, e.g., $Intact(Spare)$?
  - Disjunctive effects, e.g., $Inflate(x)$ causes $Inflated(x) \vee SlowHiss(x) \vee Burst(x) \vee BrokenPump \vee \ldots$
- **Example 1.3 (Incorrect Information).**
  - Current state incorrect, e.g., spare NOT intact
  - Missing/incorrect effects in actions.
- **Definition 1.4.** The qualification problem in planning is that we can never finish listing all the required preconditions and possible conditional effects of actions.
- **Root Cause:** The environment is partially observable and/or non-deterministic.
- **Technical Problem:** We cannot know the "current state of the world", but search/planning algorithms are based on this assumption.
- **Idea:** Adapt search/planning algorithms to work with "sets of possible states".

What can we do if things (can) go wrong?

- **One Solution:** Sensorless planning: plans that work regardless of state/outcome.
- **Problem:** Such plans may not exist! (but they often do in practice)
- **Another Solution:** Conditional plans:
  - Plan to obtain information, (observation actions)
  - Subplan for each contingency.
- **Example 1.5 (A conditional Plan).** (AAA $\widehat{=}$ ADAC)
  $[Check(T1), \text{if } Intact(T1) \text{ then } Inflate(T1) \text{ else } CallAAA \text{ fi}]$
- **Problem:** Expensive because it plans for many unlikely cases.
- **Still another Solution:** Execution monitoring/replanning
  - Assume normal states/outcomes, check progress *during execution*, replan if necessary.
- **Problem:** Unanticipated outcomes may lead to failure. (e.g., no AAA card)
- **Observation 1.6.** *We really need a combination; plan for likely/serious eventualities, deal with others when they arise, as they must eventually.*

19.2 The Furniture Coloring Example

The Furniture-Coloring Example: Specification

- **Example 2.1 (Coloring Furniture).**

Paint a chair and a table in matching colors.

- The initial state is:
  - we have two cans of paint of unknown color,
  - the color of the furniture is unknown as well,
  - only the table is in the agent's field of view.
- Actions:
  - remove lid from can
  - paint object with paint from open can.

The Furniture-Coloring Example: PDDL

- **Example 2.2 (Formalization in PDDL).**
  - The PDDL domain file is as expected (actions below)

```
(define (domain furniture—coloring)
  (:predicates (object ?x) (can ?x) (inview ?x) (color ?x ?y))
  ...)
```

The Furniture-Coloring Example: PDDL

- **Example 2.3 (Formalization in PDDL).**
  - The PDDL domain file is as expected (actions below)
  - The PDDL problem file has a "free" variable ?c for the (undetermined) joint color.

```
(define (problem tc-coloring)
  (:domain furniture-objects)
  (:objects table chair c1 c2)
  (:init (object table) (object chair) (can c1) (can c2) (inview table))
  (:goal (color chair ?c) (color table ?c)))
```

The Furniture-Coloring Example: PDDL

- **Example 2.4 (Formalization in PDDL).**
  - The PDDL domain file is as expected (actions below)
  - The PDDL problem file has a "free" variable ?c for the (undetermined) joint color.
  - Two action schemata: *remove can lid to open* and *paint with open can*

```
(:action remove—lid
          :parameters (?x)
          :precondition (can ?x)
          :effect (open can))
(:action paint
          :parameters (?x ?y)
          :precondition (and (object ?x) (can ?y) (color ?y ?c) (open ?y))
          :effect (color ?x ?c))
```

    has a universal variable ?c for the paint action ⇜ we cannot just give paint a color argument in a partially observable environment.
  - **Sensorless Plan**: Open one can, paint chair and table in its color.

The Furniture-Coloring Example: PDDL

- **Example 2.5 (Formalization in PDDL).**
  - The PDDL domain file is as expected (actions below)
  - The PDDL problem file has a "free" variable ?c for the (undetermined) joint color.
  - Two action schemata: *remove can lid to open* and *paint with open can* has a universal variable ?c for the paint action ⇜ we cannot just give paint a color argument in a partially observable environment.
  - **Sensorless Plan**: Open one can, paint chair and table in its color.
  - **Note**: Contingent planning can create better plans, but needs perception
  - Two percept schemata: *color of an object* and *color in a can*

```
(:percept color
          :parameters (?x ?c)
          :precondition (and (object ?x) (inview ?x)))
(:percept can—color
          :parameters (?x ?c)
          :precondition (and (can ?x) (inview ?x) (open ?x)))
```

To perceive the color of an object, it must be in view, a can must also be open.
Note: In a fully observable world, the percepts would not have preconditions.

The Furniture-Coloring Example: PDDL

- **Example 2.6 (Formalization in PDDL).**
  - The PDDL domain file is as expected (actions below)
  - The PDDL problem file has a "free" variable ?c for the (undetermined) joint color.
  - Two action schemata: *remove can lid to open* and *paint with open can* has a universal variable ?c for the paint action ⇜ we cannot just give paint a color argument in a partially observable environment.
  - **Sensorless Plan**: Open one can, paint chair and table in its color.
  - **Note**: Contingent planning can create better plans, but needs perception
  - Two percept schemata: *color of an object* and *color in a can*
  - An action schema: *look at an object* that causes it to come into view.

```
(:action lookat
          :parameters (?x)
          :precond: (and (inview ?y) and (notequal ?x ?y))
          :effect (and (inview ?x) (not (inview ?y))))
```

The Furniture-Coloring Example: PDDL

- **Example 2.7 (Formalization in PDDL).**
  - The PDDL domain file is as expected (actions below)
  - The PDDL problem file has a "free" variable ?c for the (undetermined) joint color.
  - Two action schemata: *remove can lid to open* and *paint with open can* has a universal variable ?c for the paint action $\rightsquigarrow$ we cannot just give paint a color argument in a partially observable environment.
  - **Sensorless Plan**: Open one can, paint chair and table in its color.
  - **Note**: Contingent planning can create better plans, but needs perception
  - Two percept schemata: *color of an object* and *color in a can*
  - An action schema: *look at an object* that causes it to come into view.
  - **Contingent Plan**:
    1. look at furniture to determine color, if same $\rightsquigarrow$ done.
    2. else, look at open and look at paint in cans
    3. if paint in one can is the same as an object, paint the other with this color
    4. else paint both in any color

19.3 Searching/Planning with Non-Deterministic Actions

Conditional Plans

- **Definition 3.1.** Conditional plans extend the possible actions in plans by conditional steps that execute sub plans conditionally whether $K + P \models C$, where $K + P$ is the current knowledge base + the percepts.
- **Definition 3.2.** Conditional plans can contain
  - conditional step: $[\ldots, \text{if } C \text{ then } Plan_A \text{ else } Plan_B \text{ fi}, \ldots]$,
  - while step: $[\ldots, \text{while } C \text{ do } Plan \text{ done}, \ldots]$, and
  - the empty plan $\emptyset$ to make modeling easier.
- **Definition 3.3.** If the possible percepts are limited to determining the current state in a conditional plan, then we speak of a contingency plan.
- **Note:** Need *some plan* for *every possible percept*! Compare to

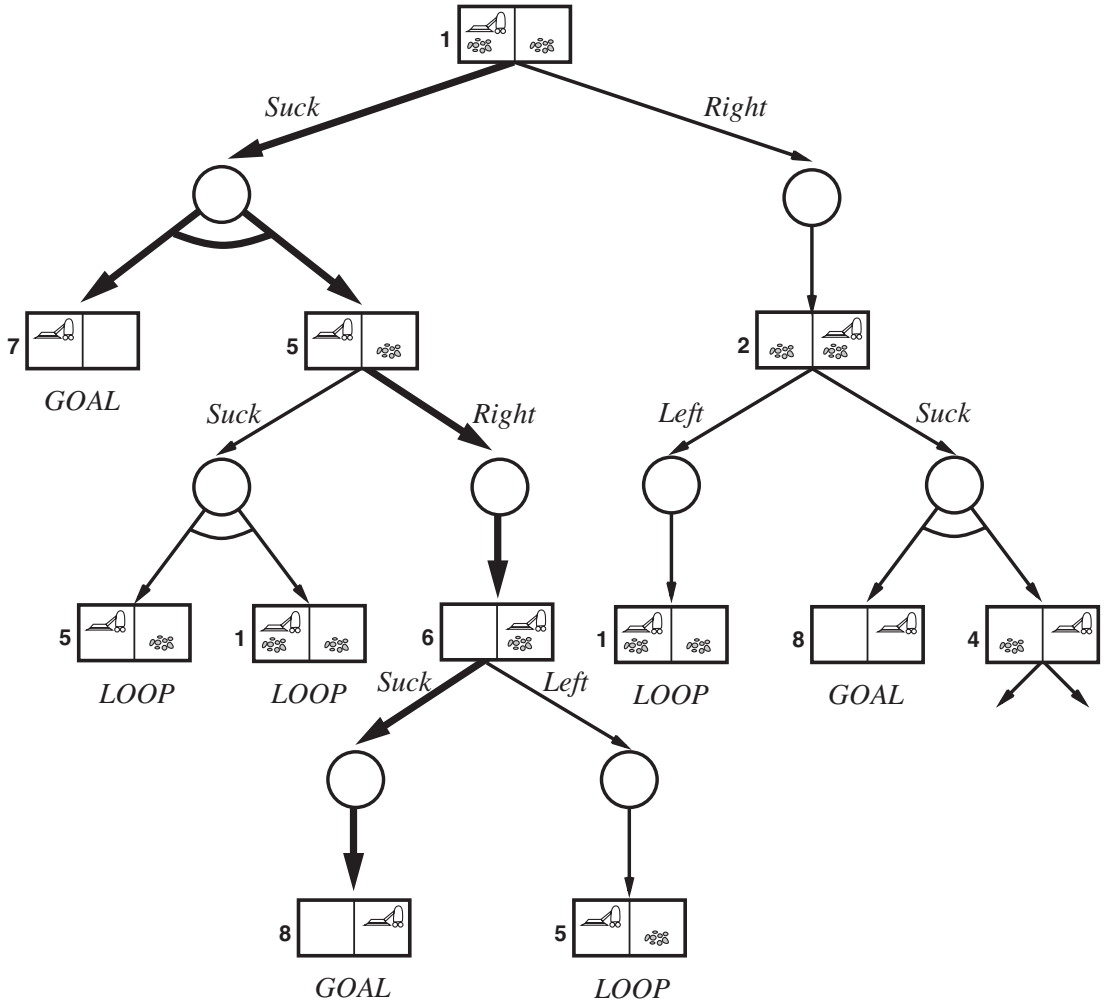
  game playing: *some response* for *every opponent* move.
  backchaining: *some rule* such that *every premise* satisfied.
- **Idea:** Use an AND–OR tree search (very similar to backward chaining algorithm)

Contingency Planning: The Erratic Vacuum Cleaner

- **Example 3.4 (Erratic vacuum world).**

A variant *suck* action:
if square is

- *dirty*: clean the square, sometimes remove dirt in adjacent square.
- *clean*: sometimes deposits dirt on the carpet.



Solution: $[suck, \text{if } State = 5 \text{ then } [right, suck] \text{ else } [] \text{ fi}]$

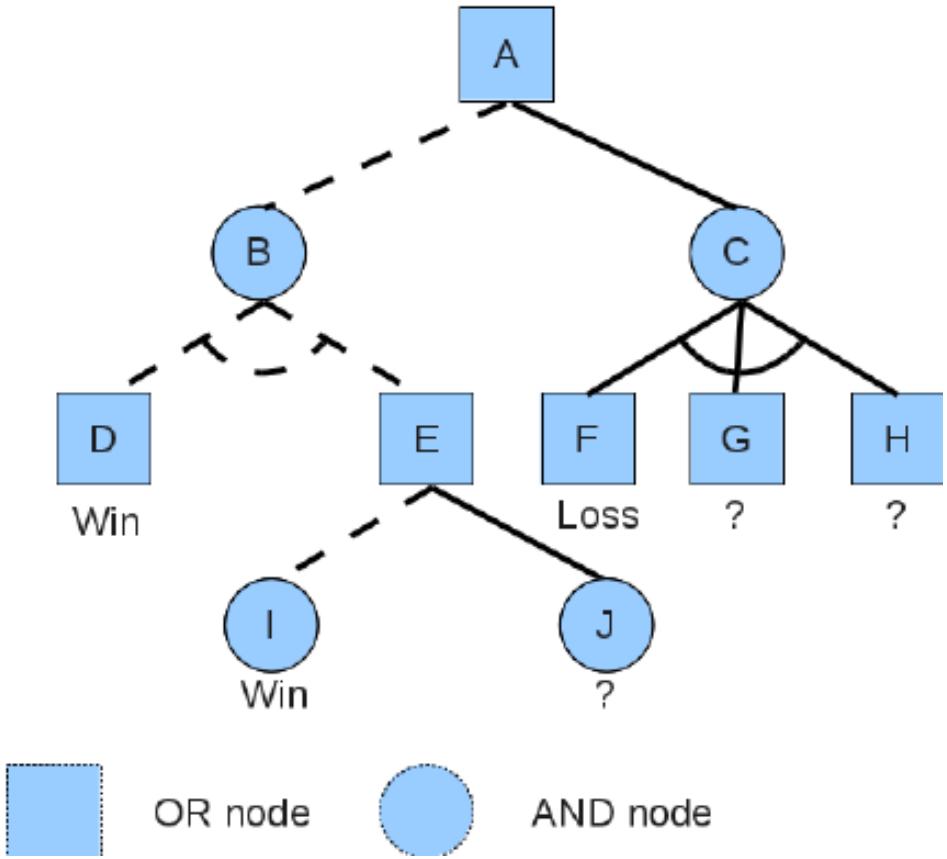
Conditional AND-OR Search (Data Structure)

- **Idea:** Use AND-OR trees as data structures for representing problems (or goals) that can be reduced to to conjunctions and disjunctions of subproblems (or subgoals).
- **Definition 3.5.** An AND-OR graph is a is a graph whose non-terminal nodes are partitioned into AND nodes and OR nodes. A valuation of an AND-OR graph T is an assignment of T or F to the nodes of T. A valuation of the terminal nodes of T can be extended by all nodes recursively: Assign T to an
  - OR node, iff at least one of its children is T.
  - AND node, iff all of its children are T.

  A solution for T is a valuation that assigns T to the initial nodes of T.
- **Idea:** A planning task with non deterministic actions generates a AND-OR graph T. A solution that assigns T to a terminal node, iff it is a goal node. Corresponds to a conditional plan.

Conditional AND-OR Search (Example)

- **Definition 3.6.** An AND-OR tree is a AND-OR graph that is also a tree.
  Notation: AND nodes are written with arcs connecting the child edges.
- **Example 3.7 (An AND-OR-tree).**



Conditional AND-OR Search (Algorithm)

- **Definition 3.8.** AND-OR search is an algorithm for searching AND–OR graphs generated by nondeterministic environments.

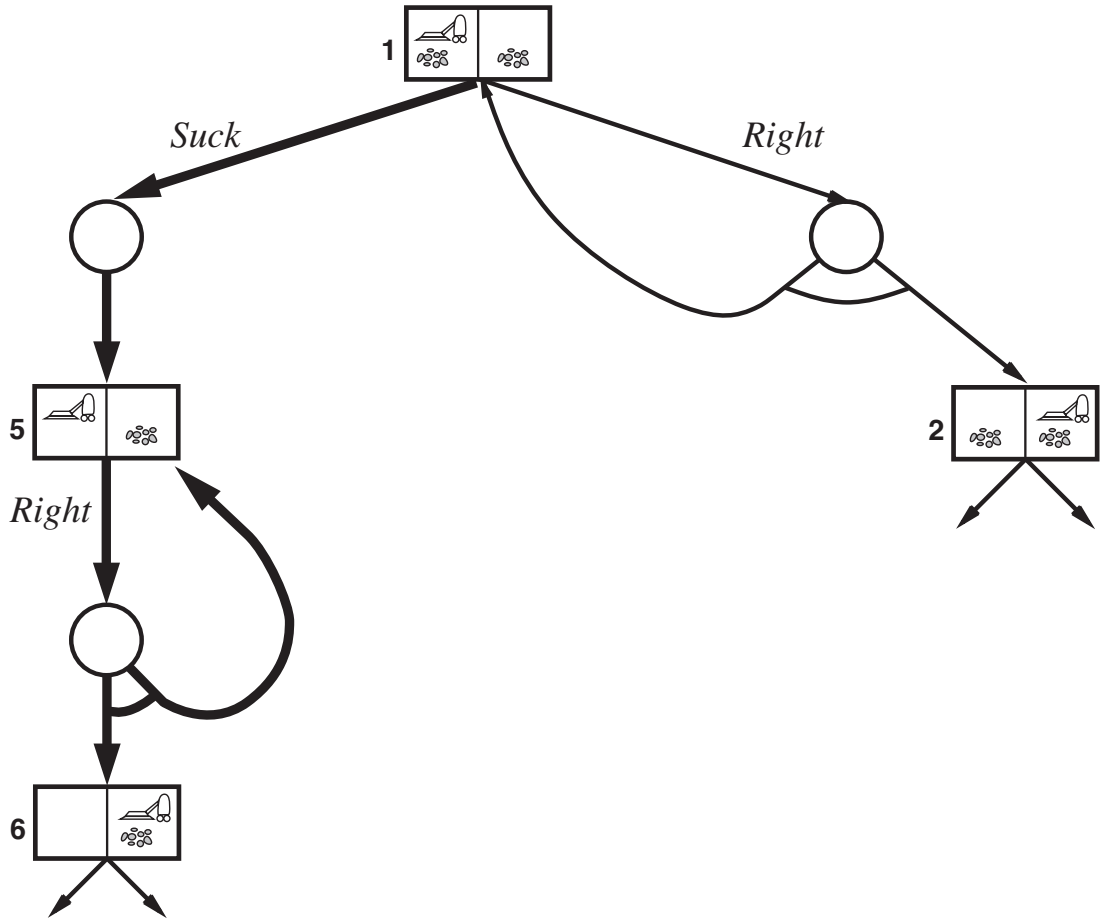
```
function AND/OR–GRAPH–SEARCH(prob) returns a conditional plan, or fail
  OR–SEARCH(prob.INITIAL–STATE, prob, [])
function OR–SEARCH(state,prob,path) returns a conditional plan, or fail
  if prob.GOAL–TEST(state) then return the empty plan
  if state is on path then return fail
  for each action in prob.ACTIONS(state) do
    plan := AND–SEARCH(RESULTS(state,action),prob,[state | path])
    if plan ≠ fail then return [action | plan]
  return fail
function AND–SEARCH(states,prob,path) returns a conditional plan, or fail
  for each s_i in states do
    p_i := OR–SEARCH(s_i,prob,path)
    if p_i = fail then return fail
    return [if s_1 then p_1 else if s_2 then p_2 else ... if s_{n−1} then p_{n−1} else p_n]
```

- **Cycle Handling:** If a state has been seen before $\rightsquigarrow$ **fail**
  - **fail** does not mean *there is no solution*, but
  - *if there is a non-cyclic solution, then it is reachable by an earlier incarnation*!

The Slippery Vacuum Cleaner (try, try, try, . . . try again)

- **Example 3.9 (Slippery Vacuum World).**

Moving sometimes fails
$\rightsquigarrow$ AND-OR graph



Two possible solutions (depending on what our plan language allows)

- $[L_1 :$ *left*, if *AtR* then L_1 else [if *CleanL* then $\emptyset$ else *suck* fi] fi] or
- [while *AtR* do [*left*] done, if *CleanL* then $\emptyset$ else *suck* fi]

- We have an infinite loop but plan eventually works unless action always fails.

19.4 Agent Architectures based on Belief States

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!
- **Idea:** Just keep track of all the possible states it could be in.
- **Definition 4.2.** A model-based agent has a world model consisting of
  - a belief state that has information about the possible states the world may be in, and
  - a sensor model that updates the belief state based on sensor information
  - a transition model that updates the belief state based on actions.

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!
- **Idea:** Just keep track of all the possible states it could be in.
- **Definition 4.3.** A model-based agent has a world model consisting of
  - a belief state that has information about the possible states the world may be in, and
  - a sensor model that updates the belief state based on sensor information
  - a transition model that updates the belief state based on actions.
- **Idea:** The agent environment determines what the world model can be.

World Models for Uncertainty

- **Problem:** We do not know with certainty what state the world is in!
- **Idea:** Just keep track of all the possible states it could be in.
- **Definition 4.4.** A model-based agent has a world model consisting of
  - a belief state that has information about the possible states the world may be in, and
  - a sensor model that updates the belief state based on sensor information
  - a transition model that updates the belief state based on actions.
- **Idea:** The agent environment determines what the world model can be.
- In a fully observable, deterministic environment,
  - we can observe the initial state and subsequent states are given by the actions alone.
  - thus the belief state is a singleton (we call its member the world state) and the transition model is a function from states and actions to states: a transition function.

World Models by Agent Type in AI-1

- **Search-based Agents:** In a fully observable, deterministic environment
  - goal-based agent with world state $\widehat{=}$ "current state"
  - no inference. (goal $\widehat{=}$ goal state from search problem)
- **CSP-based Agents:** In a fully observable, deterministic environment
  - goal-based agent withworld state $\widehat{=}$ constraint network,
  - inference $\widehat{=}$ constraint propagation. (goal $\widehat{=}$ satisfying assignment)
- **Logic-based Agents:** In a fully observable, deterministic environment
  - model-based agent with world state $\widehat{=}$ logical formula
  - inference $\widehat{=}$ e.g. DPLL or resolution.
- **Planning Agents:** In a fully observable, deterministic, environment
  - goal-based agent with world state $\widehat{=}$ PL0, transition model $\widehat{=}$ STRIPS,
  - inference $\widehat{=}$ state/plan space search. (goal: complete plan/execution)

World Models for Complex Environments

- In a fully observable, but stochastic environment,
  - the belief state must deal with a set of possible states.
  - $\rightsquigarrow$ generalize the transition function to a transition relation.

World Models for Complex Environments

- In a fully observable, but stochastic environment,
  - the belief state must deal with a set of possible states.
  - $\rightsquigarrow$ generalize the transition function to a transition relation.
- **Note:** This even applies to online problem solving, where we can just perceive the state. (e.g. when we want to optimize utility)

World Models for Complex Environments

- In a fully observable, but stochastic environment,
  - the belief state must deal with a set of possible states.
  - $\rightsquigarrow$ generalize the transition function to a transition relation.
- **Note:** This even applies to online problem solving, where we can just perceive the state. (e.g. when we want to optimize utility)
- In a deterministic, but partially observable environment,
  - the belief state must deal with a set of possible states.
  - we can use transition functions.
  - We need a sensor model, which predicts the influence of percepts on the belief state – during update.

World Models for Complex Environments

- In a fully observable, but stochastic environment,
  - the belief state must deal with a set of possible states.
  - $\rightsquigarrow$ generalize the transition function to a transition relation.
- **Note:** This even applies to online problem solving, where we can just perceive the state. (e.g. when we want to optimize utility)
- In a deterministic, but partially observable environment,
  - the belief state must deal with a set of possible states.
  - we can use transition functions.
  - We need a sensor model, which predicts the influence of percepts on the belief state – during update.
- In a stochastic, partially observable environment,
  - mix the ideas from the last two. (sensor model + transition relation)

Preview: New World Models (Belief) $\rightsquigarrow$ new Agent Types

- **Probabilistic Agents:** In a partially observable environment
  - belief state $\widehat{=}$ Bayesian networks,
  - inference $\widehat{=}$ probabilistic inference.

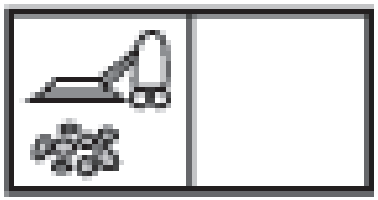
Preview: New World Models (Belief) $\rightsquigarrow$ new Agent Types

- **Probabilistic Agents:** In a partially observable environment
  - belief state $\widehat{=}$ Bayesian networks,
  - inference $\widehat{=}$ probabilistic inference.
- **Decision-Theoretic Agents:** In a partially observable, stochastic environment
  - belief state + transition model $\widehat{=}$ decision networks,
  - inference $\widehat{=}$ maximizing expected utility.
- We will study them in detail in the coming semester.

19.5 Searching/Planning without Observations

Conformant/Sensorless Planning

- **Definition 5.1.** Conformant or sensorless planning tries to find plans that work without any sensing. (not even the initial state)



- **Example 5.2 (Sensorless Vacuum Cleaner World).**

States	integer dirt and robot locations
Actions	*left*, *right*, *suck*, *noOp*
Goal states	*notdirty*?

- **Observation 5.3.** *In a sensorless world we do not know the initial state.(or any state after)*
- **Observation 5.4.** *Sensorless planning must search in the space of belief states (sets of possible actual states).*
- **Example 5.5 (Searching the Belief State Space).**
  - Start in $\{1,2,3,4,5,6,7,8\}$
  - Solution: $[right, suck, left, suck]$

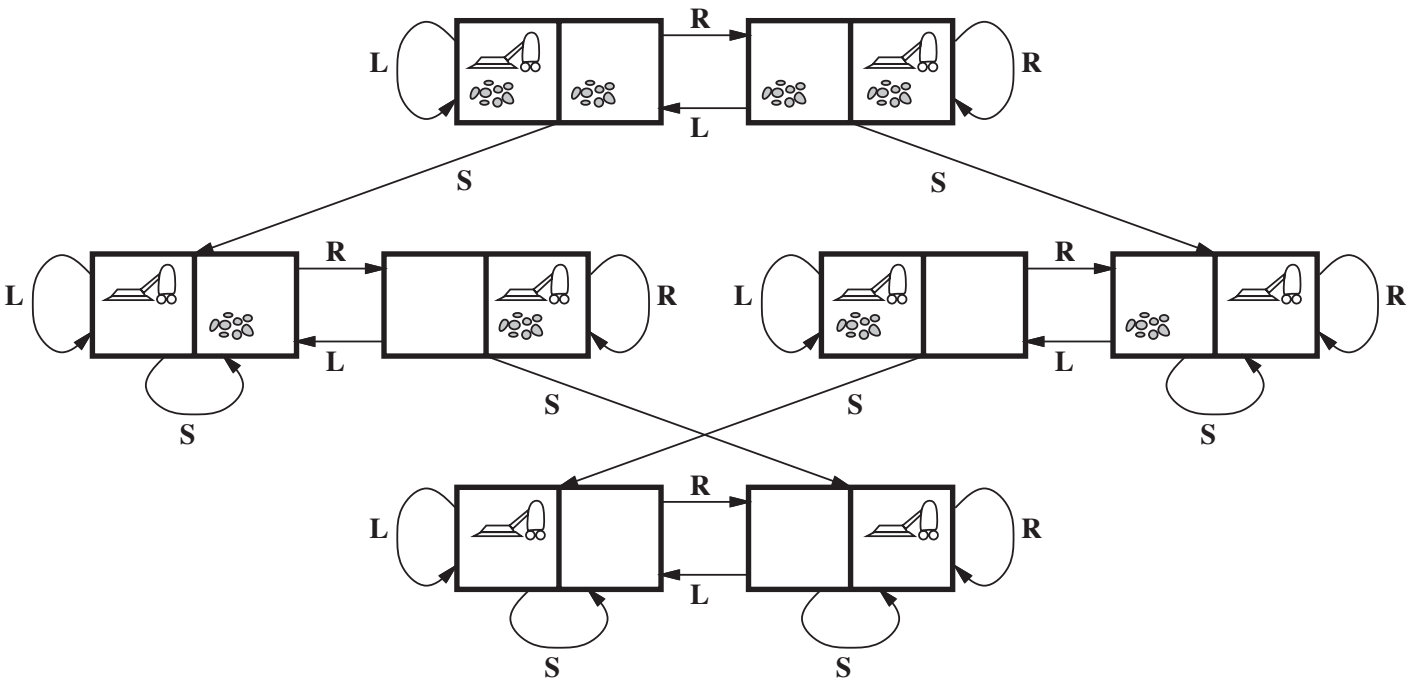
$$\begin{aligned} right &\to \{2,4,6,8\} \\ suck &\to \{4,8\} \\ left &\to \{3,7\} \\ suck &\to \{7\} \end{aligned}$$

Search in the Belief State Space: Let's Do the Math

- **Recap:** We describe an search problem $\Pi := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ via its states $\mathcal{S}$, actions $\mathcal{A}$, and transition model $\mathcal{T}: \mathcal{A} \times \mathcal{S} \to \mathcal{P}(\mathcal{A})$, goal states $\mathcal{G}$, and initial state $\mathcal{I}$.
- **Problem:** What is the corresponding sensorless problem?
- **Let' think:** Let $\Pi := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ be a (physical) problem
  - States $\mathcal{S}^b$: The belief states are the $2^{|\mathcal{S}|}$ subsets of $\mathcal{S}$.
  - The initial state $\mathcal{I}^b$ is just $\mathcal{S}$ (no information)
  - Goal states $\mathcal{G}^b := \{S \in \mathcal{S}^b \mid S \subseteq \mathcal{G}\}$ (all possible states must be physical goal states)
  - Actions $\mathcal{A}^b$: we just take $\mathcal{A}$. (that's the point!)
  - Transition model $\mathcal{T}^b: \mathcal{A}^b \times \mathcal{S}^b \to \mathcal{P}(\mathcal{A}^b)$: i.e. what is $\mathcal{T}^b(a, S)$ for $a \in \mathcal{A}$ and $S \subseteq \mathcal{S}$? This is slightly tricky as a need not be applicable to all $s \in S$.
    1. if actions are harmless to the environment, take $\mathcal{T}^b(a, S) := \bigcup_{s \in S} \mathcal{T}(a, s)$.
    2. if not, better take $\mathcal{T}^b(a, S) := \bigcap_{s \in S} \mathcal{T}(a, s)$. (the safe bet)
- **Observation 5.6.** *In belief-state space the problem is always fully observable!*

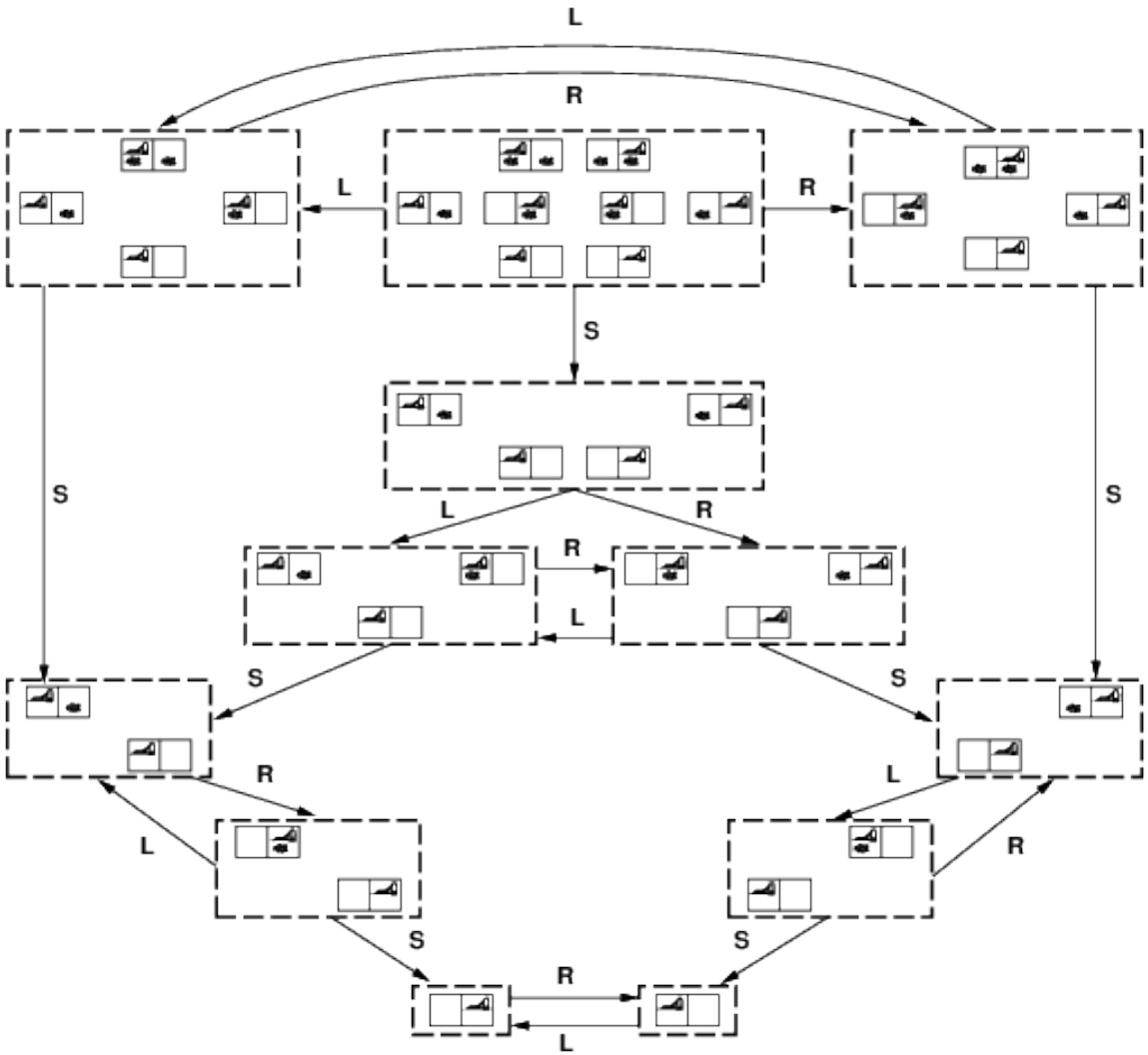
State Space vs. Belief State Space

- **Example 5.7 (State/Belief State Space in the Vacuum World).** In the vacuum world all actions are always applicable (1./2. equal)



State Space vs. Belief State Space

- **Example 5.8 (State/Belief State Space in the Vacuum World).** In the vacuum world all actions are always applicable (1./2. equal)



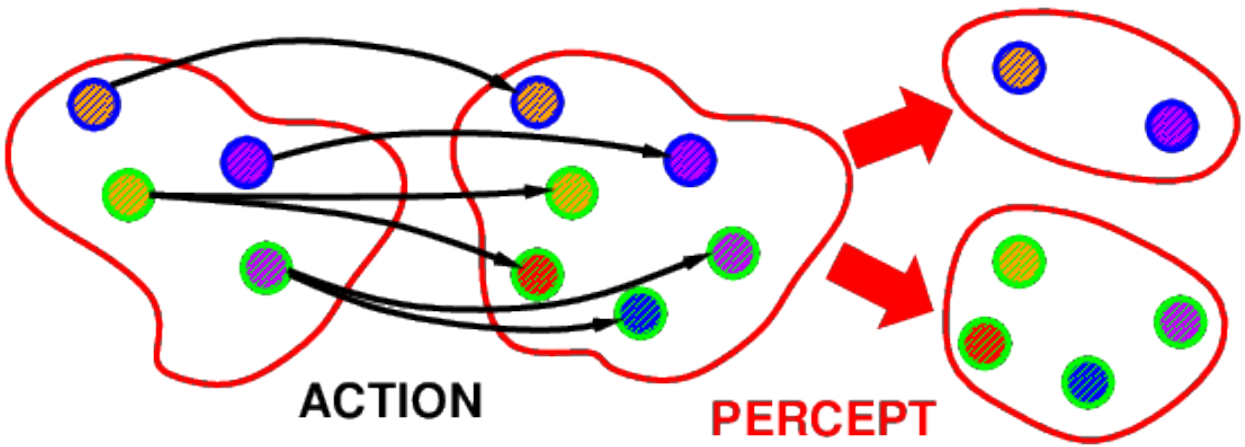
Evaluating Conformant Planning

- **Upshot:** We can build belief-space problem formulations automatically,
  - but they are exponentially bigger in theory, in practice they are often similar;
  - e.g. 12 reachable belief states out of $2^8 = 256$ for vacuum example.
- **Problem:** Belief states are HUGE; e.g. initial belief state for the 10×10 vacuum world contains $100 \cdot 2^{100} \approx 10^{32}$ physical states
- **Idea:** Use planning techniques: compact descriptions for
  - belief states; e.g. *all* for initial state or *not leftmost column* after *left*.
  - actions as belief state to belief state operations.
- **This actually works:** Therefore we talk about conformant planning!

19.6 Searching/Planning with Observation

Conditional planning (Motivation)

- **Note:** So far, we have never used the agent's sensors.
  - In ??, since the environment was observable and deterministic we could just use offline planning.
  - In ?? because we chose to.
- **Note:** If the world is nondeterministic or partially observable then percepts usually provide information, i.e., split up the belief state



- **Idea:** This can systematically be used in search/planning via belief-state search, but we need to rethink/specialize the Transition model.

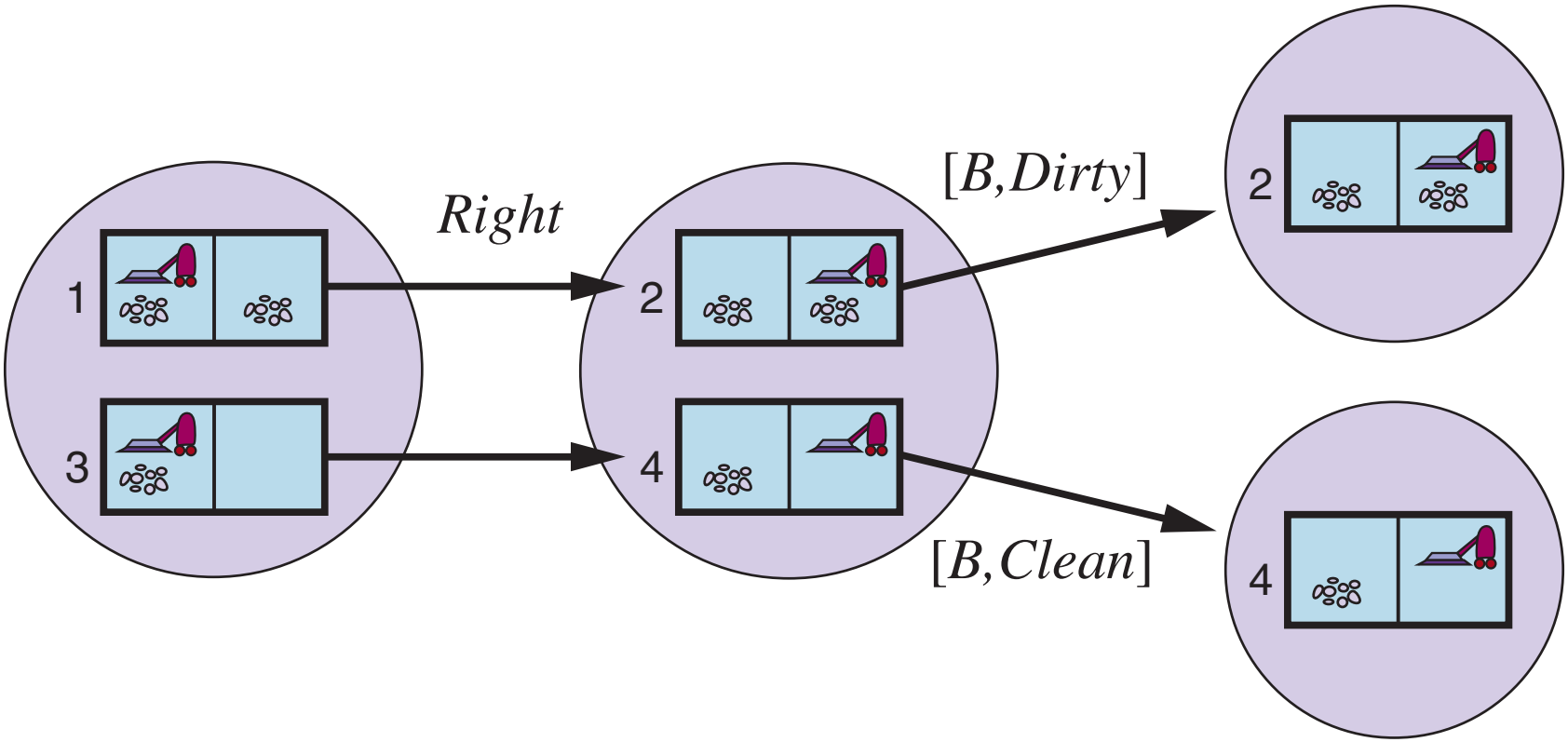
A Transition Model for Belief-State Search

- We extend the ideas from slide 659 to include partial observability.
- **Definition 6.1.** Given a (physical) search problem $\Pi := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$, we define the belief state search problem induced by Π to be $\langle \mathcal{P}(\mathcal{S}), \mathcal{A}, \mathcal{T}^b, \mathcal{S}, \{S \in \mathcal{S}^b \mid S \subseteq \mathcal{G}\} \rangle$, where the transition model $\mathcal{T}^b$ is constructed in three stages:
  - The prediction stage: given a belief state b and an action a we define $\widehat{b} := \mathrm{PRED}(b, a)$ for some function $\mathrm{PRED}\colon \mathcal{P}(\mathcal{S}) \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$.
  - The observation prediction stage determines the set of possible percepts that could be observed in the predicted belief state: $\mathrm{PossPERC}(\widehat{b}) = \{\mathrm{PERC}(s) \mid s \in \widehat{b}\}$.
  - The update stage determines, for each possible percept, the resulting belief state: $\mathrm{UPDATE}(\widehat{b}, o) := \{s \mid o = \mathrm{PERC}(s) \text{ and } s \in \widehat{b}\}$

  The functions PRED and PERC are the main parameters of this model. We define $\mathrm{RESULT}(b, a) := \{\mathrm{UPDATE}(\mathrm{PRED}(b, a), o) \mid \mathrm{PossPERC}(\mathrm{PRED}(b, a))\}$
- **Observation 6.2.** *We always have* $\mathrm{UPDATE}(\widehat{b}, o) \subseteq \widehat{b}$.
- **Observation 6.3.** *If sensing is deterministic, belief states for different possible percepts are disjoint, forming a partition of the original predicted belief state.*

Example: Local Sensing Vacuum Worlds

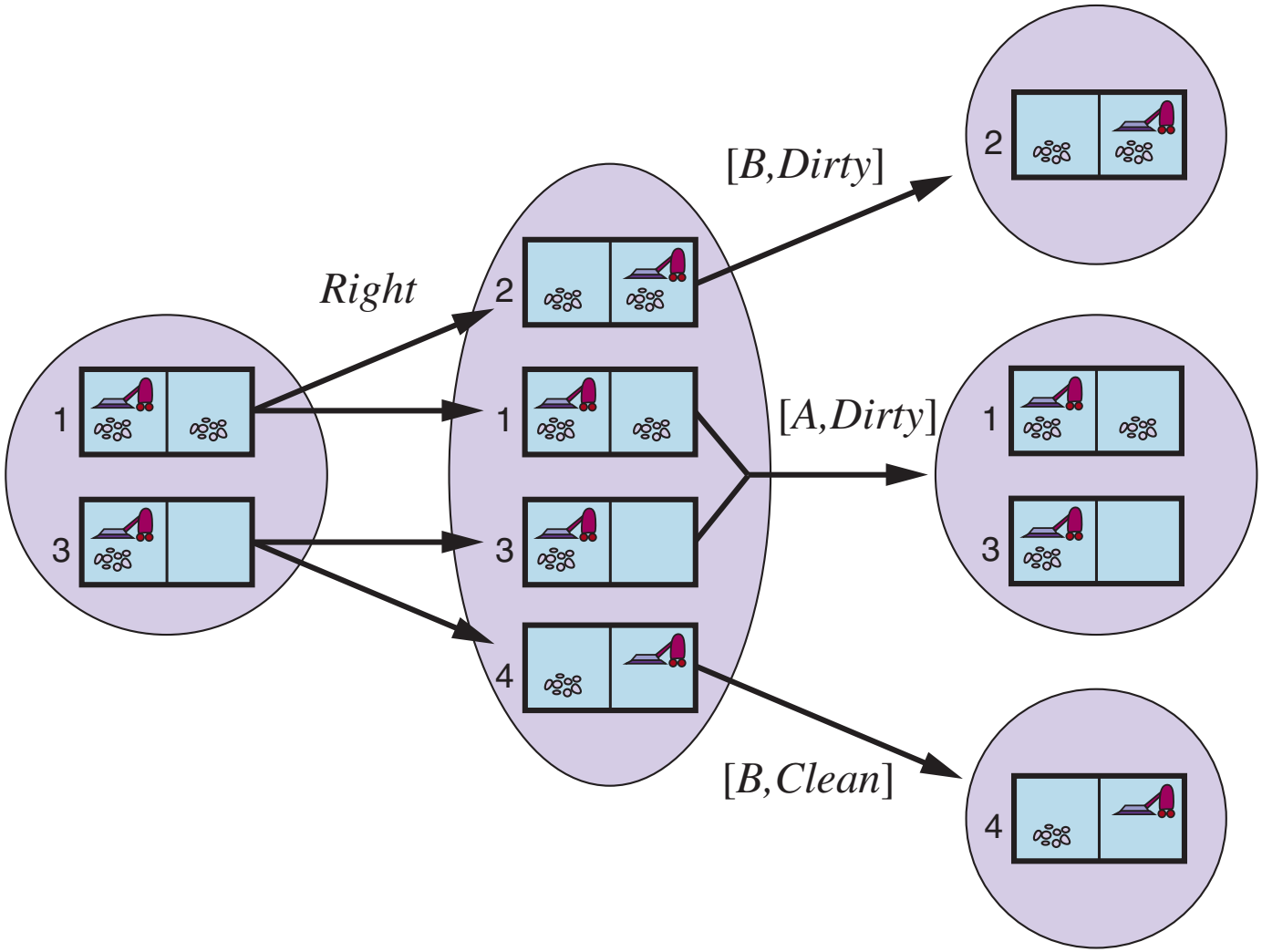
- **Example 6.4 (Transitions in the Vacuum World).** Deterministic World:



The action *Right* is deterministic, sensing disambiguates to singletons

Example: Local Sensing Vacuum Worlds

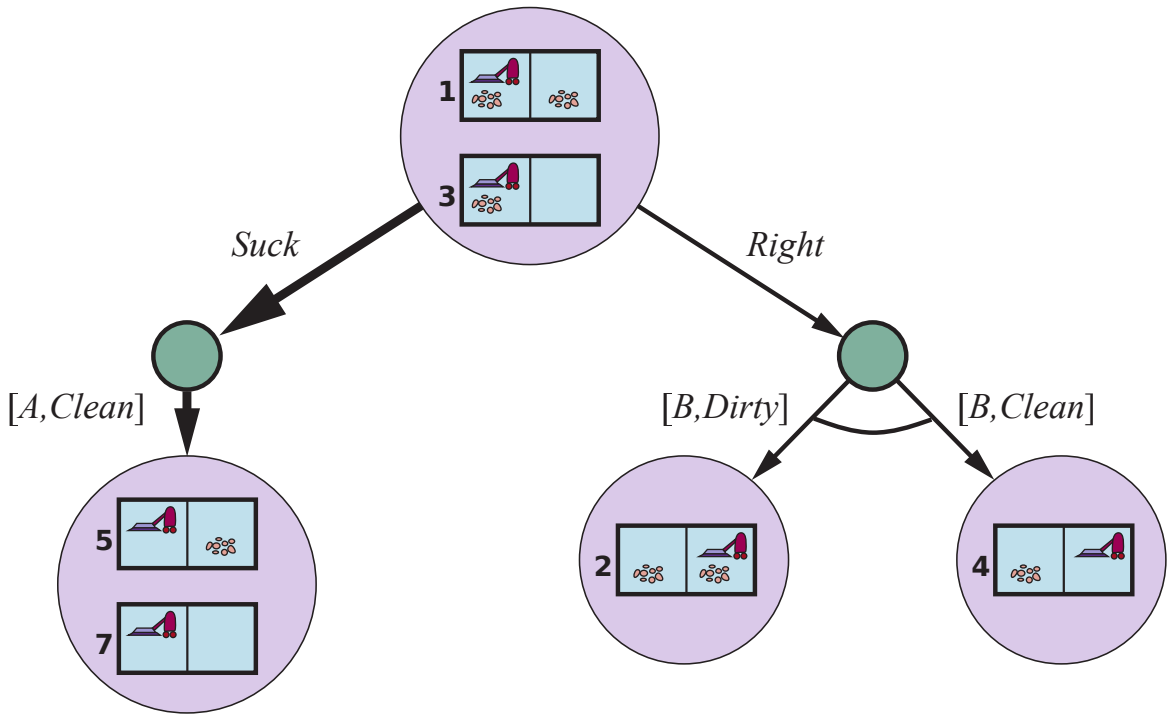
- **Example 6.5 (Transitions in the Vacuum World).** Slippery World:



The action *Right* is non-deterministic, sensing disambiguates somewhat

Belief-State Search with Percepts

- **Observation:** The belief-state transition model induces an AND-OR graph.
- **Idea:** Use AND-OR search in non deterministic environments.
- **Example 6.6.** AND-OR graph for initial percept $[A, Dirty]$.



Solution: $[Suck, Right, \text{if } Bstate = \{6\} \text{ then } Suck \text{ else } [] \text{ fi}]$

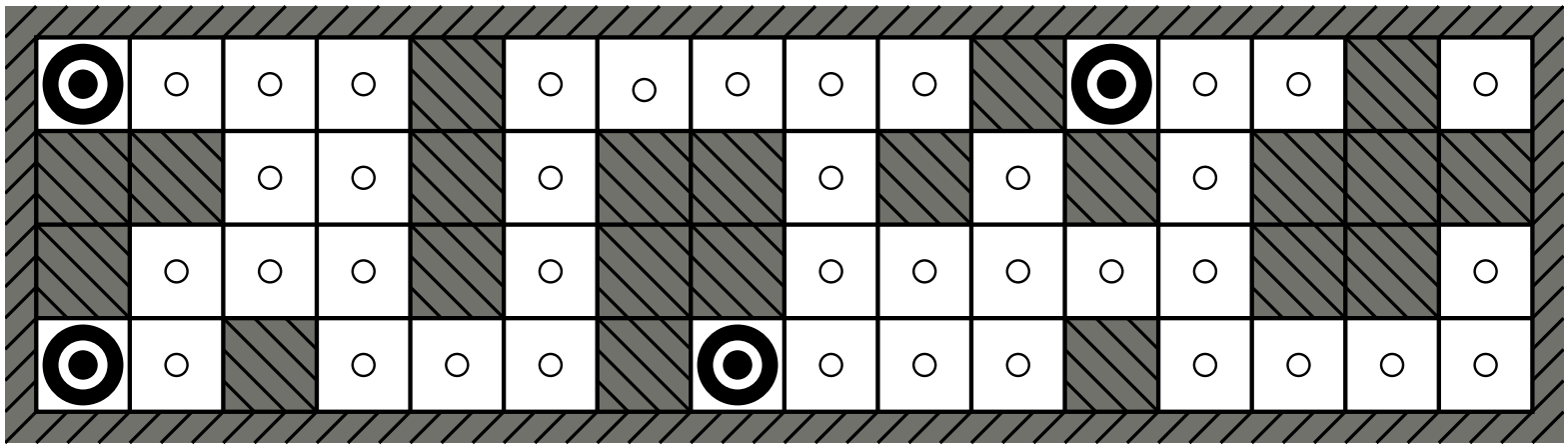
- **Note:** Belief-state-problem $\rightsquigarrow$ conditional step tests on belief-state percept (plan would not be executable in a partially observable environment otherwise)

Example: Agent Localization

- **Example 6.7.** An agent inhabits a maze of which it has an accurate map. It has four sensors that can (reliably) detect walls. The *Move* action is non-deterministic, moving the agent randomly into one of the adjacent squares.
  1. Initial belief state $\leadsto \widehat{b}_1$ all possible locations.

Example: Agent Localization

- **Example 6.8.** An agent inhabits a maze of which it has an accurate map. It has four sensors that can (reliably) detect walls. The *Move* action is non-deterministic, moving the agent randomly into one of the adjacent squares.
  1. Initial belief state $\rightsquigarrow \widehat{b}_1$ all possible locations.
  2. Initial percept: *NWS* (walls north, west, and south) $\rightsquigarrow \widehat{b}_2 = \text{UPDATE}(\widehat{b}_1, \mathit{NWS})$

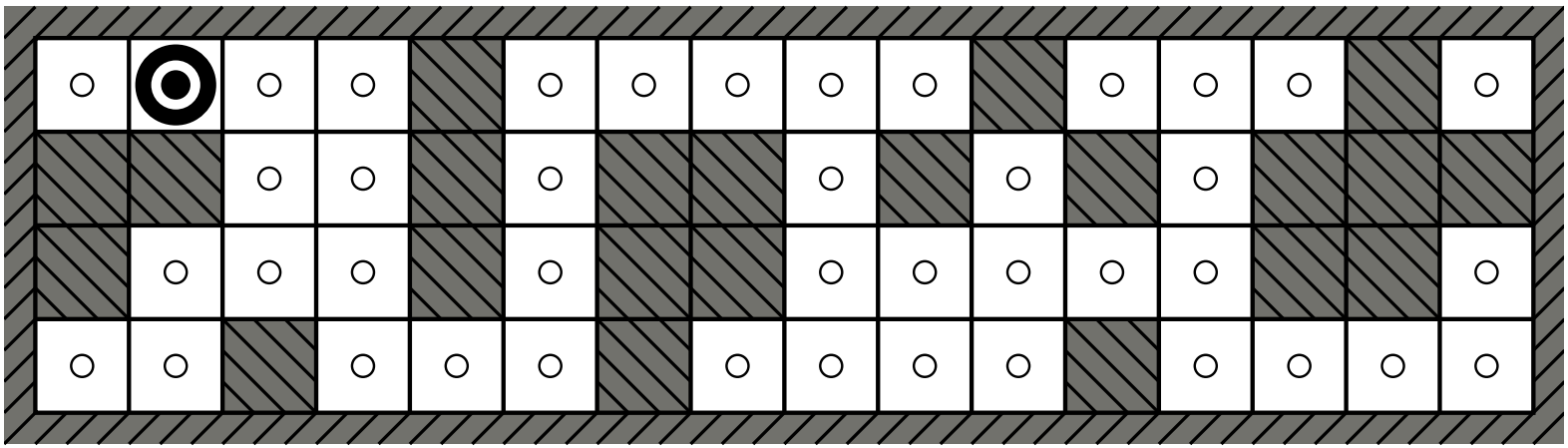


Example: Agent Localization

- **Example 6.9.** An agent inhabits a maze of which it has an accurate map. It has four sensors that can (reliably) detect walls. The *Move* action is non-deterministic, moving the agent randomly into one of the adjacent squares.
  1. Initial belief state $\leadsto \widehat{b}_1$ all possible locations.
  2. Initial percept: *NWS* (walls north, west, and south) $\leadsto \widehat{b}_2 = \mathrm{UPDATE}(\widehat{b}_1, \mathit{NWS})$
  3. Agent executes *Move* $\leadsto \widehat{b}_3 = \mathrm{PRED}(\widehat{b}_2, \mathit{Move}) =$ *one step away from these*.

Example: Agent Localization

- **Example 6.10.** An agent inhabits a maze of which it has an accurate map. It has four sensors that can (reliably) detect walls. The *Move* action is non-deterministic, moving the agent randomly into one of the adjacent squares.
  1. Initial belief state $\rightsquigarrow \widehat{b}_1$ all possible locations.
  2. Initial percept: *NWS* (walls north, west, and south) $\rightsquigarrow \widehat{b}_2 = \mathrm{UPDATE}(\widehat{b}_1, \mathit{NWS})$
  3. Agent executes *Move* $\rightsquigarrow \widehat{b}_3 = \mathrm{PRED}(\widehat{b}_2, \mathit{Move}) =$ *one step away from these*.
  4. Next percept: *NS* $\rightsquigarrow \widehat{b}_4 = \mathrm{UPDATE}(\widehat{b}_3, \mathit{NS})$



Example: Agent Localization

- **Example 6.11.** An agent inhabits a maze of which it has an accurate map. It has four sensors that can (reliably) detect walls. The *Move* action is non-deterministic, moving the agent randomly into one of the adjacent squares.
  1. Initial belief state $\leadsto \widehat{b}_1$ all possible locations.
  2. Initial percept: *NWS* (walls north, west, and south) $\leadsto \widehat{b}_2 = \mathrm{UPDATE}(\widehat{b}_1, \mathit{NWS})$
  3. Agent executes *Move* $\leadsto \widehat{b}_3 = \mathrm{PRED}(\widehat{b}_2, \mathit{Move}) =$ *one step away from these*.
  4. Next percept: *NS* $\leadsto \widehat{b}_4 = \mathrm{UPDATE}(\widehat{b}_3, \mathit{NS})$

  All in all, $\widehat{b}_4 = \mathrm{UPDATE}(\mathrm{PRED}(\mathrm{UPDATE}(\widehat{b}_1, \mathit{NWS}), \mathit{Move}), \mathit{NS})$ localizes the agent.
- **Observation:** PRED enlarges the belief state, while UPDATE shrinks it again.

Contingent Planning

- **Definition 6.12.** The generation of plan with conditional branching based on percepts is called contingent planning, solutions are called contingent plans.
- Appropriate for partially observable or non-deterministic environments.
- **Example 6.13.** Continuing **??**.
  One of the possible contingent plan is

```
((lookat table) (lookat chair)
  (if (and (color table c) (color chair c)) (noop)
    ((removelid c1) (lookat c1) (removelid c2) (lookat c2)
      (if (and (color table c) (color can c)) ((paint chair can))
        (if (and (color chair c) (color can c)) ((paint table can))
          ((paint chair c1) (paint table c1)))))))
```

- **Note:** Variables in this plan are existential; e.g. in
  - line 2: If there is come joint color c of the table and chair $\rightsquigarrow$ done.
  - line 4/5: Condition can be satisfied by $[c_1/can]$ or $[c_2/can]$ $\rightsquigarrow$ instantiate accordingly.
- **Definition 6.14.** During plan execution the agent maintains the belief state b, chooses the branch depending on whether $b \models c$ for the condition c.
- **Note:** The planner must make sure $b \models c$ can always be decided.

Contingent Planning: Calculating the Belief State

- **Problem:** How do we compute the belief state?
- **Recall:** Given a belief state b, the new belief state $\widehat{b}$ is computed based on prediction with the action a and the refinement with the percept p.
- **Here:**
  Given an action a and percepts $p = p_1 \wedge \ldots \wedge p_n$, we have
  - $\widehat{b} = b \backslash \mathrm{del}_a \cup \mathrm{add}_a$ (as for the sensorless agent)
  - If $n = 1$ and (:percept p_1 :precondition c) is the only percept axiom, also add p and c to $\widehat{b}$. (add c as otherwise p impossible)
  - If $n > 1$ and (:percept p_i :precondition c_i) are the percept axioms, also add p and $c_1 \vee \ldots \vee c_n$ to $\widehat{b}$. (belief state no longer conjunction of literals ☹)
- **Idea:** Given such a mechanism for generating (exact or approximate) updated belief states, we can generate contingent plans with an extension of AND-OR search over belief states.
- **Extension:** This also works for non-deterministic actions: we extend the representation of effects to disjunctions.

AI-1 Survey on ALeA

- Online survey evaluating ALeA until 28.02.25 24:00 (Feb last)

AI-1 Survey on ALeA

- Online survey evaluating ALeA until 28.02.25 24:00 (Feb last)
- Works on all common devices (mobile phone, notebook, etc.)
- Is in English; takes about 10 - 20 min
  depending on proficiency in english and using ALeA

AI-1 Survey on ALeA

- Online survey evaluating ALeA until 28.02.25 24:00 (Feb last)
- Works on all common devices (mobile phone, notebook, etc.)
- Is in English; takes about 10 - 20 min
  depending on proficiency in english and using ALeA
- Questions about how ALeA is used, what it is like usig ALeA, and questions about demography

AI-1 Survey on ALeA

- Online survey evaluating ALeA until 28.02.25 24:00 (Feb last)
- Works on all common devices (mobile phone, notebook, etc.)
- Is in English; takes about 10 - 20 min
  depending on proficiency in english and using ALeA
- Questions about how ALeA is used, what it is like usig ALeA, and questions about demography
- Token is generated at the end of the survey (SAVE THIS CODE!)
  - Completed survey count as a successfull prepquiz in AI1!
  - Look for Quiz 15 in the usual place (single question)
  - just submit the token to get full points
  - The token can also be used to exercise the rights of the GDPR.

AI-1 Survey on ALeA

- Online survey evaluating ALeA until 28.02.25 24:00 (Feb last)
- Works on all common devices (mobile phone, notebook, etc.)
- Is in English; takes about 10 - 20 min
  depending on proficiency in english and using ALeA
- Questions about how ALeA is used, what it is like usig ALeA, and questions about demography
- Token is generated at the end of the survey (SAVE THIS CODE!)
  - Completed survey count as a successfull prepquiz in AI1!
  - Look for Quiz 15 in the usual place (single question)
  - just submit the token to get full points
  - The token can also be used to exercise the rights of the GDPR.
- Survey has no timelimit and is free, anonymous, can be paused and continued later on and can be cancelled.

Find the Survey Here

https://ddi-survey.cs.fau.de/limesurvey/index.php/667123?lang=en

This URL will also be posted on the forum tonight.

19.7 Online Search

Online Search and Replanning

- **Note:** So far we have concentrated on offline problem solving, where the agent only acts (plan execution) after search/planning terminates.
- **Recall:** In online problem solving an agent interleaves computation and action: it computes one action at a time based on incoming perceptions.
- Online problem solving is helpful in
  - dynamic or semidynamic environments. (long computation times can be harmful)
  - stochastic environments. (solve contingencies only when they arise)
- Online problem solving is necessary in unknown environments $\rightsquigarrow$ exploration problem.

Online Search Problems

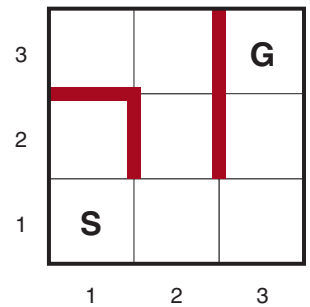
- **Observation:** Online problem solving even makes sense in deterministic, fully observable environments.
- **Definition 7.1.** A online search problem consists of a set S of states, and
  - a function $\mathrm{Actions}(s)$ that returns a list of actions allowed in state s.
  - the step cost function c, where $c(s, a, s')$ is the cost of executing action a in state s with outcome s'. (cost unknown before executing a)
  - a goal test $\mathrm{Goal\ Test}$.
- **Note:** We can only determine $\mathrm{RESULT}(s, a)$ by being in s and executing a.
- **Definition 7.2.** The competitive ratio of an online problem solving agent is the quotient of
  - offline performance, i.e. cost of optimal solutions with full information and
  - online performance, i.e. the actual cost induced by online problem solving.

Online Search Problems (Example)

- **Example 7.3 (A simple maze problem).**

  The agent starts at S and must reach G but knows nothing of the environment. In particular not that

  - $\mathrm{Up}(1,1)$ results in $(1,2)$ and
  - $\mathrm{Down}(1,1)$ results in $(1,1)$ (i.e. back)



Online Search Obstacles (Dead Ends)

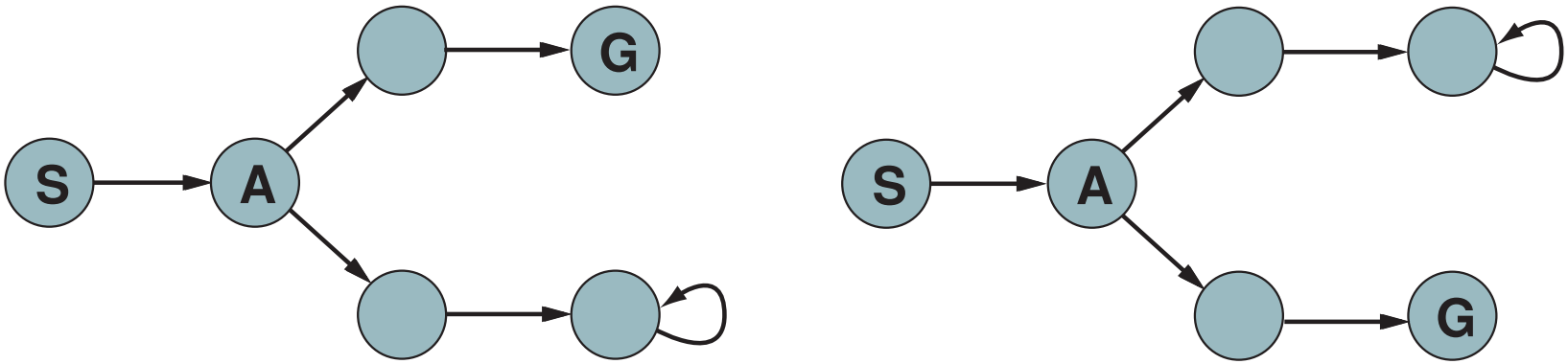
- **Definition 7.4.** We call a state a dead end, iff no state is reachable from it by an action. An action that leads to a dead end is called irreversible.
- **Note:** With irreversible actions the competitive ratio can be infinite.

Online Search Obstacles (Dead Ends)

- **Definition 7.10.** We call a state a dead end, iff no state is reachable from it by an action. An action that leads to a dead end is called irreversible.
- **Note:** With irreversible actions the competitive ratio can be infinite.
- **Observation 7.11.** *No online algorithm can avoid dead ends in all state spaces.*

Online Search Obstacles (Dead Ends)

- **Definition 7.16.** We call a state a dead end, iff no state is reachable from it by an action. An action that leads to a dead end is called irreversible.
- **Note:** With irreversible actions the competitive ratio can be infinite.
- **Observation 7.17.** *No online algorithm can avoid dead ends in all state spaces.*
- **Example 7.18.** Two state spaces that lead an online agent into dead ends:



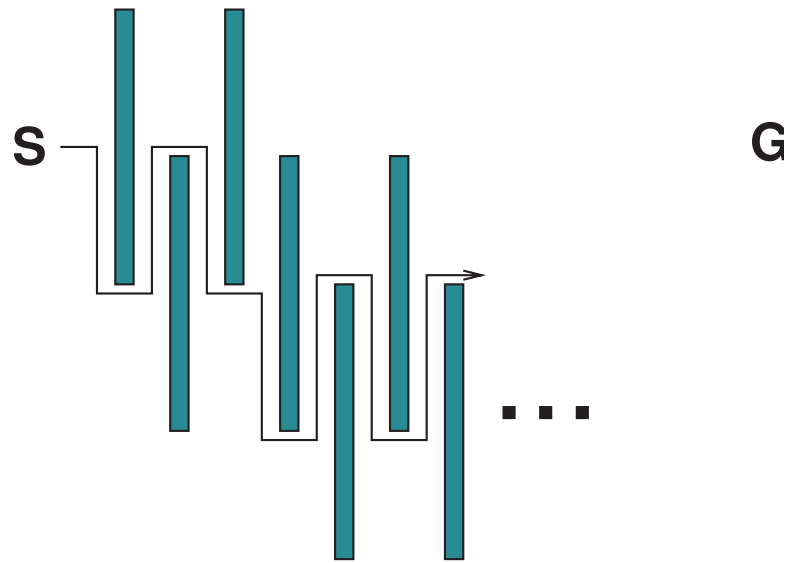
Any agent will fail in at least one of the spaces.

- **Definition 7.19.** We call **??** an adversary argument.

Online Search Obstacles (Dead Ends)

- **Definition 7.22.** We call a state a dead end, iff no state is reachable from it by an action. An action that leads to a dead end is called irreversible.
- **Note:** With irreversible actions the competitive ratio can be infinite.
- **Observation 7.23.** *No online algorithm can avoid dead ends in all state spaces.*
- **Example 7.24.** Two state spaces that lead an online agent into dead ends: Any agent will fail in at least one of the spaces.
- **Definition 7.25.** We call **??** an adversary argument.
- **Example 7.26.** Forcing an online agent into an arbitrarily inefficient route:

Whichever choice the agent makes the adversary can block with a long, thin wall



Online Search Obstacles (Dead Ends)

- **Definition 7.28.** We call a state a dead end, iff no state is reachable from it by an action. An action that leads to a dead end is called irreversible.
- **Note:** With irreversible actions the competitive ratio can be infinite.
- **Observation 7.29.** *No online algorithm can avoid dead ends in all state spaces.*
- **Example 7.30.** Two state spaces that lead an online agent into dead ends: Any agent will fail in at least one of the spaces.
- **Definition 7.31.** We call **??** an adversary argument.
- **Example 7.32.** Forcing an online agent into an arbitrarily inefficient route:
- **Observation:** Dead ends are a real problem for robots: ramps, stairs, cliffs, ...
- **Definition 7.33.** A state space is called safely explorable, iff a goal state is reachable from every reachable state.
- We will always assume this in the following.

Online Search Agents

- **Observation:** Online and offline search algorithms differ considerably:
  - For an offline agent, the environment is visible a priori.
  - An online agent builds a "map" of the environment from percepts in visited states.

  Therefore, e.g. A^* can expand any node in the fringe, but an online agent must go there to explore it.
- **Intuition:** It seems best to expand nodes in "local order" to avoid spurious travel.
- **Idea:** Depth first search seems a good fit. (must only travel for backtracking)

Online DFS Search Agent

- **Definition 7.34.** The online depth first search algorithm:

function ONLINE–DFS–AGENT(s') **returns** an action
  inputs: s', a percept that identifies the current state
  persistent: *result*, a table mapping (s, a) **to** s', initially empty
    untried, a table mapping s **to** a list of untried actions
    unbacktracked, a table mapping s **to** a list backtracks not tried
    s, a, the previous state and action, initially null
  if Goal Test(s') **then return** stop
  if $s' \notin$ *untried* **then** *untried*$[s']$:= Actions(s')
  if s is not null **then**
    result$[s, a] := s'$
    add s **to** the front of *unbacktracked*$[s']$
  if *untried*$[s']$ is empty **then**
    if *unbacktracked*$[s']$ is empty **then return** stop
    else $a :=$ an action b such that *result*$[s', b] = pop($*unbacktracked*$[s'])$
  else $a := pop($*untried*$[s'])$
  $s := s'$
  return a

- **Note:** *result* is the "environment map" constructed as the agent explores.

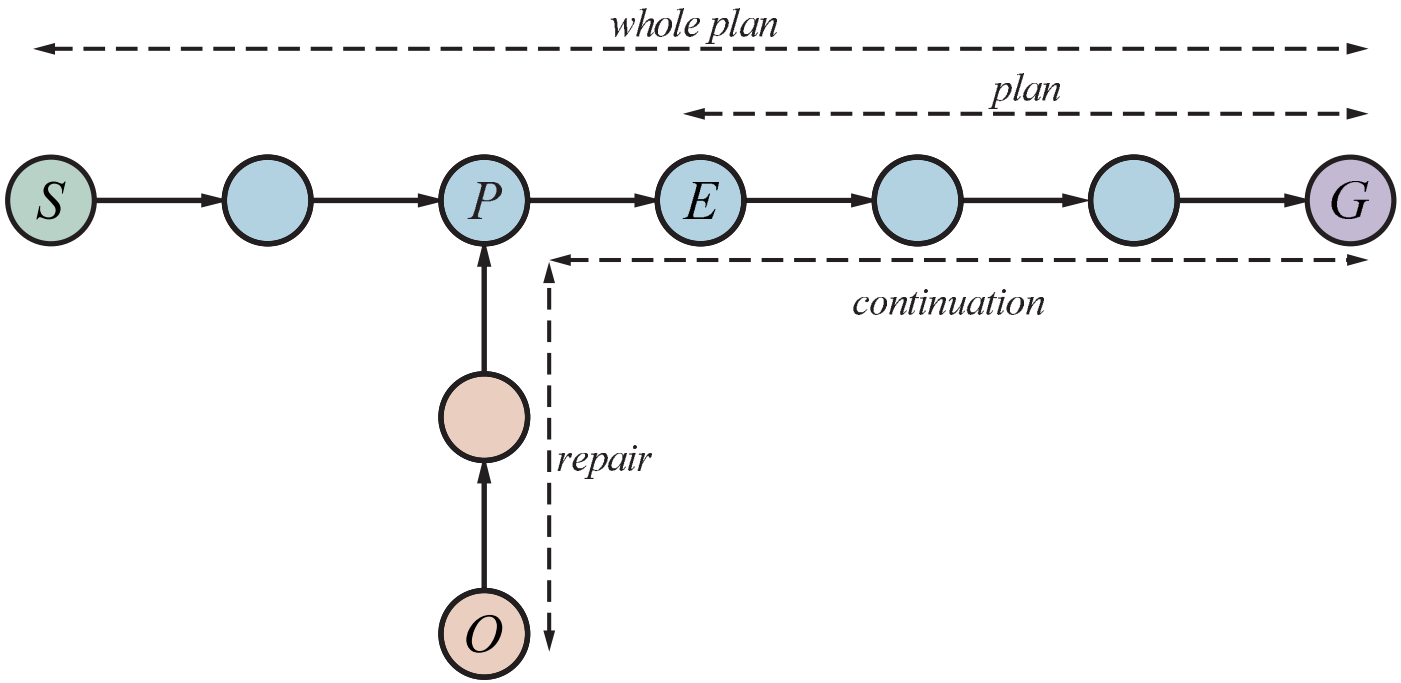
19.8 Replanning and Execution Monitoring

Replanning (Ideas)

- **Idea:** We can turn a planner P into an online problem solver by adding an action $\mathrm{RePlan}(g)$ without preconditions that re-starts P in the current state with goal g.
- **Observation:** Replanning induces a tradeoff between pre-planning and re-planning.
- **Example 8.1.** The plan $[\mathrm{RePlan}(g)]$ is a (trivially) complete plan for any goal g. (not helpful)
- **Example 8.2.** A plan with sub-plans for every contingency (e.g. what to do if a meteor strikes) may be too costly/large. (wasted effort)
- **Example 8.3.** But when a tire blows while driving into the desert, we want to have water pre-planned. (due diligence against catastrophies)
- **Observation:** In stochastic or partially observable environments we also need some form of execution monitoring to determine the need for replanning (plan repair).

Replanning for Plan Repair

- **Generally:** Replanning when the agent's model of the world is incorrect.
- **Example 8.4 (Plan Repair by Replanning).** Given a plan from S to G.



- The agent executes *wholeplan* step by step, monitoring the rest (*plan*).
- After a few steps the agent expects to be in E, but observes state O.
- Replanning: by calling the planner recursively
  - find state P in *wholeplan* and a plan *repair* from O to P. (P may be G)
  - minimize the cost of *repair* + *continuation*

Factors in World Model Failure $\rightsquigarrow$ Monitoring

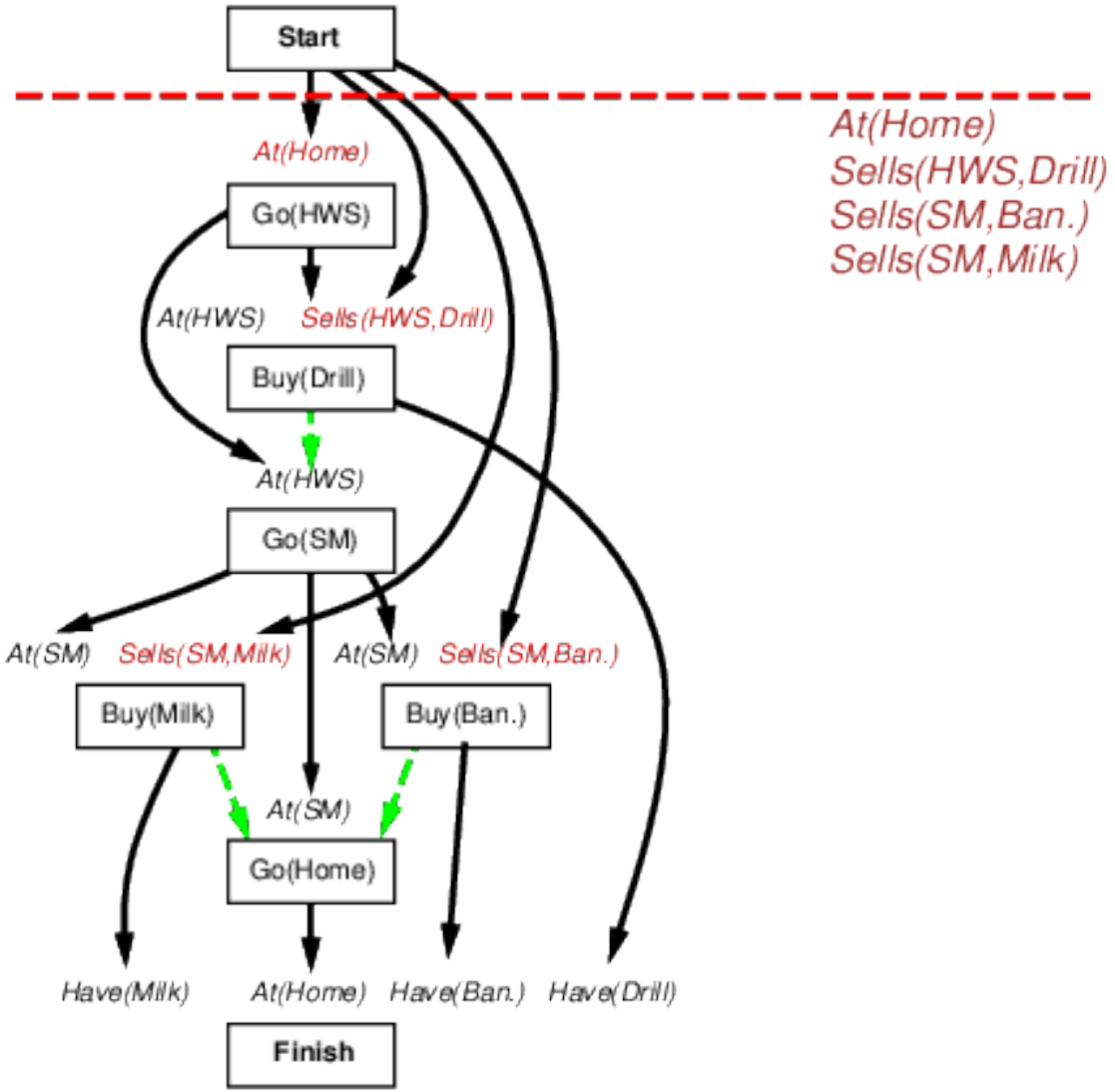
- **Generally:** The agent's world model can be incorrect, because
  - an action has a missing precondition (need a screwdriver for remove—lid)
  - an action misses an effect (painting a table gets paint on the floor)
  - it is missing a state variable (amount of paint in a can: no paint $\rightsquigarrow$ no color)
  - no provisions for exogenous events (someone knocks over a paint can)
- **Observation:** Without a way for monitoring for these, planning is very brittle.
- **Definition 8.5.** There are three levels of execution monitoring: before executing an action
  - action monitoring checks whether all preconditions still hold.
  - plan monitoring checks that the remaining plan will still succeed.
  - goal monitoring checks whether there is a better set of goals it could try to achieve.
- **Note:** **??** was a case of action monitoring leading to replanning.

Integrated Execution Monitoring and Planning

- **Problem:** Need to upgrade planing data structures by bookkeeping for execution monitoring.
- **Observation:** With their causal links, partially ordered plans already have most of the infrastructure for action monitoring:
  Preconditions of remaining plan
  $\widehat{=}$ all preconditions of remaining steps not achieved by remaining steps
  $\widehat{=}$ all causal link "crossing current time point"
- **Idea:** On failure, resume planning (e.g. by POP) to achieve open conditions from current state.
- **Definition 8.6.** IPEM (Integrated Planning, Execution, and Monitoring):
  - keep updating *Start* to match current state
  - links from actions replaced by links from *Start* when done

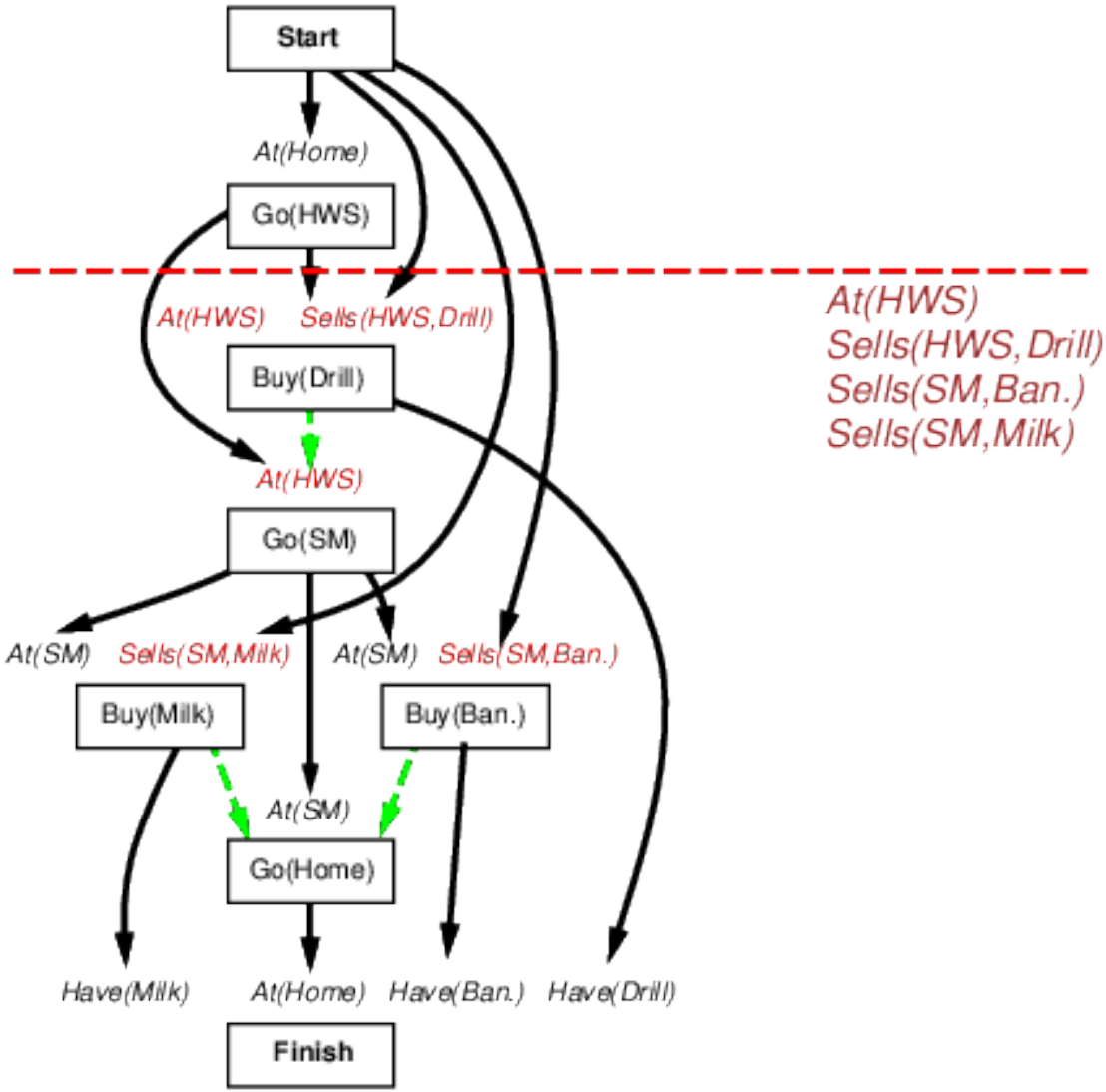
Execution Monitoring Example

- **Example 8.7 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



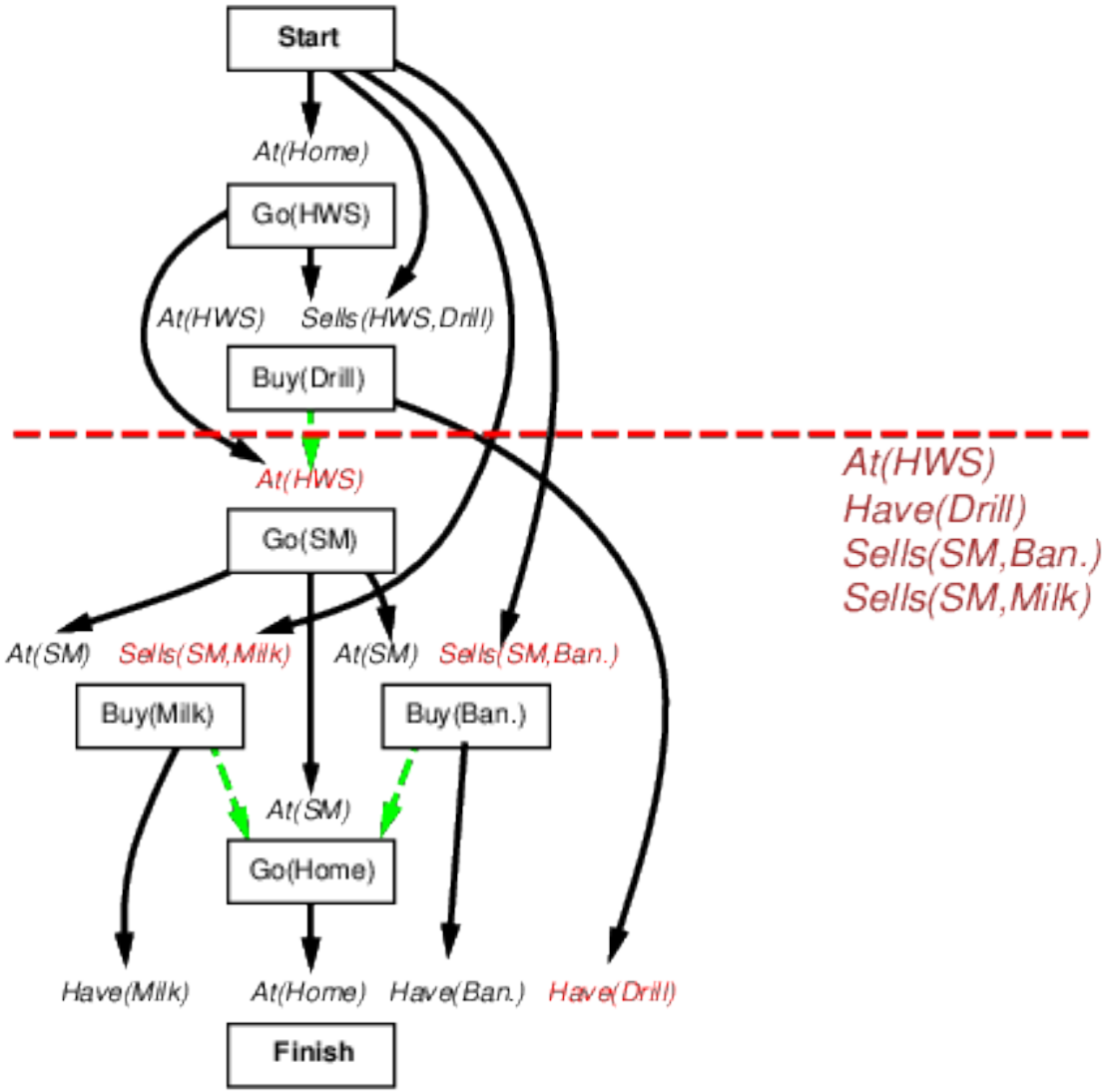
Execution Monitoring Example

- **Example 8.8 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



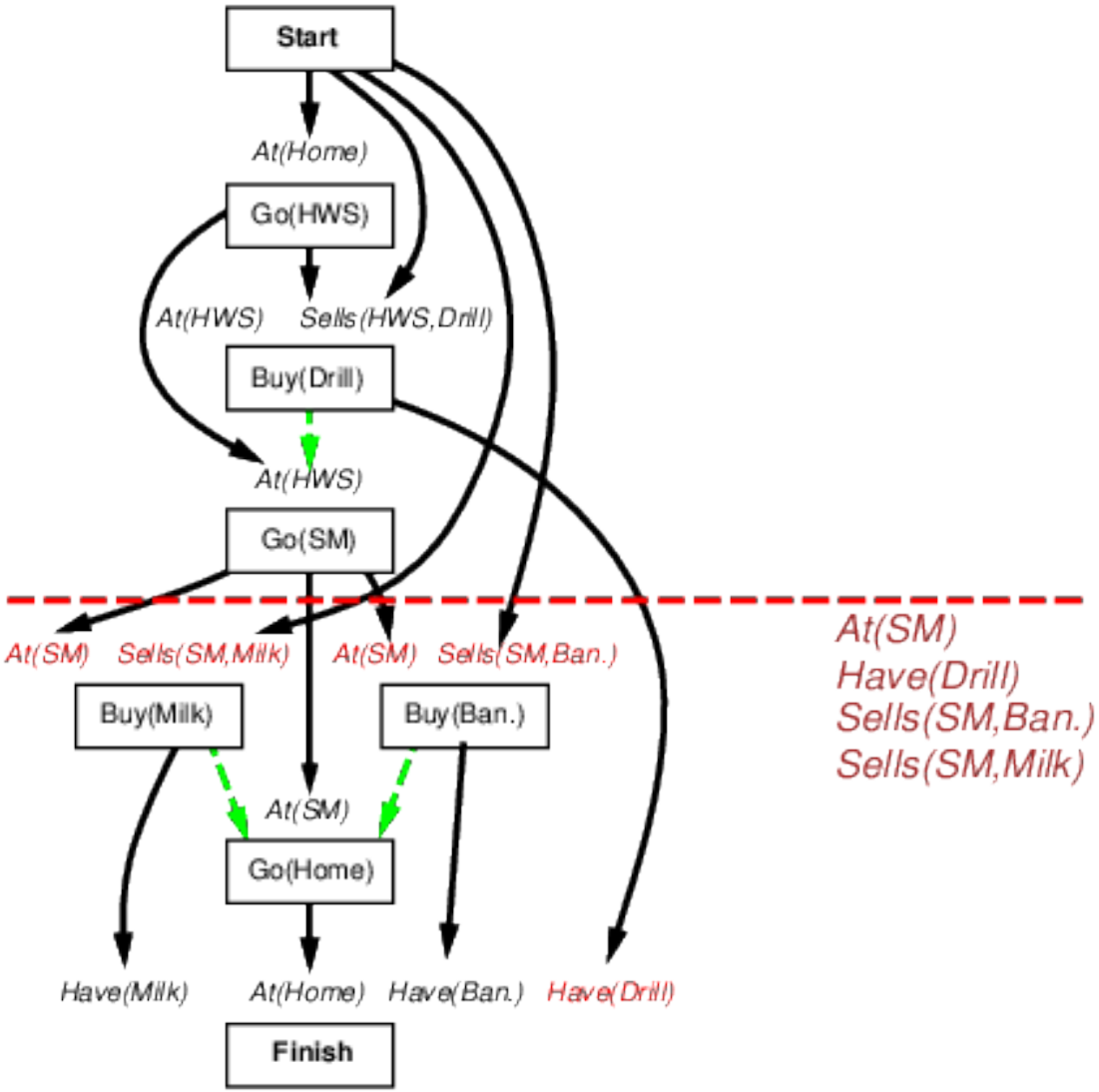
Execution Monitoring Example

- **Example 8.9 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



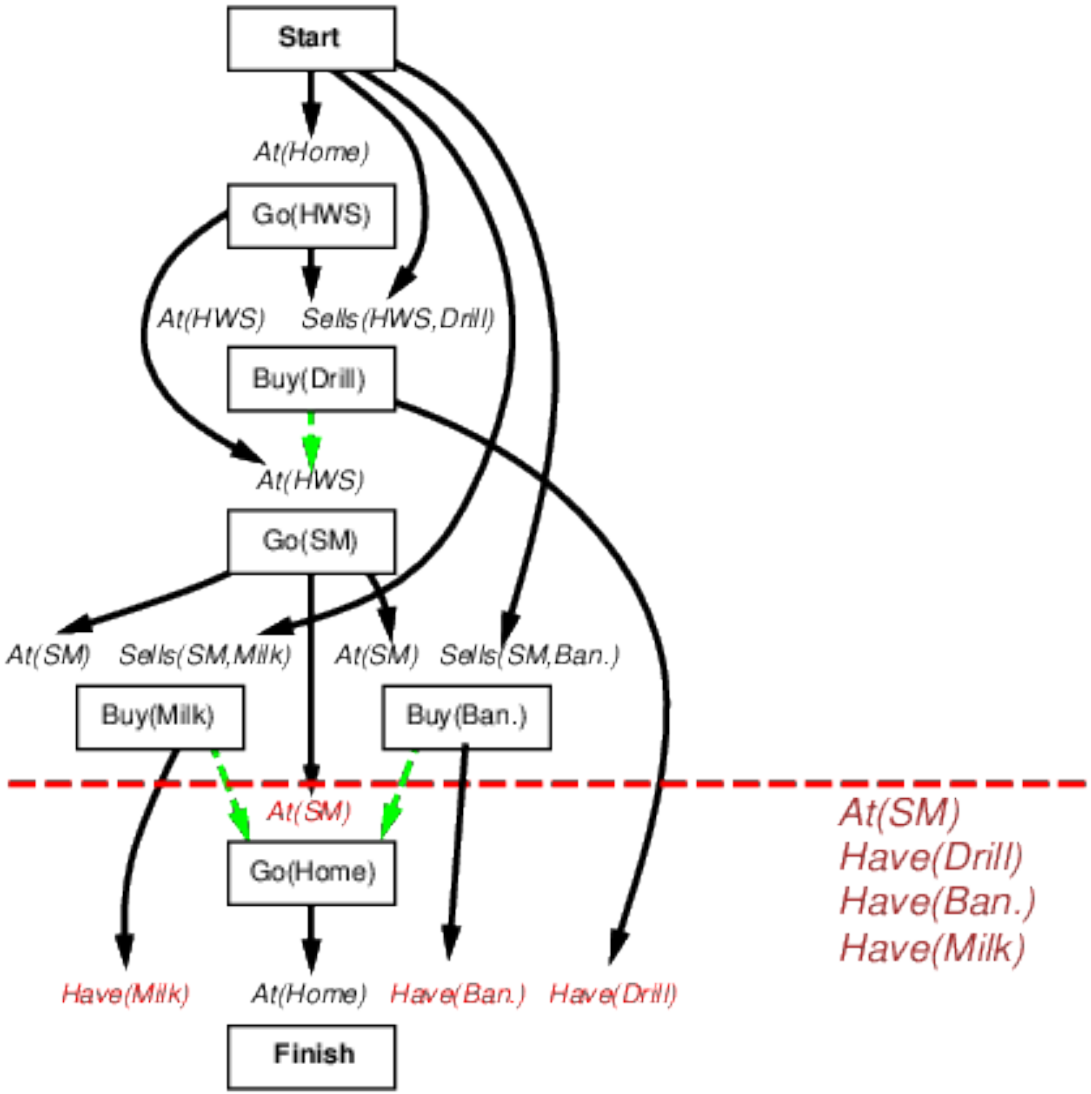
Execution Monitoring Example

- **Example 8.10 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



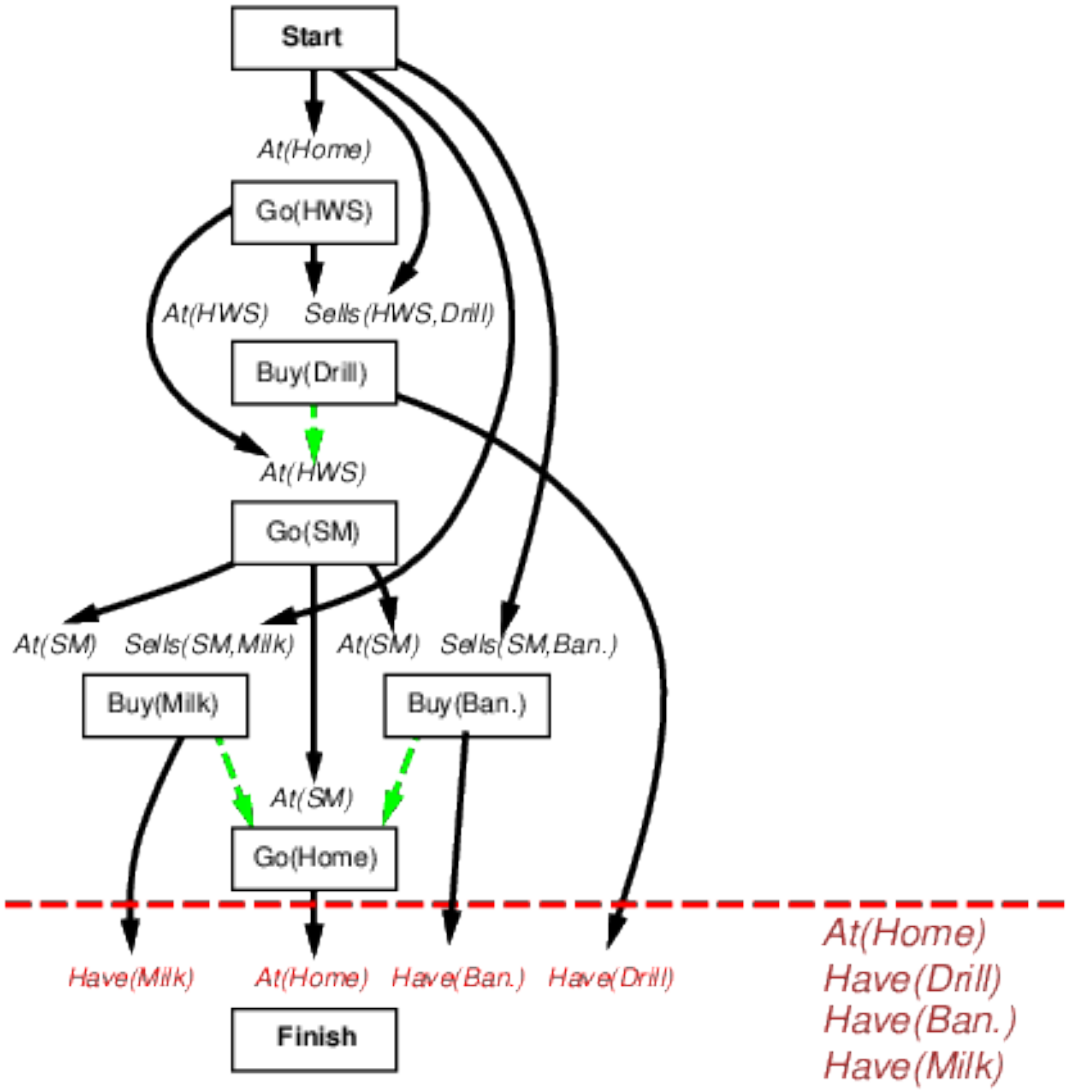
Execution Monitoring Example

- **Example 8.11 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



Execution Monitoring Example

- **Example 8.12 (Shopping for a drill, milk, and bananas).** Start/end at home, drill sold by hardware store, milk/bananas by supermarket.



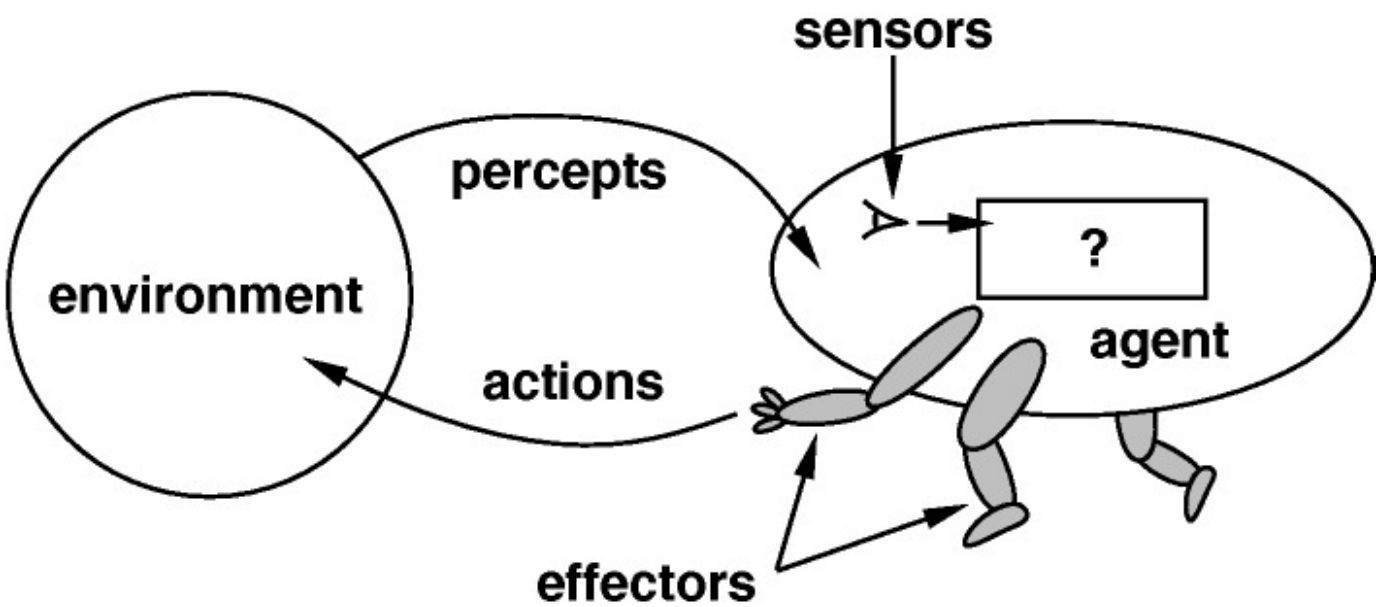
Chapter 20
What did we learn in AI 1?

Topics of AI-1 (Winter Semester)

- Getting Started
  - What is Artificial Intelligence? (situating ourselves)
  - Logic programming in Prolog (An influential paradigm)
  - Intelligent Agents (a unifying framework)
- Problem Solving
  - Problem Solving and search (Black Box World States and Actions)
  - Adversarial search (Game playing) (A nice application of search)
  - constraint satisfaction problems (Factored World States)
- Knowledge and Reasoning
  - Formal Logic as the mathematics of Meaning
  - Propositional logic and satisfiability (Atomic Propositions)
  - First-order logic and theorem proving (Quantification)
  - Logic programming (Logic + Search$\rightsquigarrow$ Programming)
  - Description logics and semantic web
- Planning
  - Planning Frameworks
  - Planning Algorithms
  - Planning and Acting in the real world

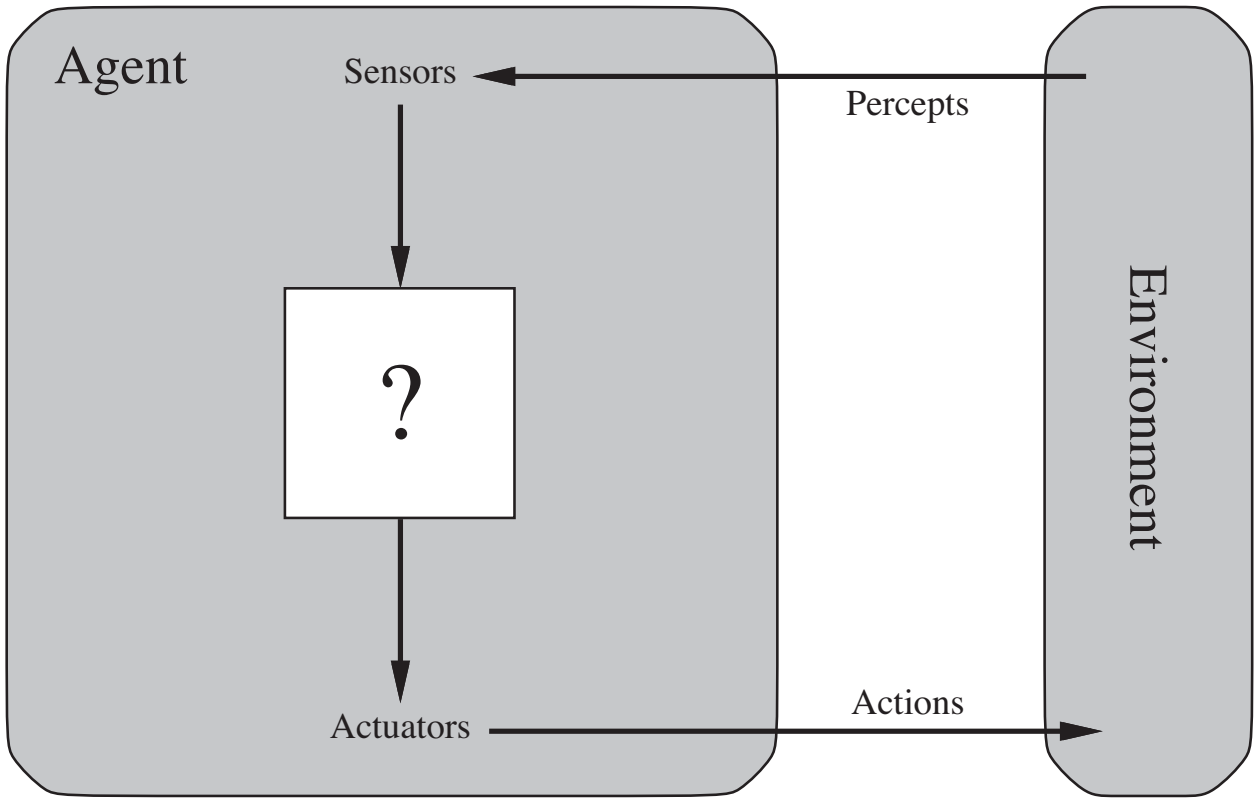
Rational Agents as an Evaluation Framework for AI

- Agents interact with the environment



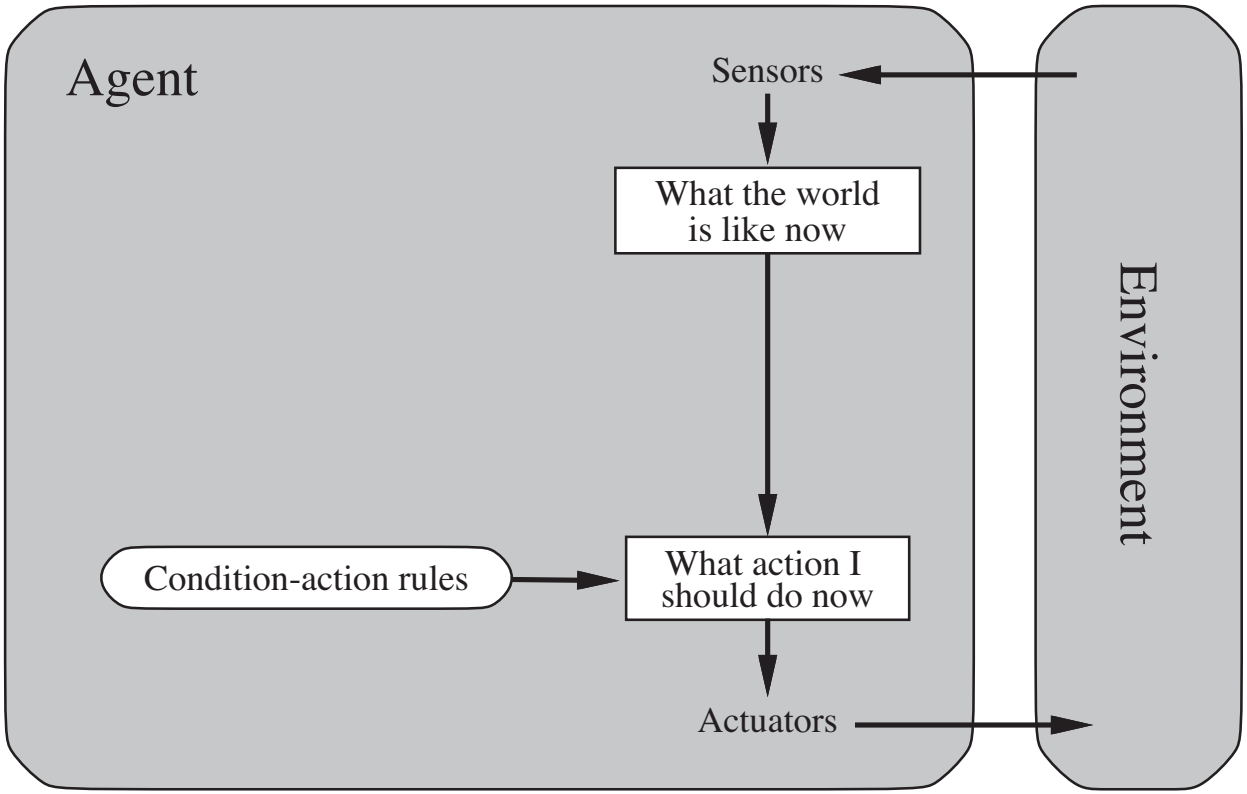
Rational Agents as an Evaluation Framework for AI

- General agent schema



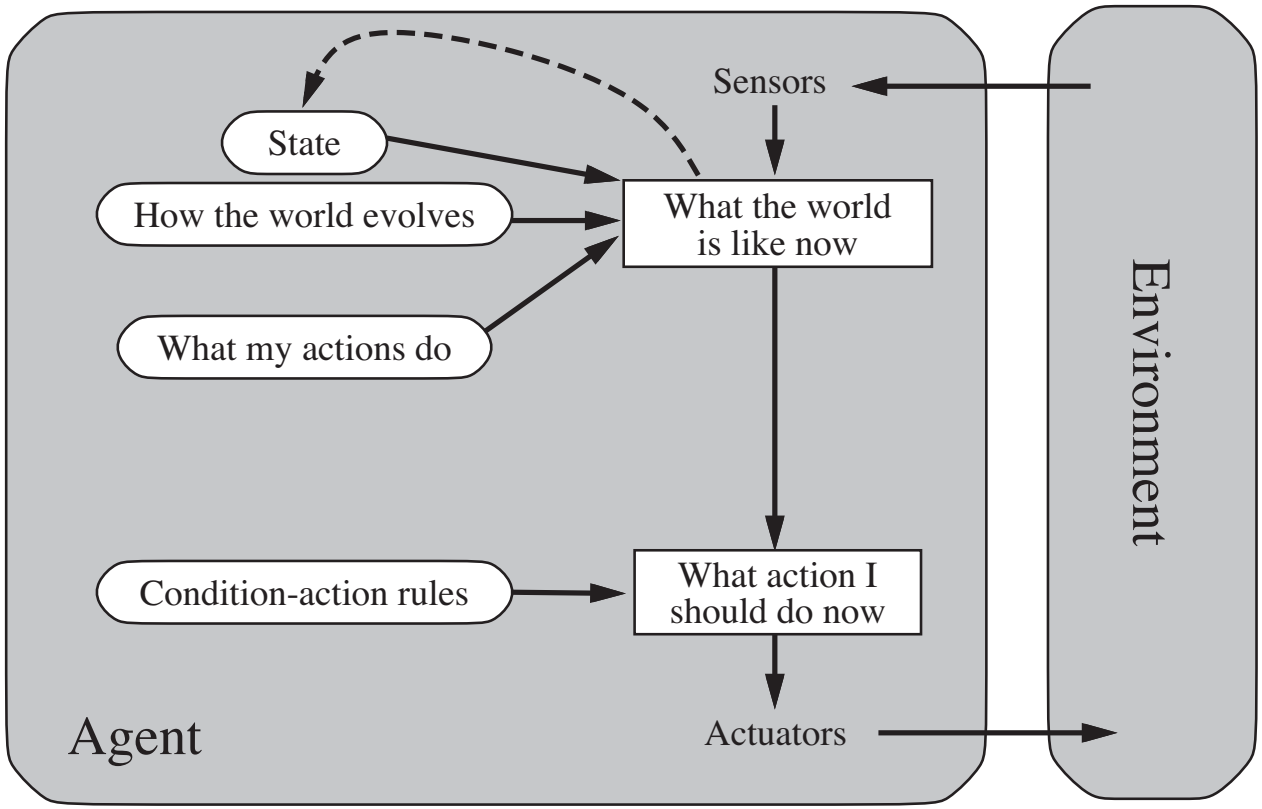
Rational Agents as an Evaluation Framework for AI

- Simple Reflex Agents



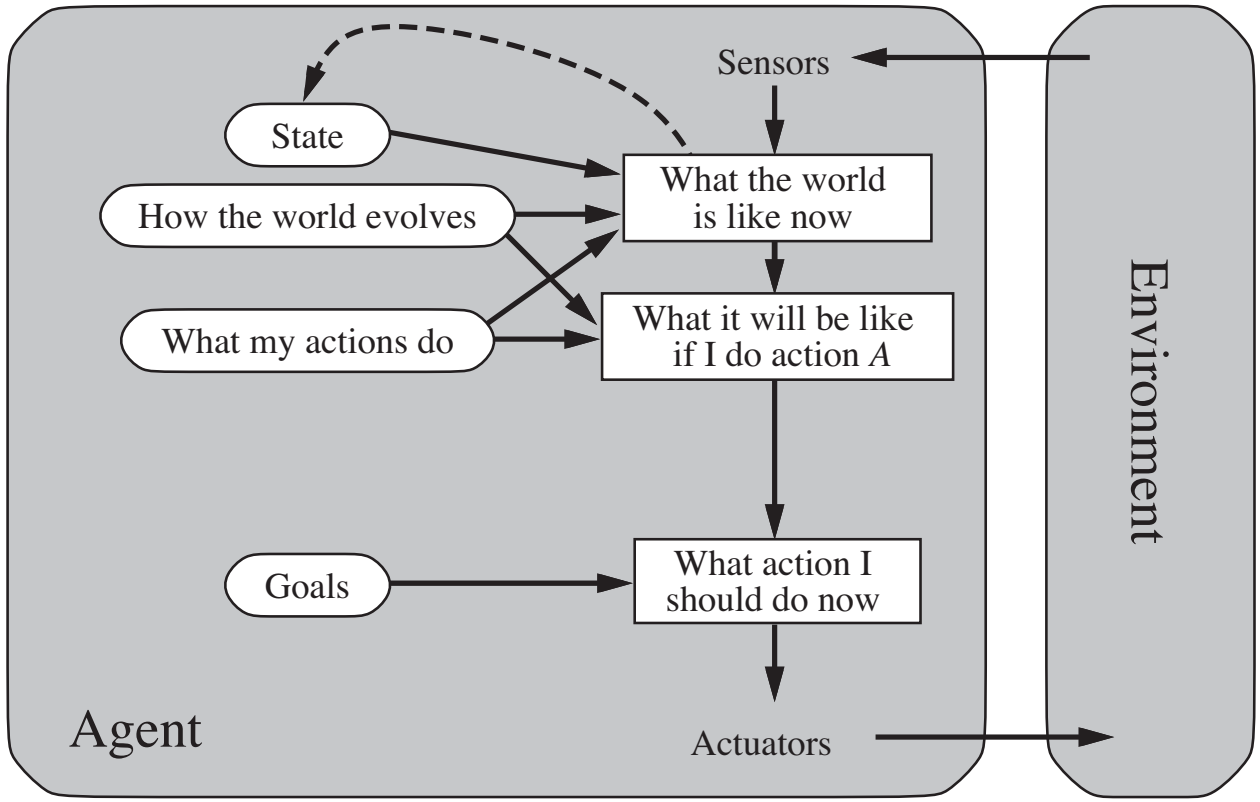
Rational Agents as an Evaluation Framework for AI

- Reflex Agents with State



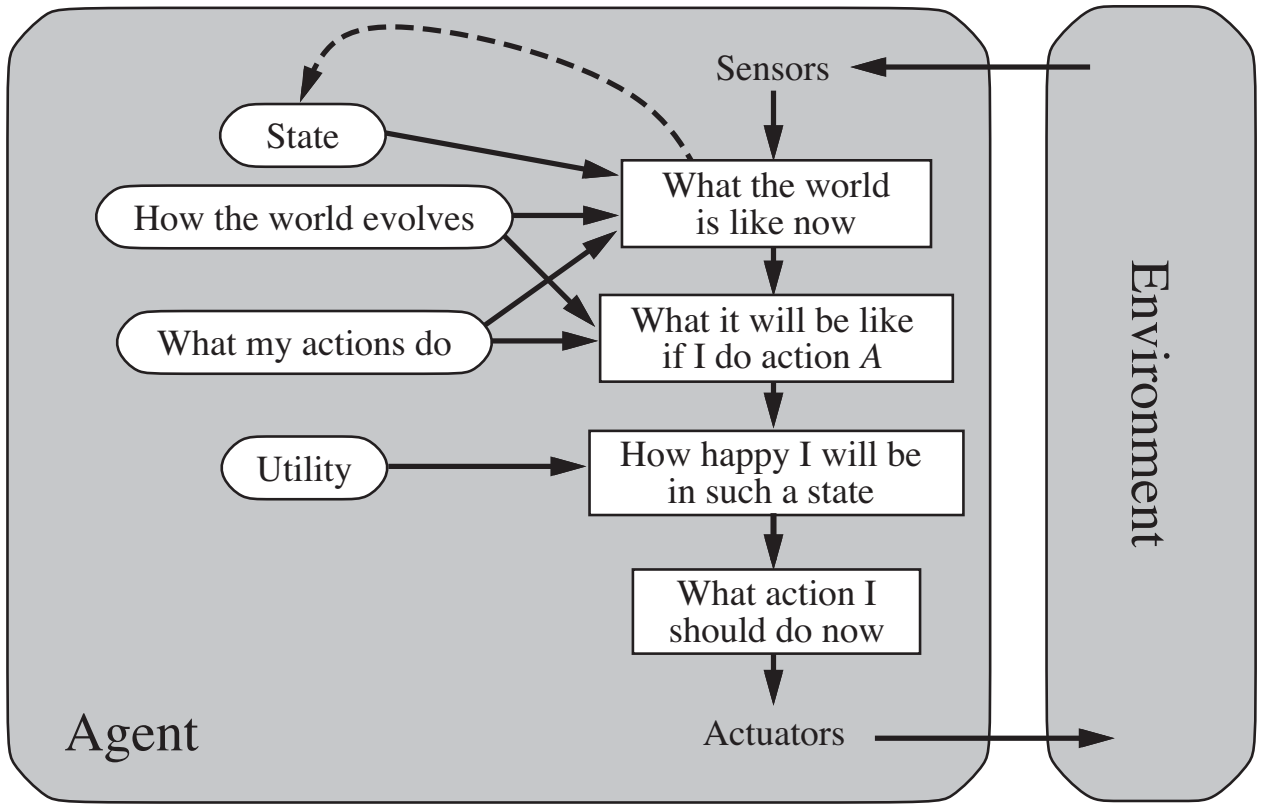
Rational Agents as an Evaluation Framework for AI

- Goal-Based Agents



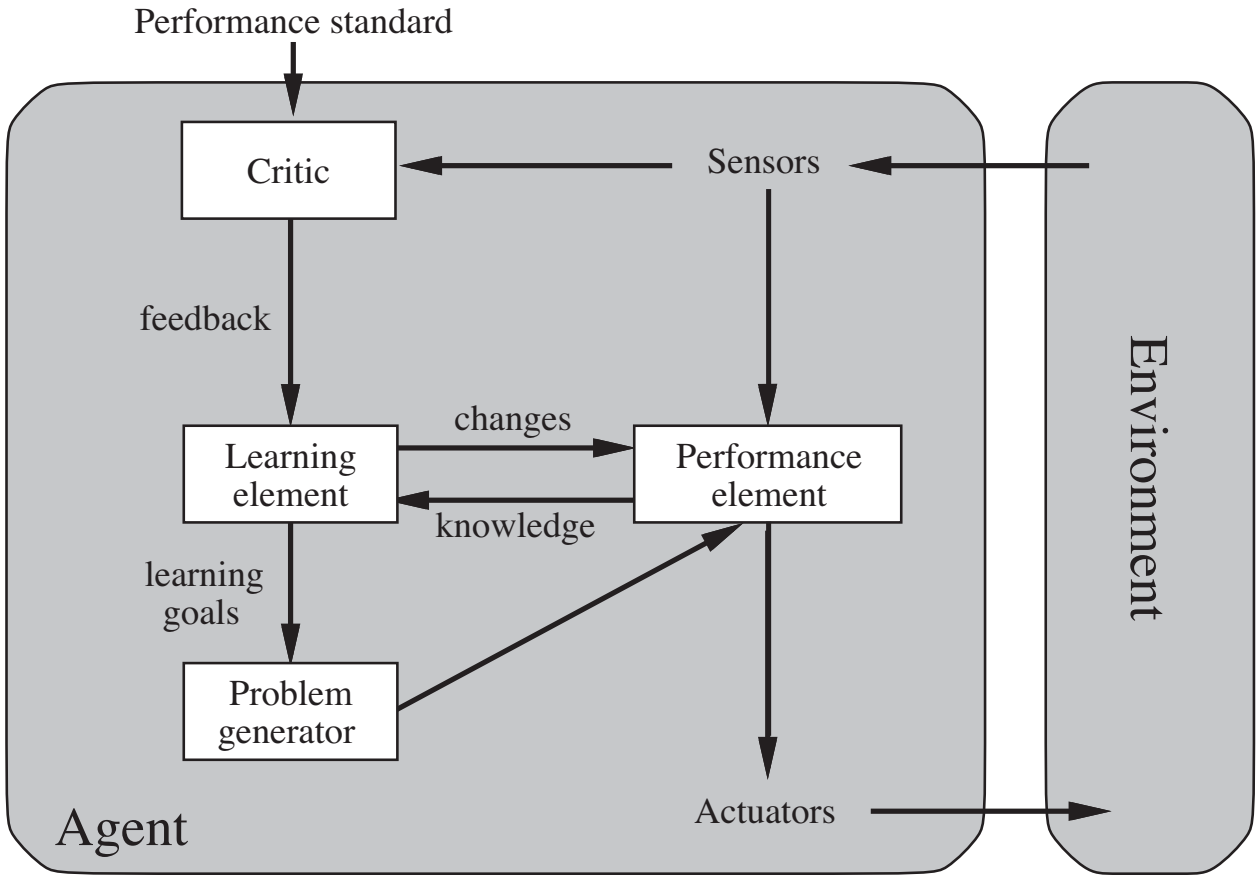
Rational Agents as an Evaluation Framework for AI

- Utility-Based Agent



Rational Agents as an Evaluation Framework for AI

- Learning Agents



Rational Agent

- **Idea:** Try to design agents that are successful (do the right thing)
- **Definition 0.1.** An agent is called rational, if it chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date. This is called the MEU principle.
- **Note:** A rational agent need not be perfect
  - only needs to maximize expected value (rational $\neq$ omniscient)
    - need not predict e.g. very unlikely but catastrophic events in the future
  - percepts may not supply all relevant information (Rational $\neq$ clairvoyant)
    - if we cannot perceive things we do not need to react to them.
    - but we may need to try to find out about hidden dangers (exploration)
  - action outcomes may not be as expected (rational $\neq$ successful)
    - but we may need to take action to ensure that they do (more often) (learning)
- Rational $\rightsquigarrow$ exploration, learning, autonomy

Symbolic AI: Adding Knowledge to Algorithms

- Problem Solving (Black Box States, Transitions, Heuristics)
  - **Framework**: Problem Solving and Search (basic tree/graph walking)
  - **Variant**: Game playing (Adversarial search) (minimax + $\alpha\beta$-Pruning)

Symbolic AI: Adding Knowledge to Algorithms

- Problem Solving (Black Box States, Transitions, Heuristics)
  - **Framework**: Problem Solving and Search (basic tree/graph walking)
  - **Variant**: Game playing (Adversarial search) (minimax + $\alpha\beta$-Pruning)
- Constraint Satisfaction Problems (heuristic search over partial assignments)
  - States as partial variable assignments, transitions as assignment
  - Heuristics informed by current restrictions, constraint graph
  - Inference as constraint propagation (transferring possible values across arcs)

Symbolic AI: Adding Knowledge to Algorithms

- Problem Solving (Black Box States, Transitions, Heuristics)
  - **Framework**: Problem Solving and Search (basic tree/graph walking)
  - **Variant**: Game playing (Adversarial search) (minimax + $\alpha\beta$-Pruning)
- Constraint Satisfaction Problems (heuristic search over partial assignments)
  - States as partial variable assignments, transitions as assignment
  - Heuristics informed by current restrictions, constraint graph
  - Inference as constraint propagation (transferring possible values across arcs)
- Describing world states by formal language (and drawing inferences)
  - Propositional logic and DPLL (deciding entailment efficiently)
  - First-order logic and ATP (reasoning about infinite domains)
  - **Digression**: Logic programming (logic + search)
  - Description logics as moderately expressive, but decidable logics

Symbolic AI: Adding Knowledge to Algorithms

- Problem Solving (Black Box States, Transitions, Heuristics)
  - **Framework**: Problem Solving and Search (basic tree/graph walking)
  - **Variant**: Game playing (Adversarial search) (minimax + $\alpha\beta$-Pruning)
- Constraint Satisfaction Problems (heuristic search over partial assignments)
  - States as partial variable assignments, transitions as assignment
  - Heuristics informed by current restrictions, constraint graph
  - Inference as constraint propagation (transferring possible values across arcs)
- Describing world states by formal language (and drawing inferences)
  - Propositional logic and DPLL (deciding entailment efficiently)
  - First-order logic and ATP (reasoning about infinite domains)
  - **Digression**: Logic programming (logic + search)
  - Description logics as moderately expressive, but decidable logics
- Planning: Problem Solving using white-box world/action descriptions
  - **Framework**: describing world states in logic as sets of propositions and actions by preconditions and add/delete lists
  - **Algorithms**: e.g heuristic search by problem relaxations

Topics of AI-2 (Summer Semester)

- Uncertain Knowledge and Reasoning
  - Uncertainty
  - Probabilistic reasoning
  - Making Decisions in Episodic Environments
  - Problem Solving in Sequential Environments
- Foundations of machine learning
  - Learning from Observations
  - Knowledge in Learning
  - Statistical Learning Methods
- Communication (If there is time)
  - Natural Language Processing
  - Natural Language for Communication

References I

[BF95] Avrim L. Blum and Merrick L. Furst. "Fast planning through planning graph analysis". In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by Chris S. Mellish. Montreal, Canada: Morgan Kaufmann, San Mateo, CA, 1995, pp. 1636–1642.

[BF97] Avrim L. Blum and Merrick L. Furst. "Fast planning through planning graph analysis". In: *Artificial Intelligence* 90.1-2 (1997), pp. 279–298.

[BG01] Blai Bonet and Héctor Geffner. "Planning as Heuristic Search". In: *Artificial Intelligence* 129.1–2 (2001), pp. 5–33.

[BG99] Blai Bonet and Héctor Geffner. "Planning as Heuristic Search: New Results". In: *Proceedings of the 5th European Conference on Planning (ECP'99)*. Ed. by S. Biundo and M. Fox. Springer-Verlag, 1999, pp. 60–72.

[Bon+12] Blai Bonet et al., eds. *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press, 2012.

References II

[DHK15] Carmel Domshlak, Jörg Hoffmann, and Michael Katz. "Red-Black Planning: A New Systematic Approach to Partial Delete Relaxation". In: *Artificial Intelligence* 221 (2015), pp. 73–114.

[Ede01] Stefan Edelkamp. "Planning with Pattern Databases". In: *Proceedings of the 6th European Conference on Planning (ECP'01)*. Ed. by A. Cesta and D. Borrajo. Springer-Verlag, 2001, pp. 13–24.

[FL03] Maria Fox and Derek Long. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 61–124.

[FN71] Richard E. Fikes and Nils Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Artificial Intelligence* 2 (1971), pp. 189–208.

[Ger+09] Alfonso Gerevini et al. "Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners". In: *Artificial Intelligence* 173.5-6 (2009), pp. 619–668.

References III

[GNT04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[GSS03] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. "Planning through Stochastic Local Search and Temporal Action Graphs". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 239–290.

[HD09] Malte Helmert and Carmel Domshlak. "Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?" In: *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*. Ed. by Alfonso Gerevini et al. AAAI Press, 2009, pp. 162–169.

[HE05] Jörg Hoffmann and Stefan Edelkamp. "The Deterministic Part of IPC-4: An Overview". In: *Journal of Artificial Intelligence Research* 24 (2005), pp. 519–579.

[Hel06] Malte Helmert. "The Fast Downward Planning System". In: *Journal of Artificial Intelligence Research* 26 (2006), pp. 191–246.

References IV

[HG00] Patrik Haslum and Hector Geffner. "Admissible Heuristics for Optimal Planning". In: *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*. Ed. by S. Chien, R. Kambhampati, and C. Knoblock. Breckenridge, CO: AAAI Press, Menlo Park, 2000, pp. 140–149.

[HG08] Malte Helmert and Hector Geffner. "Unifying the Causal Graph and Additive Heuristics". In: *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*. Ed. by Jussi Rintanen et al. AAAI Press, 2008, pp. 140–147.

[HHH07] Malte Helmert, Patrik Haslum, and Jörg Hoffmann. "Flexible Abstraction Heuristics for Optimal Sequential Planning". In: *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*. Ed. by Mark Boddy, Maria Fox, and Sylvie Thiebaux. Providence, Rhode Island, USA: Morgan Kaufmann, 2007, pp. 176–183.

References V

[HN01] Jörg Hoffmann and Bernhard Nebel. "The FF Planning System: Fast Plan Generation Through Heuristic Search". In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 253–302.

[KD09] Erez Karpas and Carmel Domshlak. "Cost-Optimal Planning with Landmarks". In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*. Ed. by C. Boutilier. Pasadena, California, USA: Morgan Kaufmann, July 2009, pp. 1728–1733.

[KHD13] Michael Katz, Jörg Hoffmann, and Carmel Domshlak. "Who Said We Need to Relax *all* Variables?" In: *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*. Ed. by Daniel Borrajo et al. Rome, Italy: AAAI Press, 2013, pp. 126–134.

[KHH12a] Michael Katz, Jörg Hoffmann, and Malte Helmert. "How to Relax a Bisimulation?" In: *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. Ed. by Blai Bonet et al. AAAI Press, 2012, pp. 101–109.

References VI

[KHH12b] Emil Keyder, Jörg Hoffmann, and Patrik Haslum. "Semi-Relaxed Plan Heuristics". In: *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. Ed. by Blai Bonet et al. AAAI Press, 2012, pp. 128–136.

[Koe+97] Jana Koehler et al. "Extending Planning Graphs to an ADL Subset". In: *Proceedings of the 4th European Conference on Planning (ECP'97)*. Ed. by S. Steel and R. Alami. Springer-Verlag, 1997, pp. 273–285. URL: ftp://ftp.informatik.uni-freiburg.de/papers/ki/koehler-etal-ecp-97.ps.gz.

[KS00] Jana Köhler and Kilian Schuster. "Elevator Control as a Planning Problem". In: *AIPS 2000 Proceedings*. AAAI, 2000, pp. 331–338. URL: https://www.aaai.org/Papers/AIPS/2000/AIPS00-036.pdf.

[KS92] Henry A. Kautz and Bart Selman. "Planning as Satisfiability". In: *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*. Ed. by B. Neumann. Vienna, Austria: Wiley, Aug. 1992, pp. 359–363.

References VII

[KS98] Henry A. Kautz and Bart Selman. "Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search". In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence AAAI-96*. MIT Press, 1998, pp. 1194–1201.

[McD+98] Drew McDermott et al. *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee. 1998.

[NHH11] Raz Nissim, Jörg Hoffmann, and Malte Helmert. "Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstraction in Optimal Planning". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*. Ed. by Toby Walsh. AAAI Press/IJCAI, 2011, pp. 1983–1990.

[NS63] Allen Newell and Herbert Simon. "GPS, a program that simulates human thought". In: *Computers and Thought*. Ed. by E. Feigenbaum and J. Feldman. McGraw-Hill, 1963, pp. 279–293.

References VIII

[PW92] J. Scott Penberthy and Daniel S. Weld. "UCPOP: A Sound, Complete, Partial Order Planner for ADL". In: *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference (KR-92)*. Ed. by B. Nebel, W. Swartout, and C. Rich. Cambridge, MA: Morgan Kaufmann, Oct. 1992, pp. 103–114. URL: ftp://ftp.cs.washington.edu/pub/ai/ucpop-kr92.ps.Z.

[RHN06] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. "Planning as satisfiability: parallel plans and algorithms for plan search". In: *Artificial Intelligence* 170.12-13 (2006), pp. 1031–1080.

[Rin10] Jussi Rintanen. "Heuristics for Planning with SAT". In: *Proceeedings of the 16th International Conference on Principles and Practice of Constraint Programming*. 2010, pp. 414–428.

[RW10] Silvia Richter and Matthias Westphal. "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks". In: *Journal of Artificial Intelligence Research* 39 (2010), pp. 127–177.