Algorithms and Satisfiability

10. Abstraction Heuristics For Planning

It's a Long Way to the Goal, But How Long Exactly?

Álvaro Torralba

AALBORG UNIVERSITET

Spring 2023

Thanks to Jörg Hoffmann for slide sources

Agenda

Agenda

Planning

Ambition:

Write one program (planner) that can solve all sequential decision-making problems.

How do we describe our problem to the planner?

- A *logical description* of the possible states
- A *logical description* of the initial state I
- A *logical description* of the goal condition G
- *logical description* of the set A of actions in terms of preconditions and effects

$\rightarrow$ Solution (plan) $=$ sequence of actions from A, transforming I into a state that satisfies G.

$\rightarrow$Here, we focus on the simplest form of planning: Classical Planning. In the mini-project, we will briefly cover other extensions.

Algorithmic Problems in Planning

Satisficing Planning

Input: A planning task Π.
Output: A plan for Π, or "unsolvable" if no plan for Π exists.

Optimal Planning

Input: A planning task Π.
Output: An *optimal* plan for Π, or "unsolvable" if no plan for Π exists.

$\rightarrow$ The techniques successful for either one of these are almost disjoint. And satisficing planning is *much* more effective in practice.

$\rightarrow$ Programs solving these problems are called (optimal) planners, planning systems, or planning tools.

Classical Planning IPC Overview

- **IPC 2000:** Winner heuristic search.
- **IPC 2002:** Winner heuristic search.
- **IPC 2004:** Winner satisficing heuristic search; optimal compilation to SAT.
- **IPC 2006:** Winner satisficing heuristic search; optimal compilation to SAT.
- **IPC 2008:** Winner satisficing heuristic search; optimal symbolic search.
- **IPC 2011:** Winner satisficing heuristic search; optimal heuristic search.
- **IPC 2014:** Winner satisficing heuristic search; optimal symbolic search.
- **IPC 2018:** Winner satisficing heuristic search; optimal portfolio/symbolic search/heuristic search.

→ This and next lecture focus on planning as heuristic search;
Chapter 12 will focus on compilation to SAT and symbolic search.

→ This is a VERY short summary of the history of the IPC! There are many different categories, and many different awards.

Our Agenda for This Topic

Planning and heuristic search were already introduced in the Machine Intelligence course, as it is a sub-area of Artificial Intelligence. Here, we focus on how to (1) design efficient algorithms that can solve planning tasks in practice; and (2) make use of existing planners by encoding your problems as planning tasks.

- **This Chapter:** How to automatically generate a heuristic function, given planning language input?

  → Focusing on heuristic search as the solution method, this is the main question that needs to be answered.

- **Mini-project:** How to use planners to solve your problems?
- **Chapter 11:** How to solve planning using SAT? How to solve planning using BDDs?

  →Other algorithms to solve planning based on the techniques we have seen before.

STRIPS Planning: Syntax

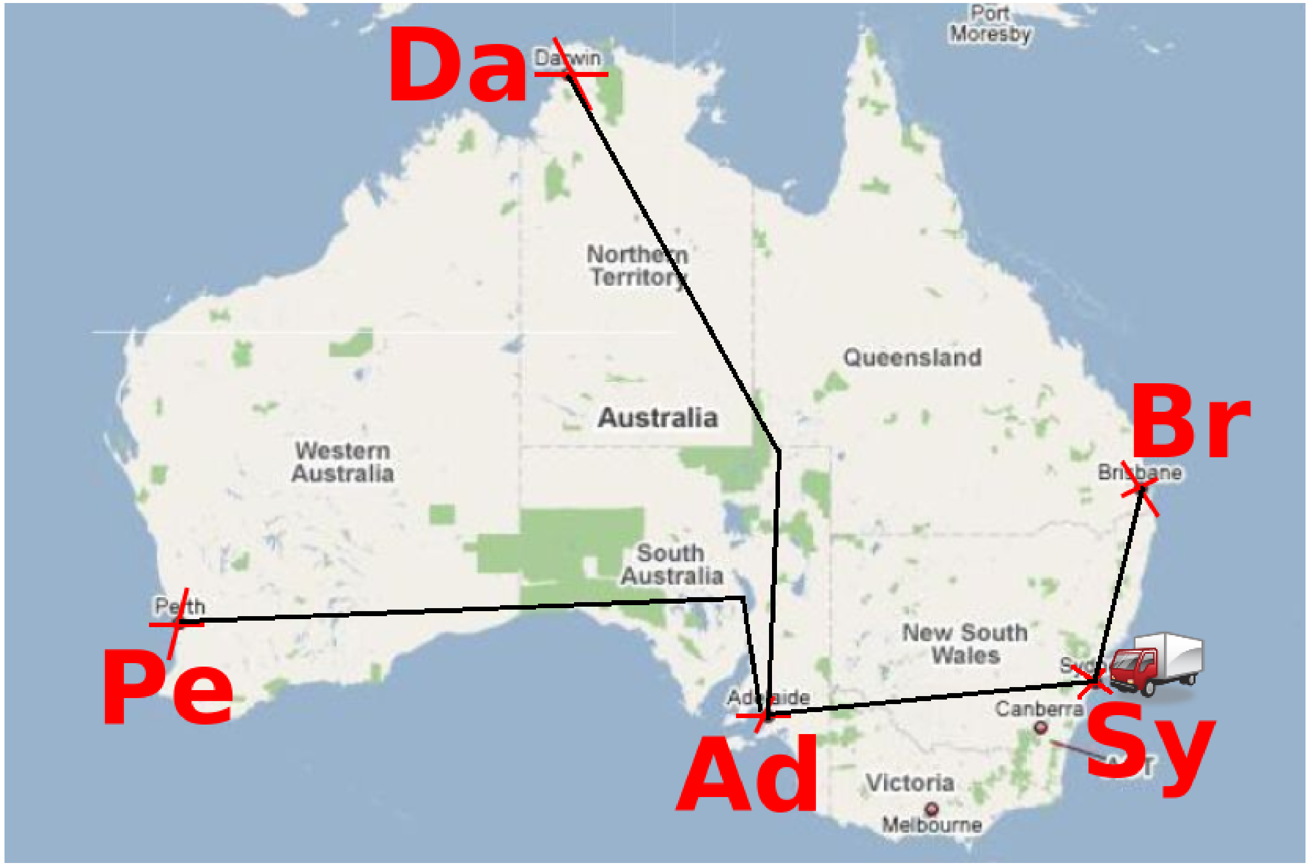
Definition (STRIPS Planning Task). *A STRIPS planning task, short planning task, is a 4-tuple $\Pi = (P, A, I, G)$ where:*

- *P is a finite set of facts (aka propositions).*
- *A is a finite set of actions; each $a \in A$ is a triple $a = (pre_a, add_a, del_a)$ of subsets of P referred to as the action's precondition, add list, and delete list respectively; we require that $add_a \cap del_a = \emptyset$.*
- *$I \subseteq P$ is the initial state.*
- *$G \subseteq P$ is the goal.*

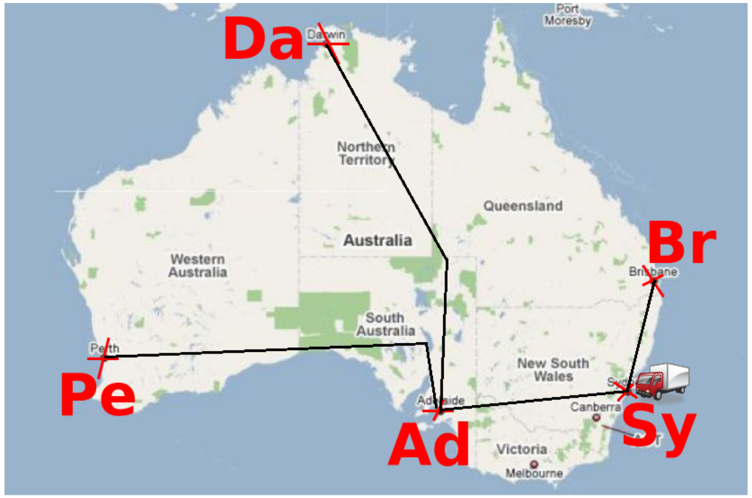
We will often give each action $a \in A$ a name (a string), and identify a with that name.

$\rightarrow$We'll see some extensions beyond STRIPS for the mini-project, when we discuss PDDL.

"TSP" in Australia



STRIPS Encoding of "TSP"



- Facts P: $\{at(x), visited(x) \mid x \in \{Sydney, Adelaide, Brisbane, Perth, Darwin\}\}$.
- Initial state I: $\{at(Sydney), visited(Sydney)\}$.
- Goal G:
  $\{at(Sydney)\} \cup \{visited(x) \mid x \in \{Sydney, Adelaide, Brisbane, Perth, Darwin\}\}$.
- Actions $a \in A$: $drive(x, y)$ where x, y have a road.
  Precondition pre_a: $\{at(x)\}$.
  Add list add_a: $\{at(y), visited(y)\}$.
  Delete list del_a: $\{at(x)\}$.
- Plan: $\langle drive(Sydney, Brisbane), drive(Brisbane, Sydney), drive(Sydney, Adelaide), drive(Adelaide, Perth), drive(Perth, Adelaide), drive(Adelaide, Darwin), drive(Darwin, Adelaide), drive(Adelaide, Sydney)\rangle$.

STRIPS Planning: Semantics

Definition (STRIPS State Space). *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task. The state space of Π is $\Theta_\Pi = (S, A, T, I, S^G)$ where:*

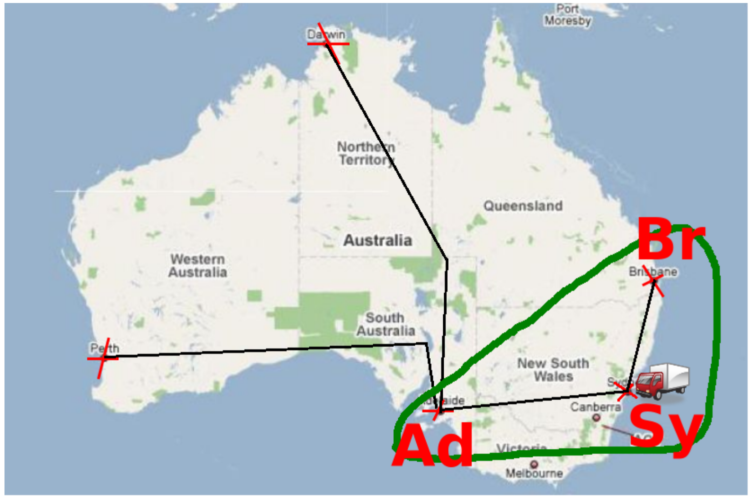
- *The states (also world states) $S = 2^P$ are the subsets of P.*
- *A is Π's action set.*
- *The transitions are $T = \{s \xrightarrow{a} s' \mid pre_a \subseteq s, s' = appl(s, a)\}$.*

  If $pre_a \subseteq s$, then a is applicable in s and $appl(s, a) := (s \cup add_a) \setminus del_a$.
  If $pre_a \not\subseteq s$, then $appl(s, a)$ is undefined.
- *I is Π's initial state.*
- *The goal states $S^G = \{s \in S \mid G \subseteq s\}$ are those that satisfy Π's goal.*

An (optimal) plan for $s \in S$ is an (optimal) solution for s in Θ_Π, i.e., a path from s to some $s' \in S^G$. A solution for I is called a plan for Π. Π is solvable if a plan for Π exists.

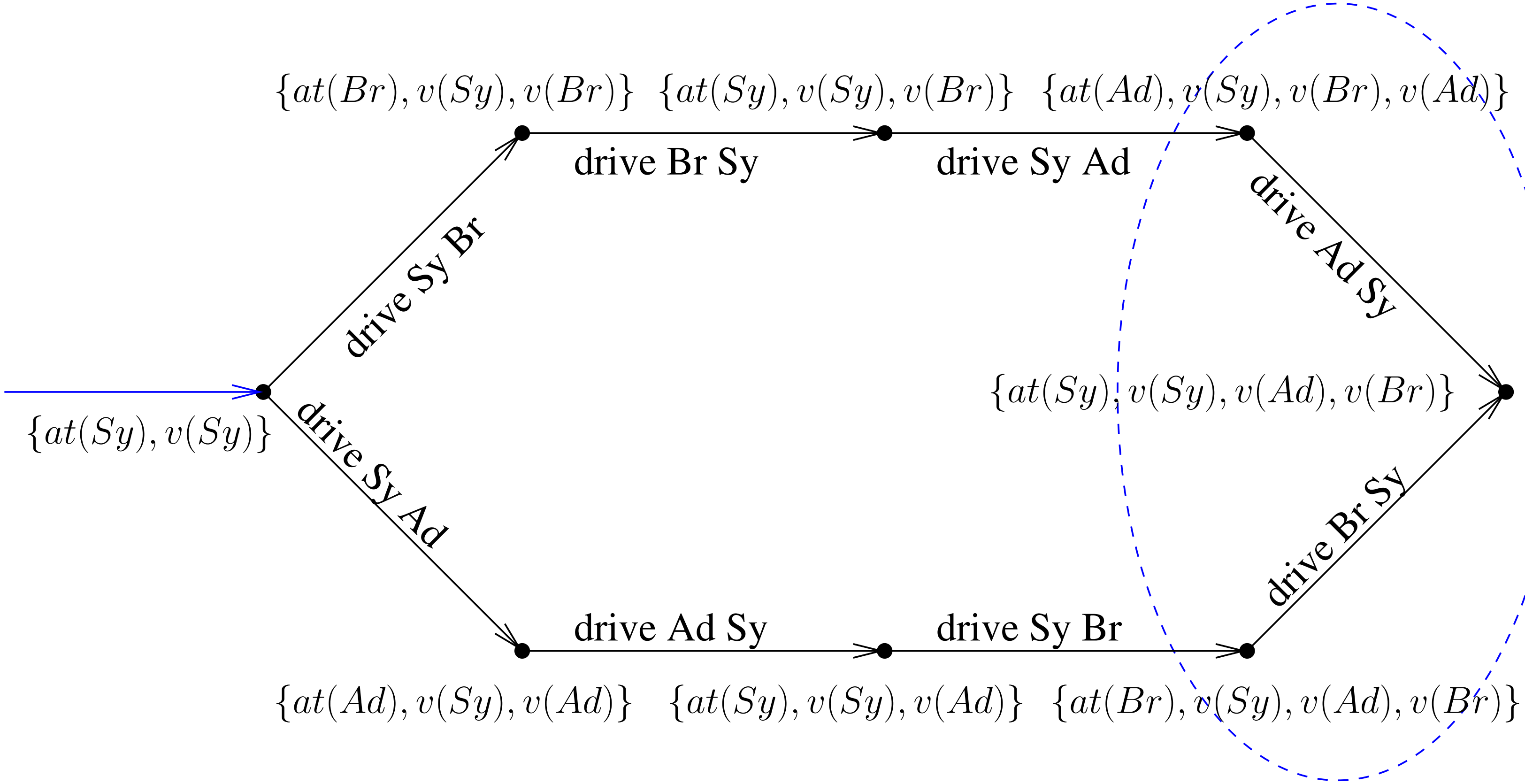
For $\vec{a} = \langle a_1, \dots, a_n \rangle$, $appl(s, \vec{a}) := appl(\dots appl(appl(s, a_1), a_2) \dots, a_n)$ if each a_i is applicable in the respective state; else, $appl(s, \vec{a})$ is undefined.

STRIPS Encoding of Simplified "TSP"



- Facts P: $\{at(x), visited(x) \mid x \in \{Sydney, Adelaide, Brisbane\}\}$.
- Initial state I: $\{at(Sydney), visited(Sydney)\}$.
- Goal G: $\{visited(x) \mid x \in \{Sydney, Adelaide, Brisbane\}\}$. (Note: no "$at(Sydney)$".)
- Actions $a \in A$: $drive(x, y)$ where x, y have a road.
  Precondition pre_a: $\{at(x)\}$.
  Add list add_a: $\{at(y), visited(y)\}$.
  Delete list del_a: $\{at(x)\}$.

STRIPS Encoding of Simplified "TSP": State Space



$\rightarrow$ Is this actually the state space? No, only the reachable part. E.g., Θ_Π also includes the states $\{v(Sy)\}$ and $\{at(Sy), at(Br)\}$.

Decision Problems in (STRIPS) Planning

Definition (PlanEx). *Given a STRIPS task Π, does there exists a plan for Π?*
$\rightarrow$ Corresponds to satisficing planning.

Theorem. *PlanEx is* **PSPACE***-complete.*

Definition (PlanLen). *Given a STRIPS task Π and an integer K, does there exists a plan for Π of length at most K?* $\rightarrow$ Corresponds to optimal planning.

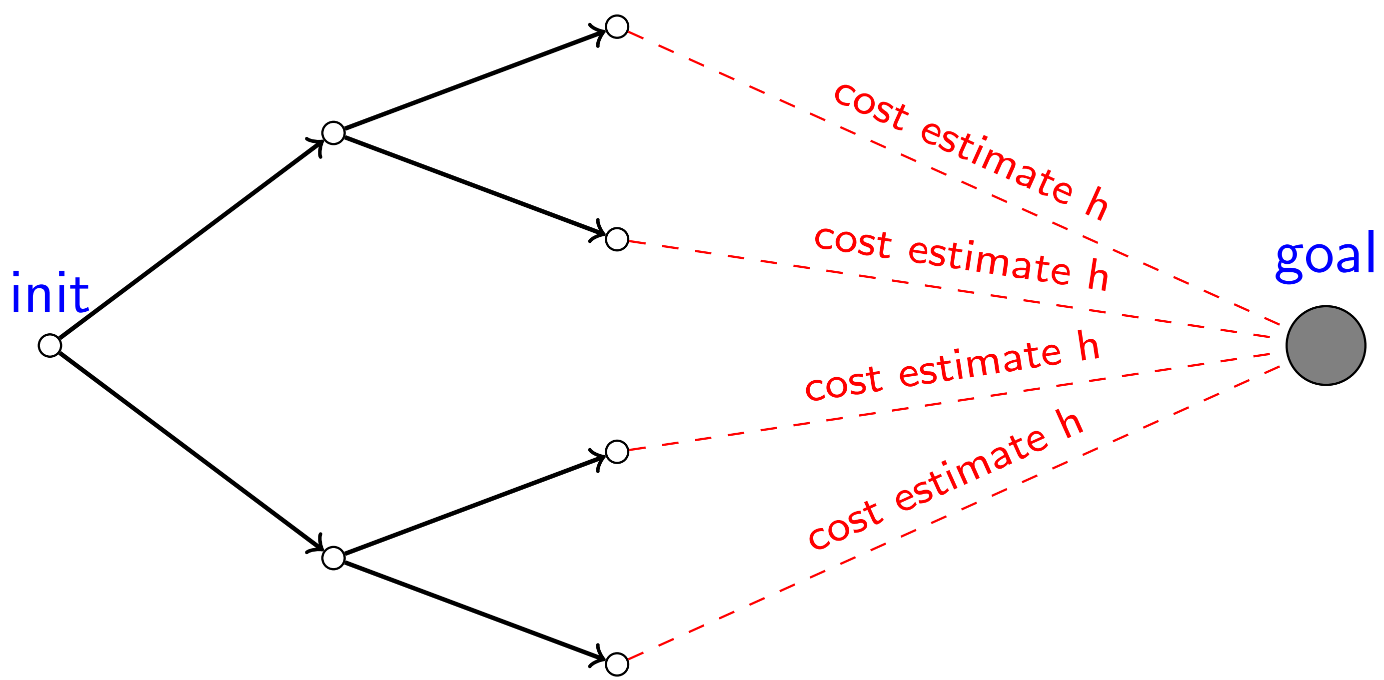
Theorem. *PlanLen is* **PSPACE***-complete.*

Definition (PolyPlanLen). *Given a STRIPS planning task Π and an integer K bounded by a polynomial in the size of Π, does there exists a plan for Π of length at most K?* $\rightarrow$ Corresponds to optimal planning with "small" plans.

Theorem. *PolyPlanLen is* **NP***-complete.*

$\rightarrow$Classical Planning is as hard as SAT if plans are of polynomial length, harder if plans are exponentially long

Reminder: Heuristic Search



→ Heuristic function h estimates the cost of an optimal path from a state s to the goal; search prefers to expand states s with small $h(s)$.

Reminder: Heuristic Functions

Definition (Heuristic Function). *Let* Π *be a planning task with states* S*. A heuristic function, short heuristic, for* Π *is a function* $h : S \mapsto \mathbb{N}_0^+ \cup \{\infty\}$ *so that* $h(s) = 0$ *whenever* s *is a goal state.*

Definition (h^*, Admissibility). *Let* Π *be a planning 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 function* h *for* Π *is admissible if, for all* $s \in S$*, we have* $h(s) \leq h^*(s)$*.*

$\rightarrow$ In all cases, we attempt to approximate $h^*(s)$, the length of an optimal plan for s. Some algorithms guarantee to lower-bound $h^*(s)$.

Reminder: Greedy Best-First Search and A^*

Duplicate elimination omitted for simplicity:

function Greedy Best-First Search $[\mathrm{A}^*]$*(problem)* **returns** a solution, or failure
  node $\leftarrow$ a node n with n.*state*=*problem.InitialState*
  frontier $\leftarrow$ a priority queue ordered by ascending h $[g+h]$, only element n
  loop do
    if *Empty?(frontier)* **then return** failure
    $n \leftarrow$ *Pop(frontier)*
    if *problem.GoalTest*(n.*State*) **then return** *Solution*(n)
    for each *action* a **in** *problem.Actions*(n.*State*) **do**
      $n' \leftarrow$ *ChildNode*(*problem*,n,a)
      Insert(n', $h(n')$ $[g(n')+h(n')]$, *frontier*)

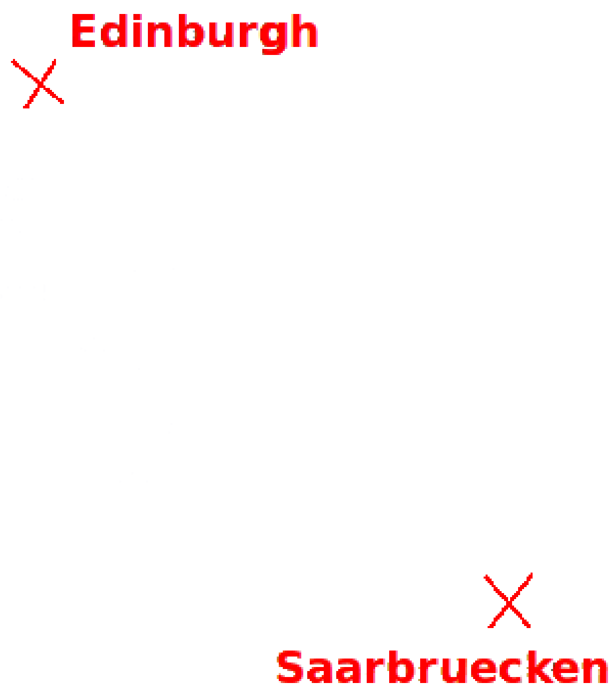
→ Is Greedy Best-First Search optimal? No $\Longrightarrow$ *satisficing* planning.

→ Is A^* optimal? Yes, but only if h is admissible $\Longrightarrow$ *optimal* planning, **with such** h.

Heuristic Functions from Relaxed Problems

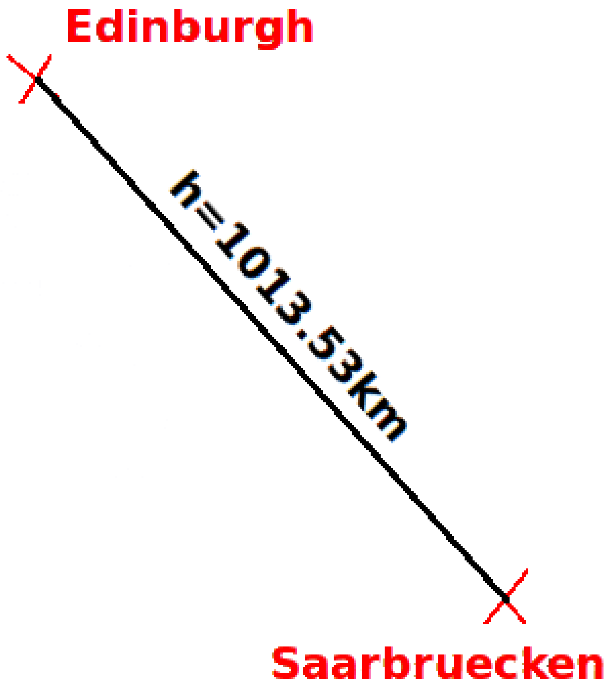
Problem Π: Find a route from Saarbruecken To Edinburgh.

Heuristic Functions from Relaxed Problems



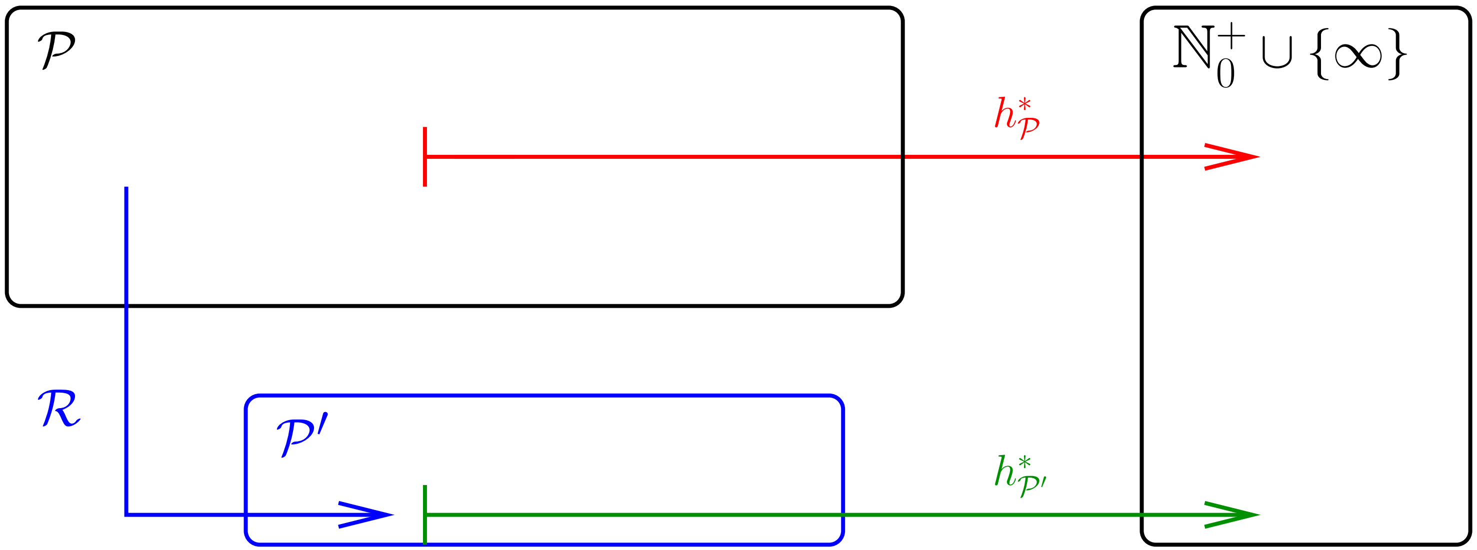
Relaxed Problem Π': Throw away the map.

Heuristic Functions from Relaxed Problems



Heuristic function h: Straight line distance.

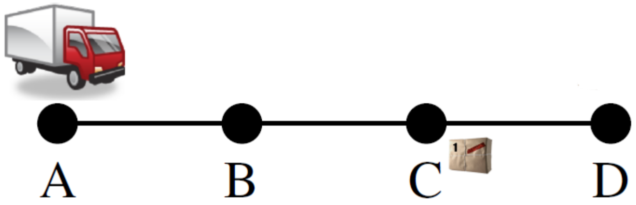
How to Relax



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

How to Relax in Planning? (A Reminder!)

Example: "Logistics"



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

Example Delete-Relaxation: Drop the delete effects.

Search + Inference

In Chapter 7:

DPLL is an example of a successful algorithmic pattern:
Search + Inference

- DPLL $\approx$ Search = Backtracking, with Inference() = unit propagation.
- Unit propagation is sound: It does not reduce the set of solutions.

In Planning:

- Search = A^* or GBFS
- Inference = Heuristic function

FDR Representation

Finite-domain representation (c.f. next slide) is an alternative representation for planning tasks with multi-valued variables.

- $at(Adelaide), at(Sydney), \dots$
- We know that the truck is always in exactly one location

$\Rightarrow$ Variable "at" which may take values Adelaide, Sydney, etc.

$\rightarrow$Both representations are equivalent, but FDR is more convenient when analyzing abstractions and pattern databases so we will use it in this lecture

FDR Planning: Syntax

Definition (FDR Planning Task). *A finite-domain representation planning task, short FDR planning task, is a 5-tuple $\Pi = (V, A, c, I, G)$ where:*

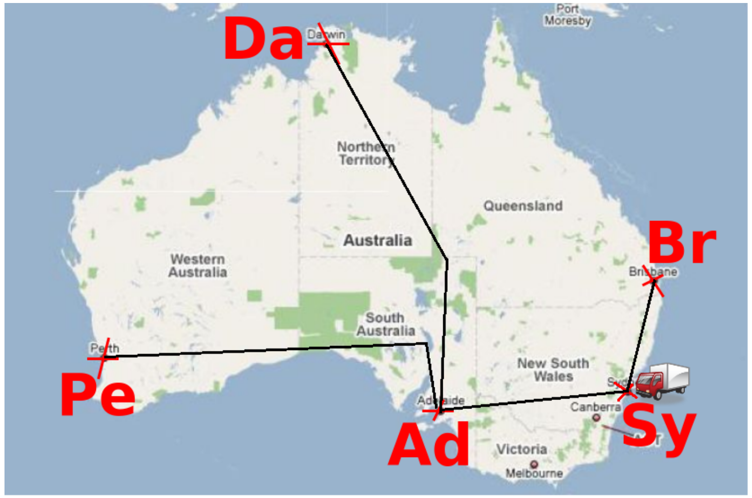
- *V is a finite set of state variables, each $v \in V$ with a finite domain D_v. We refer to (partial) functions on V, mapping each $v \in V$ into a member of D_v, as (partial) variable assignments.*
- *A is a finite set of actions; each $a \in A$ is a pair (pre_a, eff_a) of partial variable assignments referred to as the action's precondition and effects.*
- *$c : A \mapsto \mathbb{R}_0^+$ is the cost function.*
- *I is a complete variable assignment called the initial state.*
- *G is a partial variable assignment called the goal.*

We say that Π has unit costs if, for all $a \in A$, $c(a) = 1$.

→ In FDR, a (partial) variable assignment represents a state in I, a condition in pre_a and G, and an effect instruction in eff_a.

Notation: Pairs (v, d) are facts, also written $v = d$. We identify partial variable assignments p with fact sets. We write $V[p] := \{v \in V \mid p(v) \text{ is defined}\}$.

FDR Encoding of "TSP"

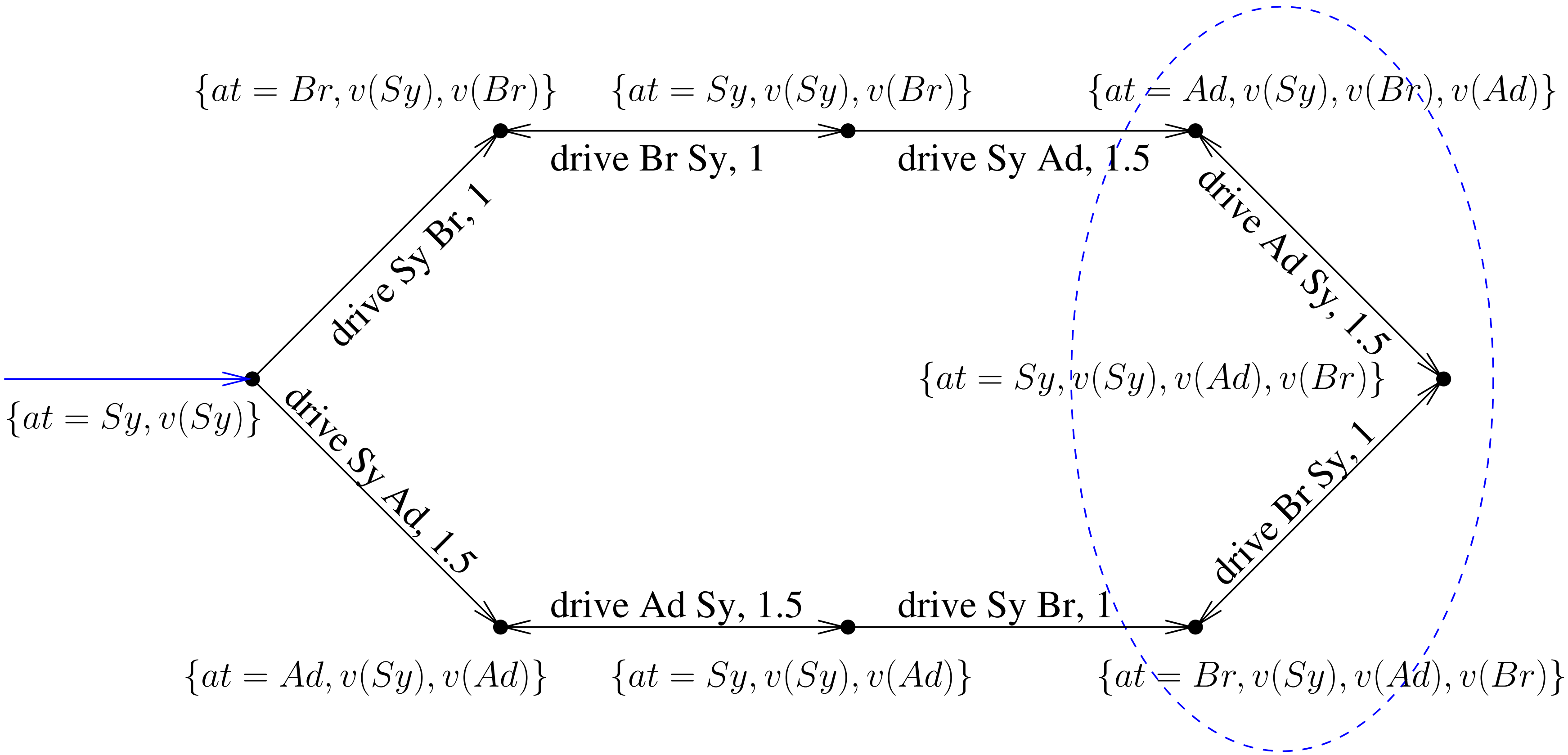


- Variables V: $at : \{Sydney, Adelaide, Brisbane, Perth, Darwin\}$; $visited(x) : \{T, F\}$ for $x \in \{Sydney, Adelaide, Brisbane, Perth, Darwin\}$.
- Initial state I: $at = Sydney, visited(Sydney) = T, visited(x) = F$ for $x \neq Sydney$.
- Goal G: $at = Sydney, visited(x) = T$ for all x.
- Actions $a \in A$: $drive(x, y)$ where x, y have a road.
  Precondition pre_a: $at = x$.
  Effect eff_a: $at = y, visited(y) = T$.
- Cost function c:

$$c(drive(x,y)) = \begin{cases} 1 & \{x,y\} = \{Sydney, Brisbane\} \\ 1.5 & \{x,y\} = \{Sydney, Adelaide\} \\ 3.5 & \{x,y\} = \{Adelaide, Perth\} \\ 4 & \{x,y\} = \{Adelaide, Darwin\} \end{cases}$$

FDR Encoding of Simplified "TSP": State Space

(using "$v(x)$" as shorthand for $visited(x) = T$)



$\rightarrow$ This is only the reachable part of the state space: E.g., Θ_Π also includes the state $\{at = Sy, v(Br)\}$. (But neither $\{v(Sy)\}$ nor $\{at = Sy, at = Br\}$, compare slide 14.)

The 15-Puzzle

9	2	12	6
5	7	14	13
3	4	1	11
15	10	8	

—

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

$\rightarrow$ Abstractions, in the context of AI, were first introduced in the form of pattern database heuristics for the 15-Puzzle.

How to apply the idea of relaxation to solve this puzzle?

Pattern Databases in a Nutshell

9	2	12	6
5	7	14	13
3	4	1	11
15	10	8	

—

	2		6
5	7		
3	4	1	

"Abstract the planning task by choosing a subset P of variables (the pattern), and ignoring the values of all other variables."

Concrete vs. Abstract State Space

9	2	12	6
5	7	14	13
3	4	1	11
15	10	8	

—

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Concrete State Space: $16^{16} \approx 1.8 * 10^{19}$ states.

Concrete vs. Abstract State Space

	2		6
5	7		
3	4	1	

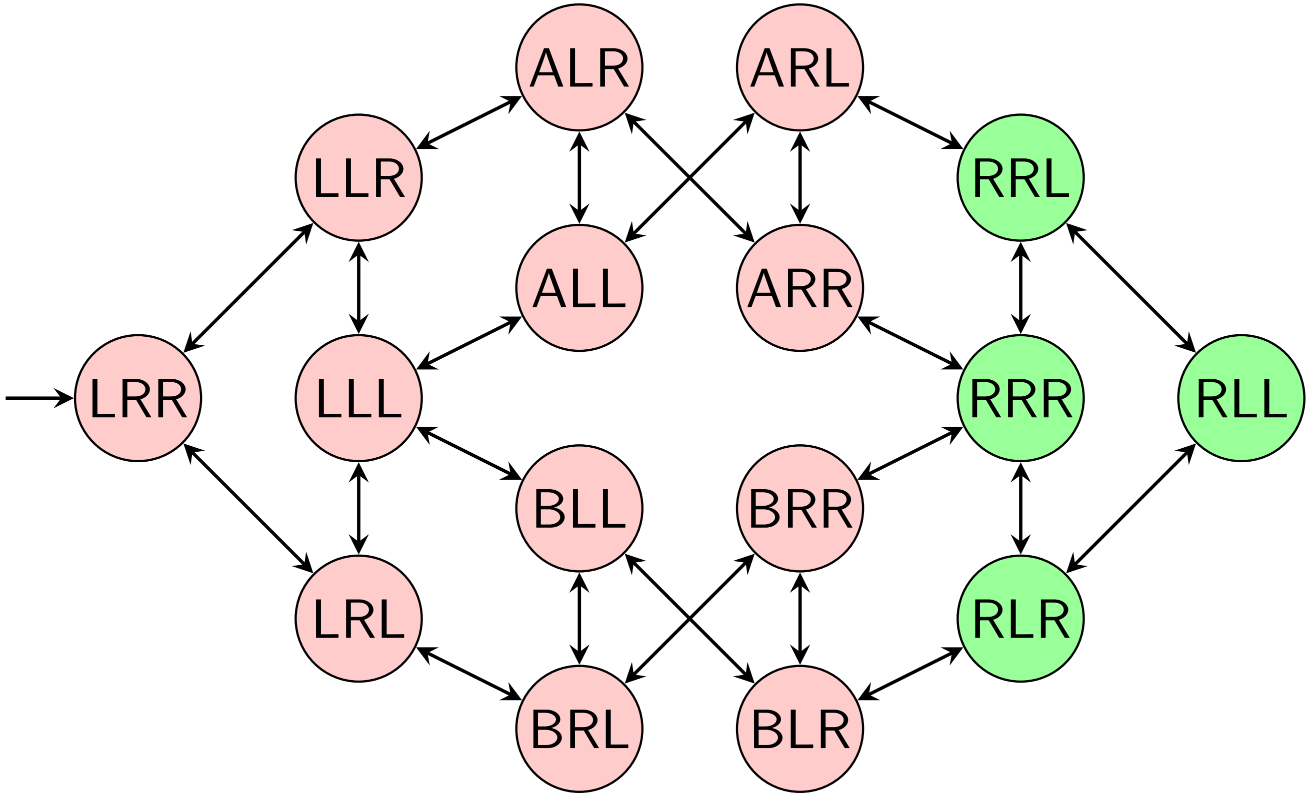
—

1	2	3	4
5	6	7	

Abstract State Space: $16^8 \approx 4.2 * 10^9$ states.

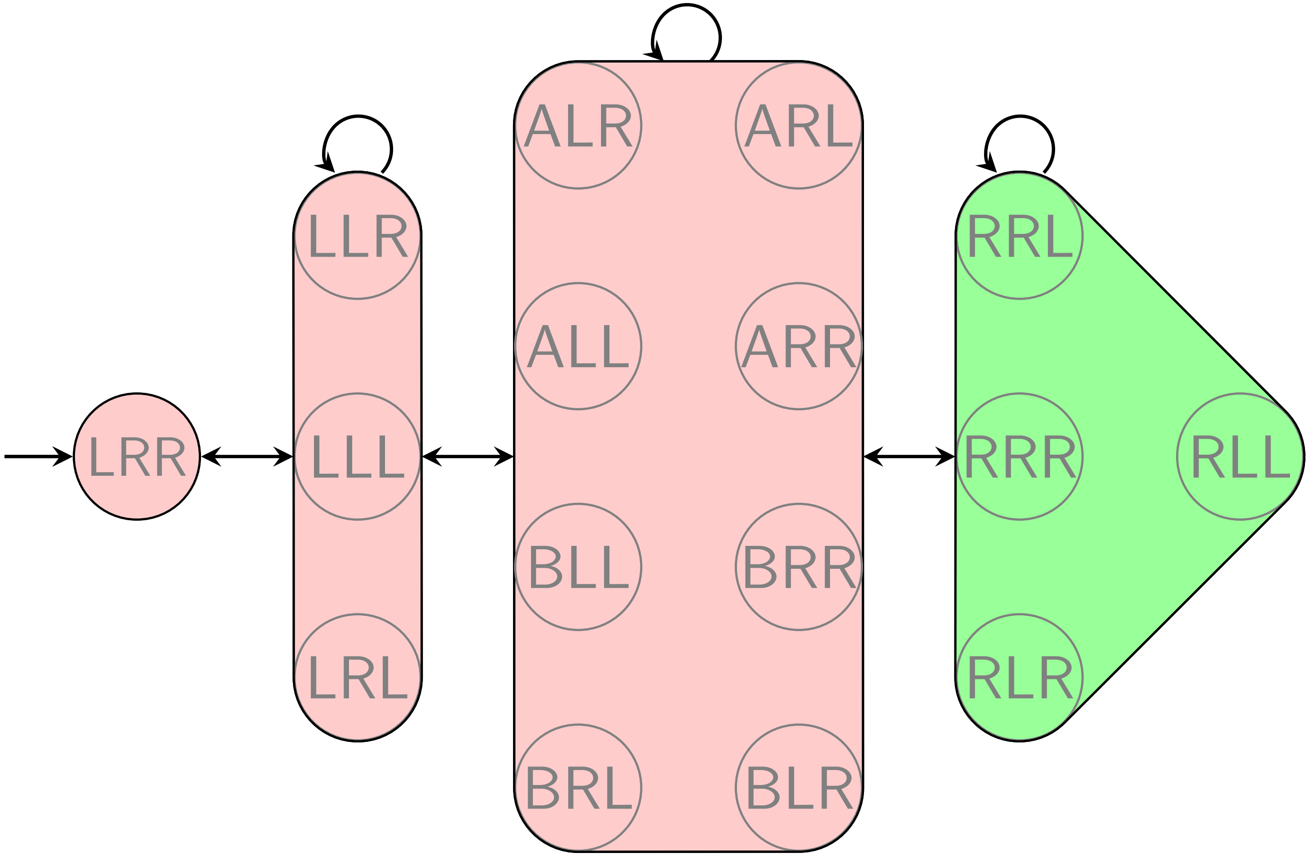
Abstractions in a Nutshell: Example

Concrete transition system: (of “Logistics mal anders”, see later)



Abstractions in a Nutshell: Example

Abstract transition system: (of “Logistics mal anders”, see later)



Abstractions in a Nutshell: Wrap-Up

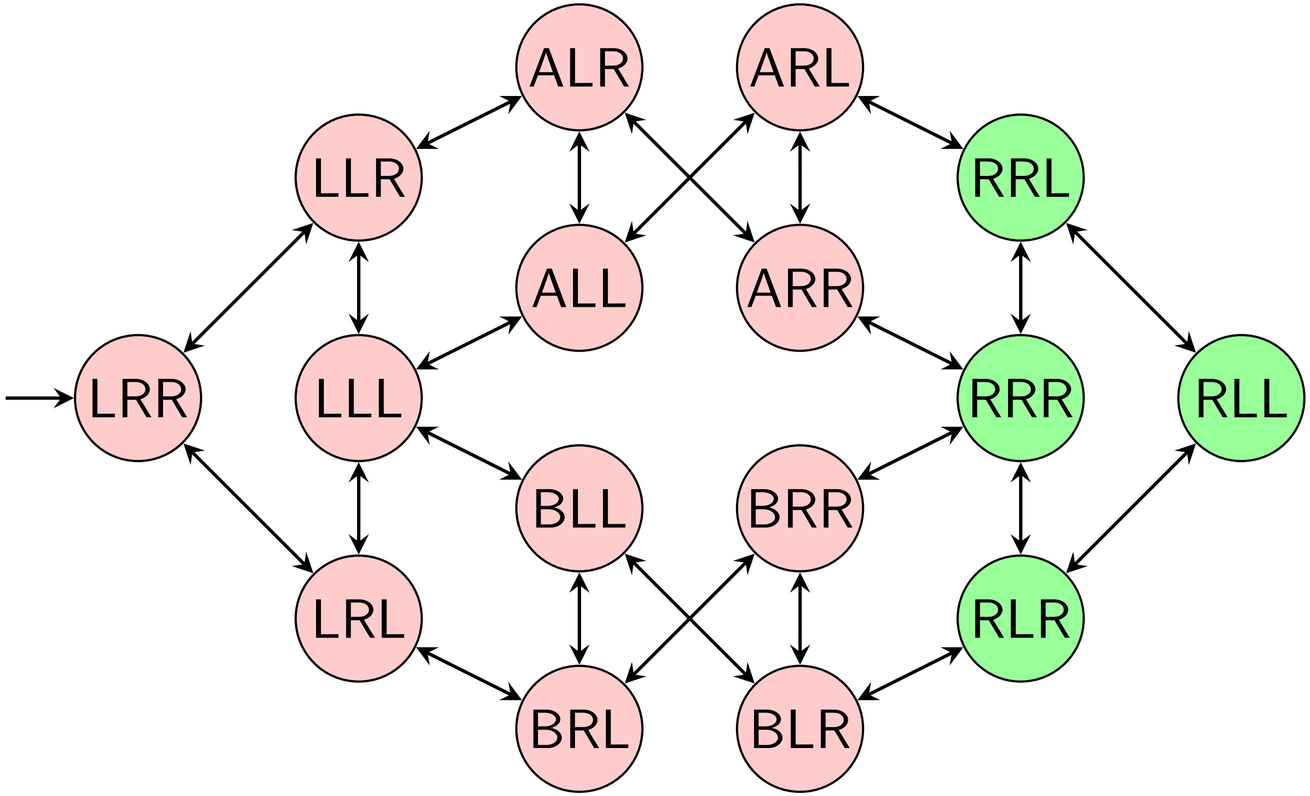
→ Abstracting a transition system means dropping some distinctions between states, while preserving all transitions and goal states.

- An abstraction of a transition system Θ is defined by a function α (the abstraction mapping), mapping states to abstract states (also block states).
- If α maps states s and t to the same abstract state, then s and t are not distinguished anymore (they are equivalent under α).
- The abstract transition system Θ^α on the image of α is defined by homomorphically mapping over all goal states and transitions from Θ, and thus preserving all solutions.
- The abstract remaining cost, i.e., remaining cost in Θ^α, is an estimate h^α for remaining cost in Θ. As we preserve all solutions, h^α is admissible.

Our Agenda for This Chapter

2. **Abstraction Basics:** Formal definition of abstractions and their associated structures; proving their basic properties.
3. **Practical vs. Pathological Abstractions:** We briefly illuminate basic practical issues, through a number of examples illustrating "how not to do it".
4. **Pattern Databases:** How to implement PDB heuristics (namely, via a "pattern database").

The State Space of “Logistics mal anders”



- State $p = x$, $t_A = y$, $t_B = z$ is depicted as xyz.
- Transition labels not shown. For example, the transition from LLL to ALL has the label $pickup(A, L)$.

Abstractions

Definition (Abstraction). *Let $\Theta = (S, L, c, T, I, S^G)$ be a transition system. An abstraction of Θ is a surjective function $\alpha : S \mapsto S^\alpha$, also referred to as the abstraction mapping. The abstract state space induced by α, written Θ^α, is the transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, I^\alpha, S^{\alpha G})$ defined by:*

(i) $I^\alpha = \alpha(I)$.

(ii) $S^{\alpha G} = \{\alpha(s) \mid s \in S^G\}$. */* preserve goal states */*

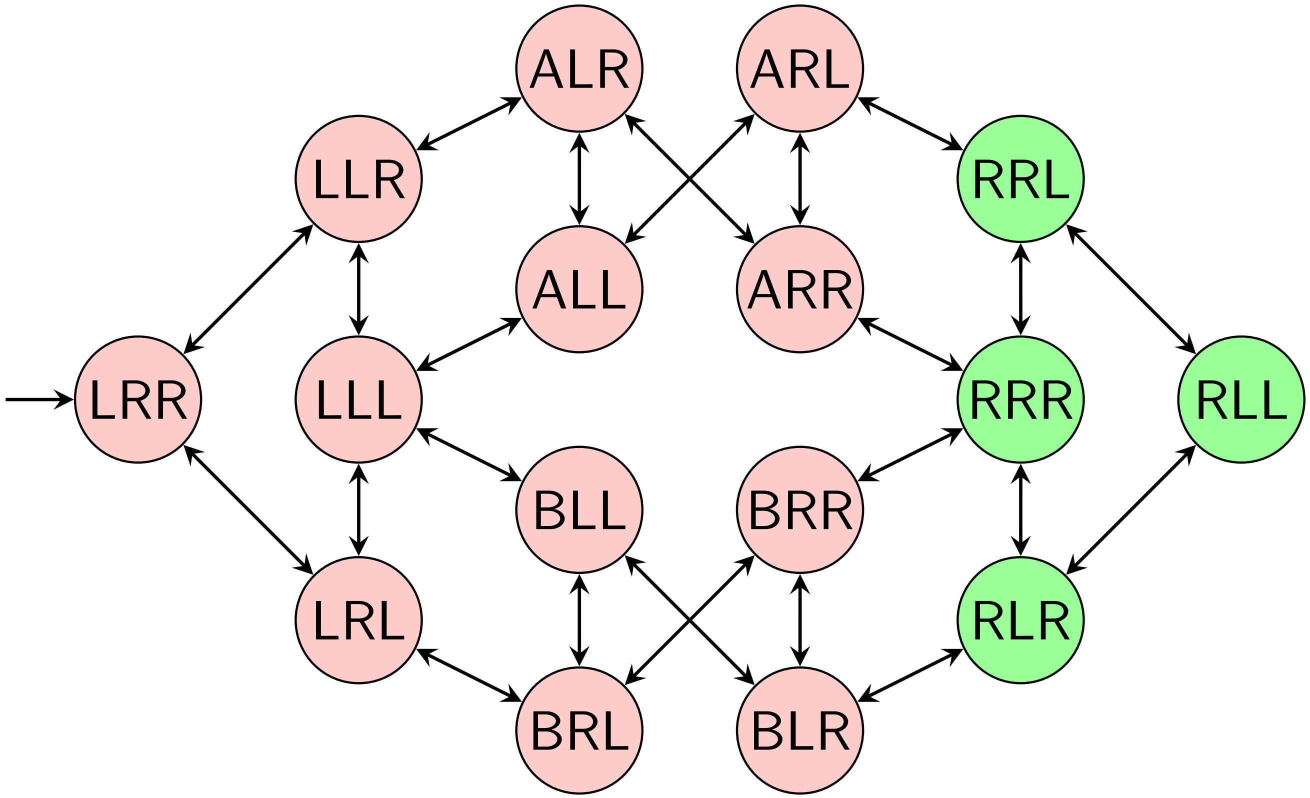
(iii) $T^\alpha = \{(\alpha(s), l, \alpha(t)) \mid (s, l, t) \in T\}$. */* preserve transitions */*

The size of the abstraction is the number $|S^\alpha|$ of abstract states.

$\rightarrow$ Θ is called the concrete state space. Similarly: concrete/abstract transition system, concrete/abstract transition, etc.

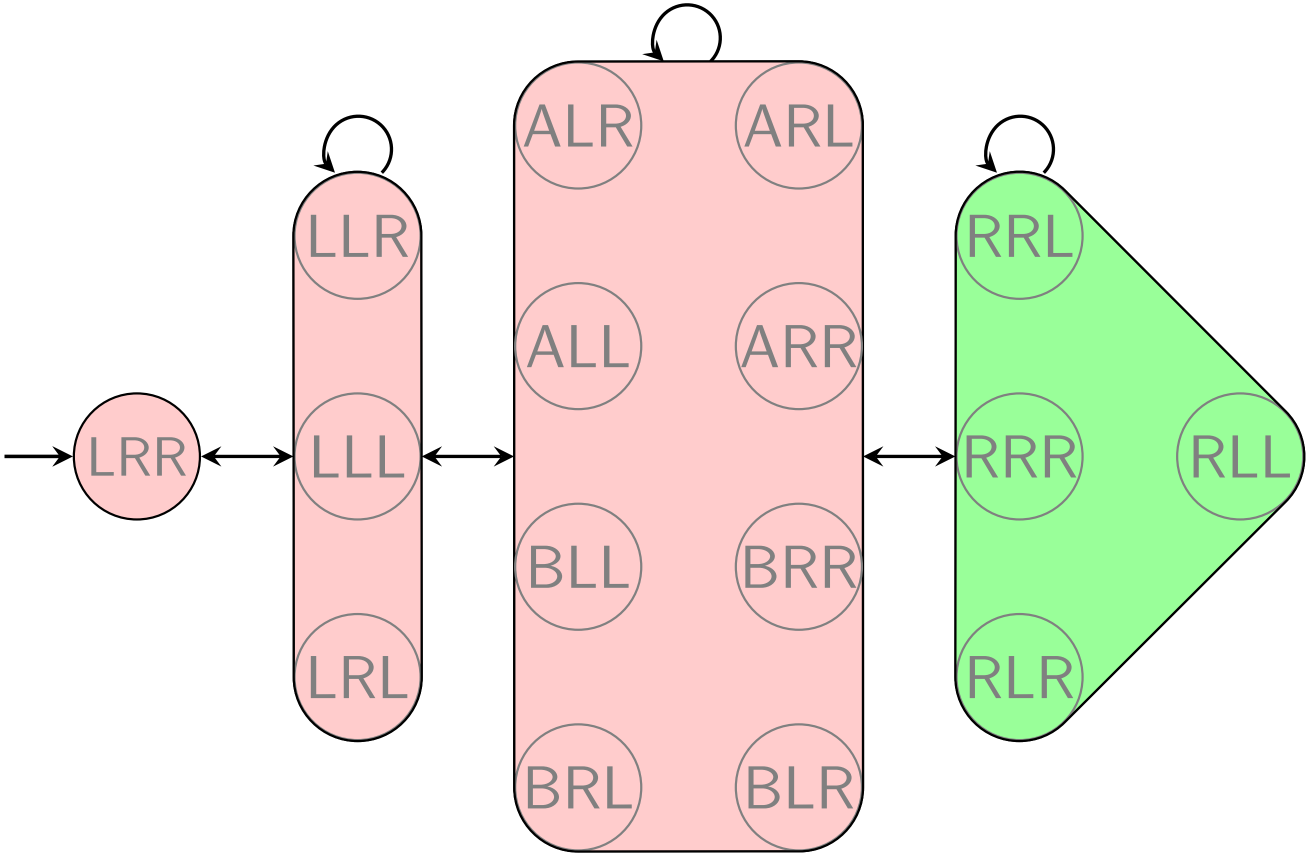
Abstractions: “Logistics mal anders”

Concrete transition system:



Abstractions: "Logistics mal anders"

Abstract transition system:



→ A transition between concrete states is "spurious" if it exists in the abstract but not in the concrete state space. Example here? We can go in a single step from LRR to LLL.

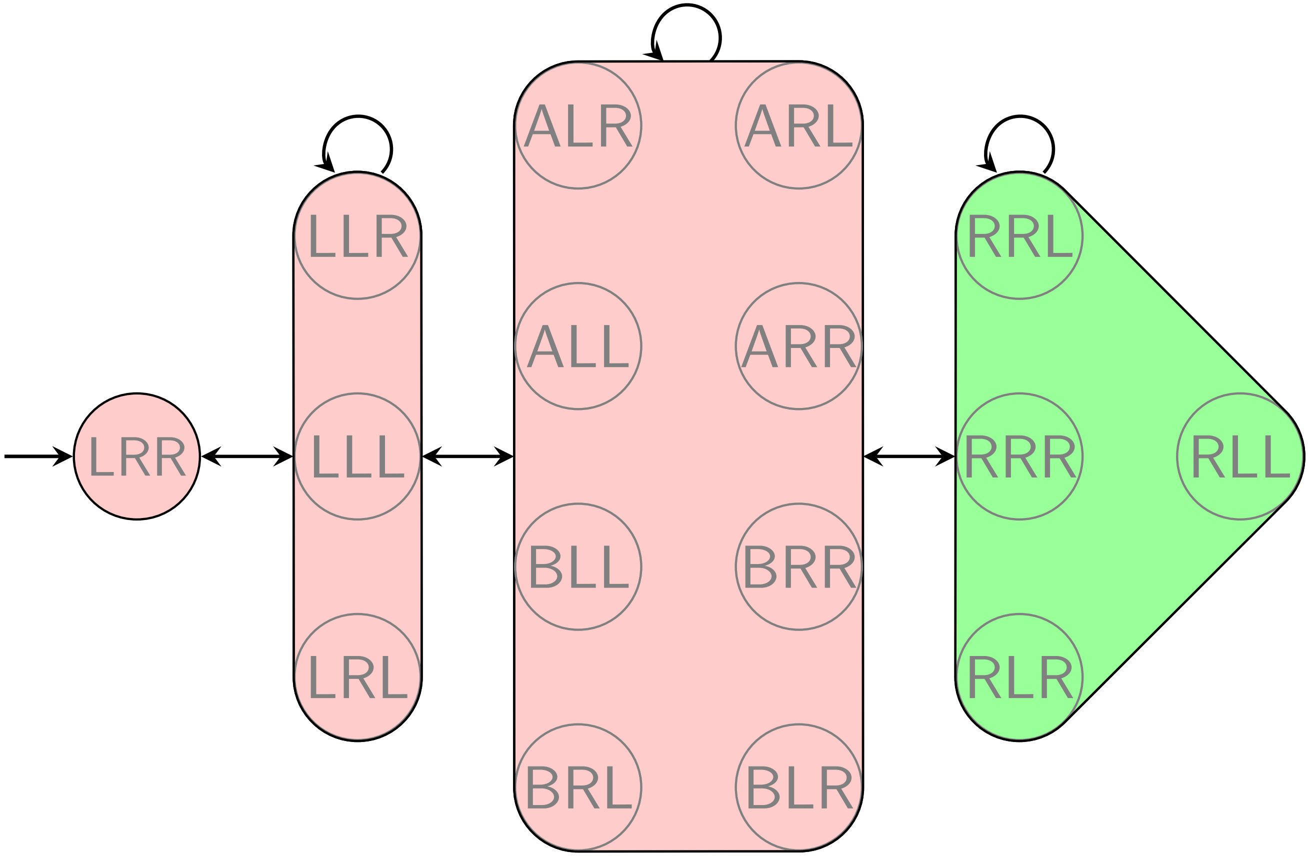
Abstraction Heuristics

Definition (Abstraction Heuristic). *Let $\Theta = (S, L, c, T, I, S^G)$ be a transition system, and let α be an abstraction of Θ. The abstraction heuristic induced by α, written h^α, is the heuristic function $h^\alpha : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$ which maps each state $s \in S$ to $h^*_{\Theta^\alpha}(\alpha(s))$, i.e., to the remaining cost of $\alpha(s)$ in Θ^α.*

→ The abstract remaining cost (remaining cost in Θ^α) is used as the heuristic estimate for remaining cost in Θ.

→ $h^\alpha(s) = \infty$ if no goal state of Θ^α is reachable from $\alpha(s)$.

Abstraction Heuristics: “Logistics mal anders”



$$h^{\alpha}(\{p = L, t_A = R, t_B = R\}) = 3 \neq h^{*}(\{p = L, t_A = R, t_B = R\}) = 4$$

Abstraction Heuristics: Properties

Proposition (h^α is Admissible). *Let Θ be a transition system, and let α be an abstraction of Θ. Then h^α is consistent and goal-aware, and thus also admissible and safe.*

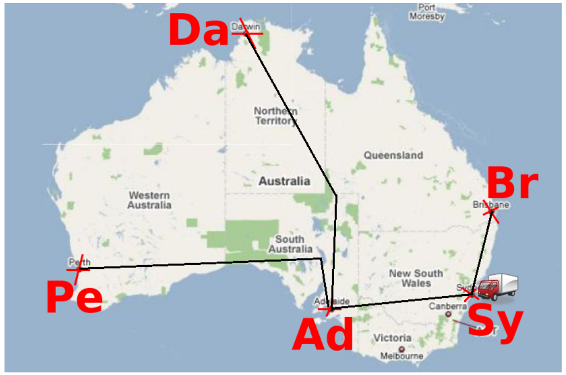
Proof Idea: Any plan in the original transition system is a valid plan in the abstract transition system.

Proof. (for reference) Let $\Theta = (S, L, c, T, I, S^G)$ and $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, I^\alpha, S^{\alpha G})$.

For goal-awareness, we need to show that $h^\alpha(s) = 0$ for all $s \in S^G$. So let $s \in S^G$. Then $\alpha(s) \in S^{\alpha G}$ by definition of abstractions, and hence $h^\alpha(s) = h^*_{\Theta^\alpha}(\alpha(s)) = 0$.

For consistency, we need to show that whenever $(s, a, t) \in T$, $h^\alpha(s) \leq h^\alpha(t) + c(a)$. By definition, $h^\alpha(s) = h^*_{\Theta^\alpha}(\alpha(s))$ and $h^\alpha(t) = h^*_{\Theta^\alpha}(\alpha(t))$, so we need to show that $h^*_{\Theta^\alpha}(\alpha(s)) \leq h^*_{\Theta^\alpha}(\alpha(t)) + c(a)$. Since (s, a, t) is a concrete transition, by definition of abstractions we have an abstract transition $(\alpha(s), a, \alpha(t))$ in Θ^α. But then, $h^*_{\Theta^\alpha}(\alpha(s)) \leq h^*_{\Theta^\alpha}(\alpha(t)) + c(a)$ holds simply because h^* is consistent. (In our notation here: $h^*_{\Theta^\alpha}$ is consistent in Θ^α).

Questionnaire



- Variables: $at : \{Sy, Ad, Br, Pe, Ad\}$; $v(x) : \{T, F\}$ for $x \in \{Sy, Ad, Br, Pe, Ad\}$.
- Actions: $drive(x, y)$ where x, y have a road.
- Costs: $Sy \leftrightarrow Br : 1$, $Sy \leftrightarrow Ad : 1.5$, $Ad \leftrightarrow Pe : 3.5$, $Ad \leftrightarrow Da : 4$.
- Initial state: $at = Sy, v(Sy) = T, v(x) = F$ for $x \neq Sy$.
- Goal: $at = Sy, v(x) = T$ for all x.

Question!

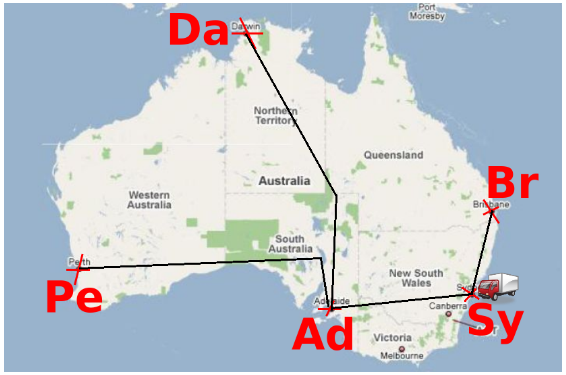
Say α projects this planning task onto $\{at, v(Pe), v(Da)\}$, i.e., $\alpha(s) = \alpha(t)$ iff they agree on these variables. What is $h^\alpha(I)$?

(A): 10 (B): 12.5

(C): 18 (D): 20

$\rightarrow$ In the abstract state space induced by α, any solution must visit Perth and Darwin, then return to Sydney. The optimal sequence doing so has cost 18, so (C) is correct.

Questionnaire, ctd.



- Variables: $at : \{Sy, Ad, Br, Pe, Ad\}$; $v(x) : \{T, F\}$ for $x \in \{Sy, Ad, Br, Pe, Ad\}$.
- Actions: $drive(x, y)$ where x, y have a road.
- Costs: $Sy \leftrightarrow Br : 1$, $Sy \leftrightarrow Ad : 1.5$, $Ad \leftrightarrow Pe : 3.5$, $Ad \leftrightarrow Da : 4$.
- Initial state: $at = Sy, v(Sy) = T, v(x) = F$ for $x \neq Sy$.
- Goal: $at = Sy, v(x) = T$ for all x.

Question!

Say α projects this task onto $\{v(Pe), v(Da)\}$. What is $h^\alpha(I)$?

(A): 2 (B): 7.5

(C): 12.5 (D): 14

$\rightarrow$ We can drive to Perth and Darwin without achieving the truck precondition. The only actions driving to these cities cost 3.5 respectively 4, so (B) is correct.

Which Abstractions Should We Use in Practice?

Conflicting Objectives

The eternal trade-off between accuracy and efficiency:

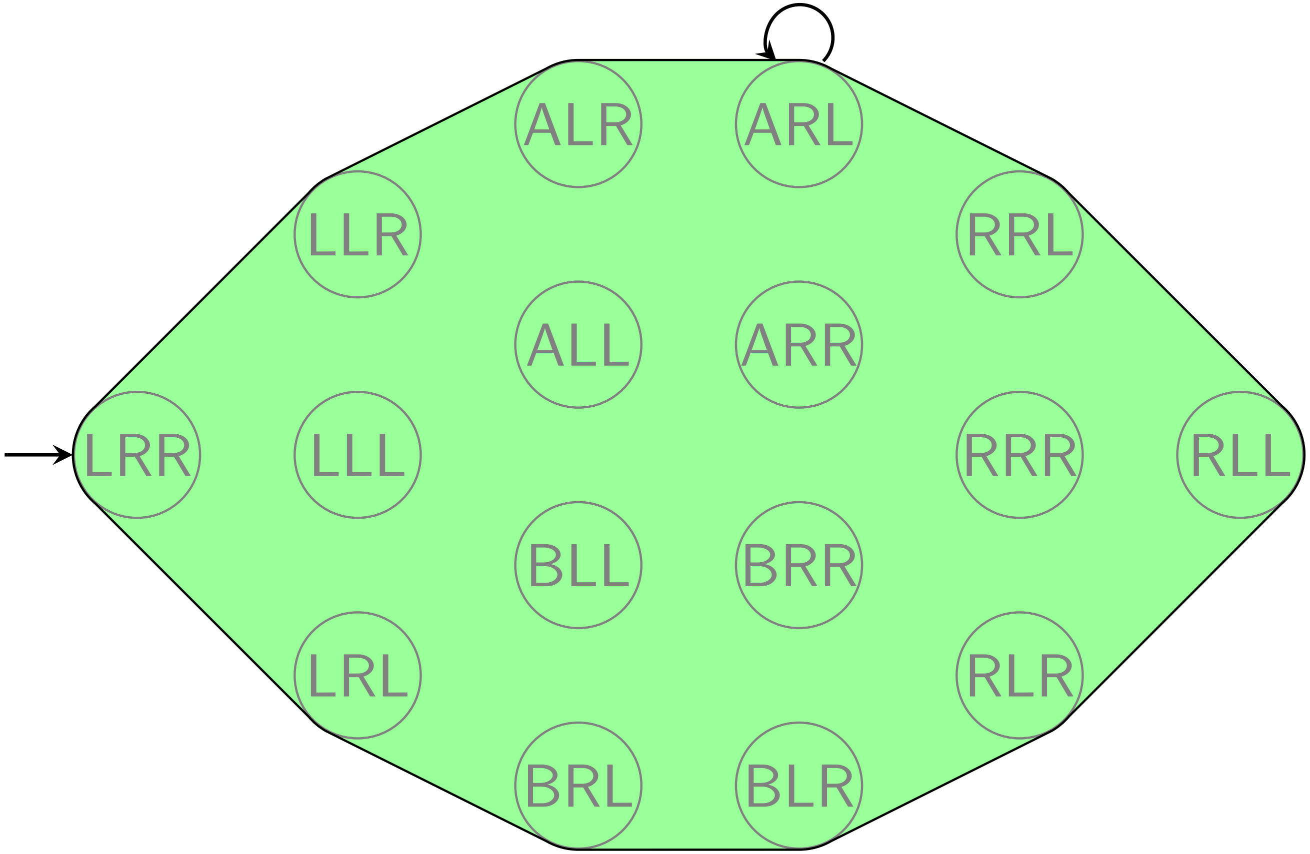
- We want to obtain an informative heuristic.
- We want to obtain a small computational overhead.

$\rightarrow$ The abstraction function α is a very powerful parameter, allowing to travel the whole way between both extremes (see next slides).

$\rightarrow$ What do we mean by "small computational overhead"?

- **Fast computation of α:** For a given state s, the abstract state $\alpha(s)$ must be efficiently computable.
- **Few abstract states:** For a given abstract state $\alpha(s)$, the abstract remaining cost $h^{\alpha}(s) = h^{*}_{\Theta^{\alpha}}(\alpha(s))$ must be efficiently computable.

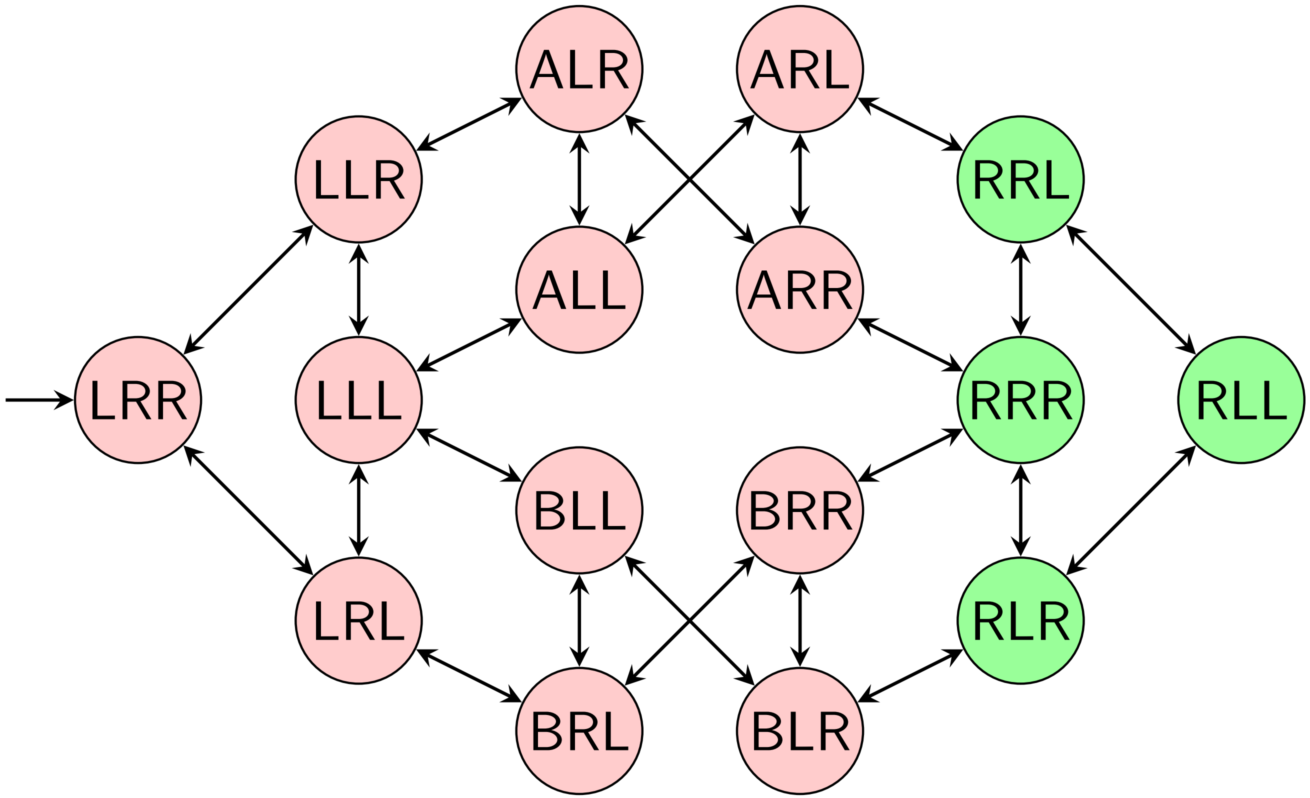
Pathological Case 1: One-State Abstraction



One-state abstraction: $\alpha(s) :=$ const.

+ Trivial to compute α, just one abstract state.
– Completely uninformative h^{α}.

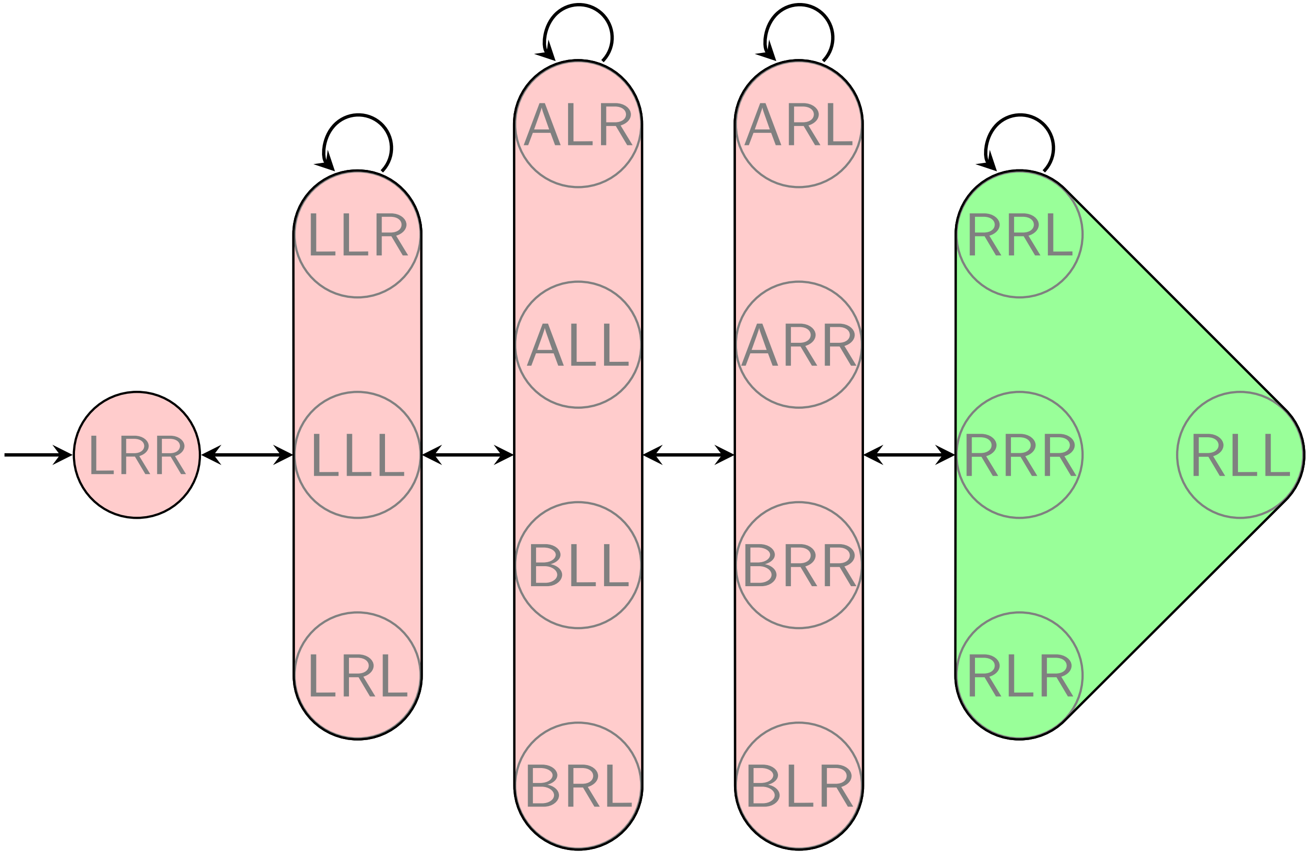
Pathological Case 2: Identity Abstraction



Identity abstraction: $\alpha(s) := s$.

+ $h^\alpha = h^*$, trivial to compute α.
− Abstract state space = concrete state space.

Pathological Case 3: Perfect Abstraction



Perfect abstraction: $\alpha(s) := h^*(s)$.

+ $h^\alpha = h^*$, usually very few abstract states.
– Computing α entails solving the optimal planning problem.

So, How to Obtain *Non*-Pathological Abstractions?

Covered in this course:

- Pattern database heuristics [Culberson and Schaeffer (1998); Edelkamp (2001); Haslum *et al.* (2007)].

Not covered in this course:

- Domain Abstractions, obtained by aggregating values within state variable domains [Hernádvölgyi and Holte (2000)]. Generalizes pattern database heuristics.
- Cartesian Abstractions, where abstract states are characterized by cross-products of state-variable-domain-subsets [Seipp and Helmert (2013)]. Generalizes domain abstractions.
- Merge-and-shrink abstractions: obtained by iteratively applying transformations to a set of transition systems [Dräger *et al.* (2006); Helmert *et al.* (2007); Katz *et al.* (2012); Helmert *et al.* (2014)].
- Structural patterns, where abstractions are implicitly represented [Katz and Domshlak (2008)].

Pattern Database Heuristics

"Pattern database heuristics" $=$ Heuristics induced by a particular class of abstraction mappings, namely projections:

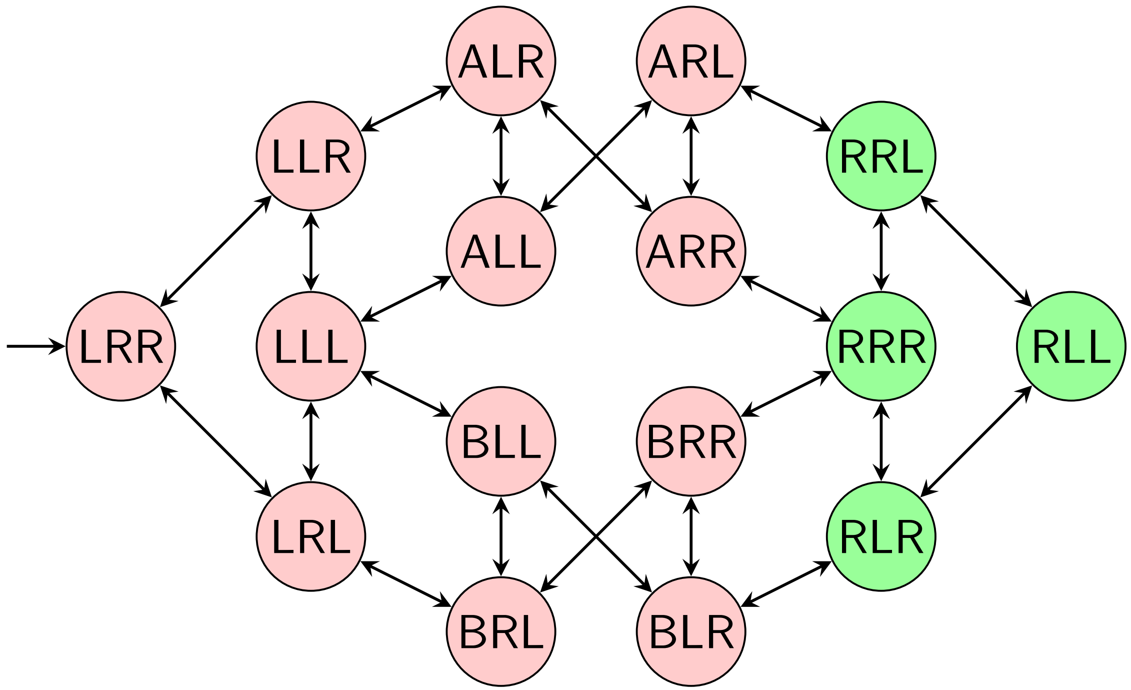
Definition (Projection, PDB Heuristic). *Let $\Pi = (V, A, c, I, G)$ be an FDR planning task with state space $\Theta_\Pi = (S, L, c, T, I, S^G)$, and let $P \subseteq V$. For a partial assignment φ to V, by $\varphi|_P$ we denote the restriction of φ to P.*
Let S^P be the set of variable assignments to P. The projection $\pi_P \colon S \mapsto S^P$ is defined by $\pi_P(s) := s|_P$.

$\rightarrow$ π_P maps two states s_1 and s_2 to the same abstract state iff they agree on all variables in the pattern.

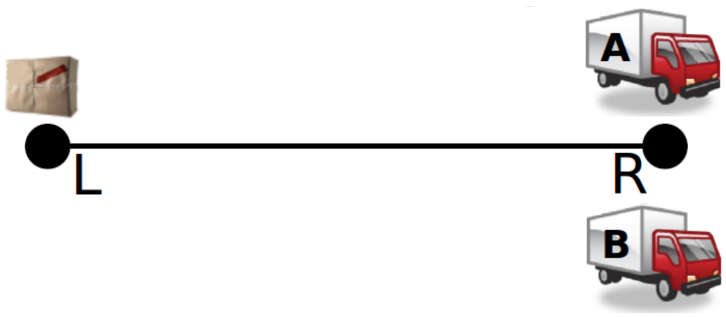
We refer to P as the pattern of π_P. The abstraction heuristic induced by π_P on Θ_Π is called a pattern database heuristic, short PDB heuristic. We write h^P as a short-hand for h^{π_P}, and we write Θ_Π^P or Θ^P as short-hands for $\Theta_\Pi^{\pi_P}$.

- h^P is usually stored in a lookup table called a pattern database (PDB).

"Logistics mal anders": State Space



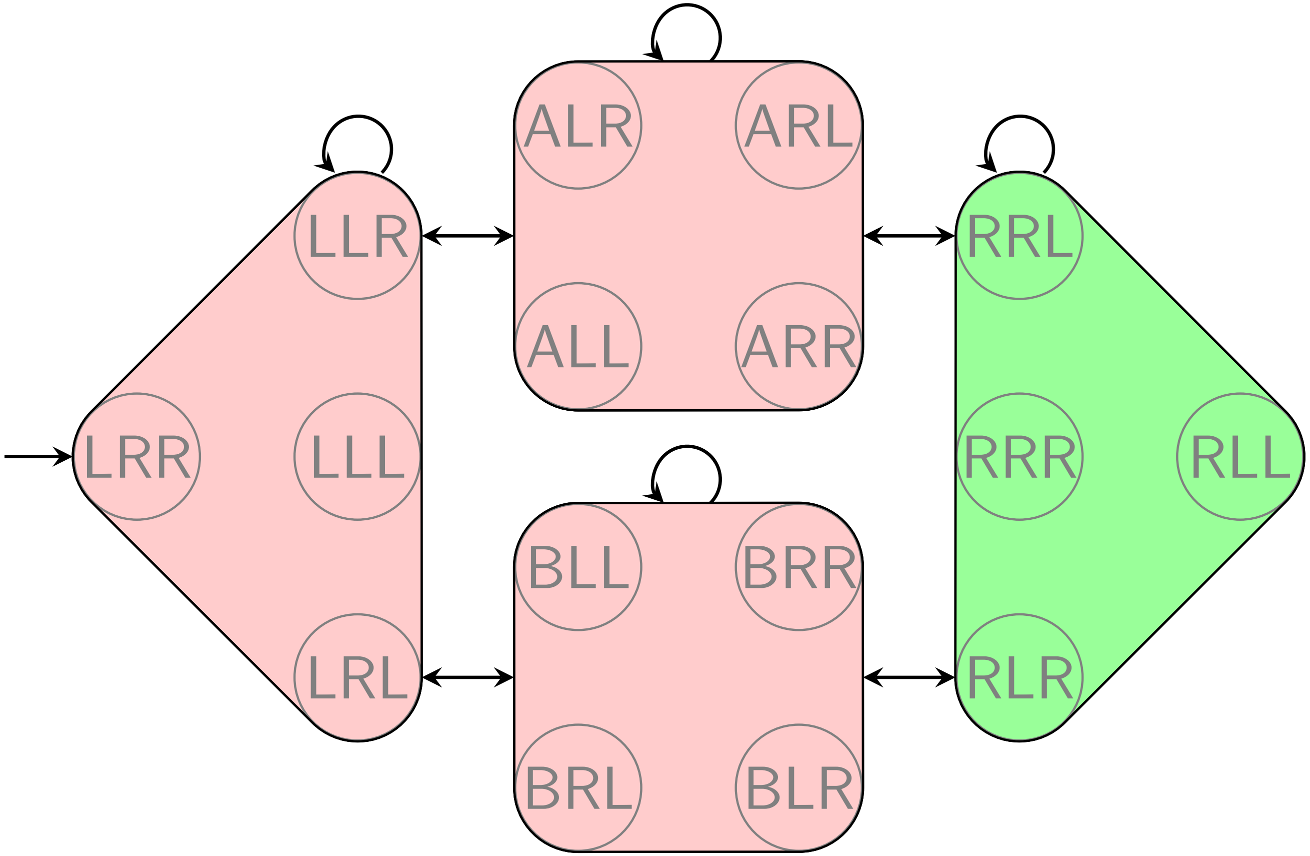
Logistics task with one package, two trucks, two locations:



- State variable package: $\{L, R, A, B\}$.
- State variable truck A: $\{L, R\}$.
- State variable truck B: $\{L, R\}$.

"Logistics mal anders": Projection 1

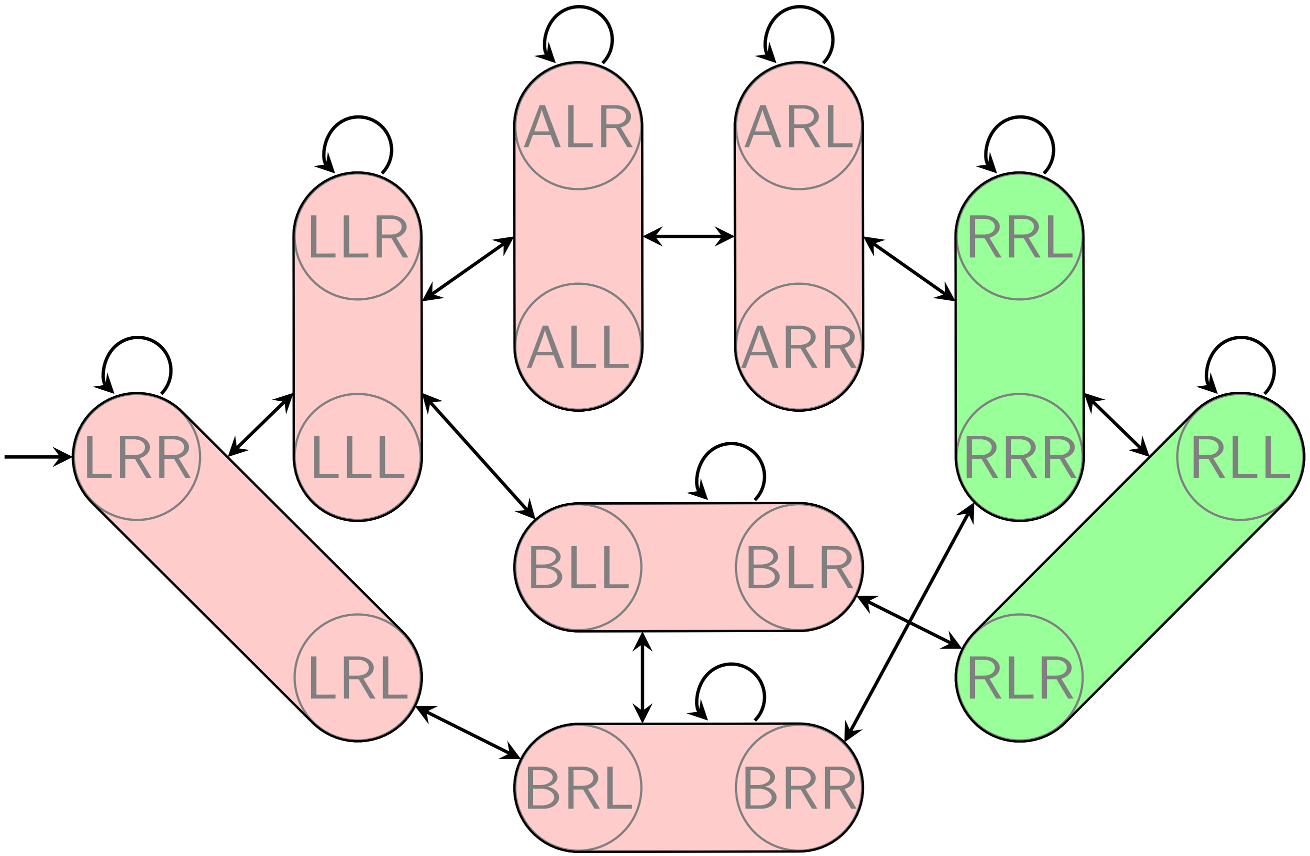
Abstraction induced by $\pi_{\{\text{package}\}}$:



$$h^{\{\text{package}\}}(\text{LRR}) = 2$$

"Logistics mal anders": Projection 2

Abstraction induced by $\pi_{\{\text{package},\text{truck A}\}}$:



$$h^{\{\text{package},\text{truck A}\}}(\text{LRR}) = 2$$

I Give You Pattern, You Give Me Database!

Asssume: You are given a pattern P.

- How do you compute h^P?
- More precisely: How do you compute a data structure that efficiently represents the function $h^P(s)$, for all states s?

Here's how:

1. In a precomputation step, we compute an explicit graph representation for the abstract state space $\Theta_\Pi^{\pi_P}$, and compute the abstract remaining cost for every abstract state.
2. During search, we use the precomputed abstract remaining costs in a lookup step.

(I) Precomputation Step: It's Not That Easy

Let Π be a planning task and P a pattern. Let $\Theta = \Theta_\Pi$ and $\Theta' = \Theta_\Pi^{\pi_P}$.
We want to compute a graph representation of Θ'.

So, what's the issue?

- Θ' is defined through a function on Θ:
  - Each concrete transition induces an abstract transition, each concrete goal state induces an abstract goal state.
- In principle, we can we compute Θ' by iterating over all transitions/goal states of Θ. BUT:
  - This would take time $\Omega(||\Theta||)$.
  - Which comes down to solving the original (concrete, not abstract) planning task in the first place, using blind search.

$\rightarrow$ We need a way of computing Θ' in time polynomial in $||\Pi||$ and $||\Theta'||$.

(I) Precomputation Step: Here's How To

Definition (Syntactic Projection). *Let* $\Pi = (V, A, c, I, G)$ *be an FDR planning task, and let* $P \subseteq V$. *The syntactic projection of* Π *to* P *is the FDR planning task* $\Pi|_P = (P, A|_P, c, I|_P, G|_P)$ *where* $A|_P := \{a|_P \mid a \in A\}$ *with* $pre_{a|_P} := (pre_a)|_P$ *and* $\mathit{eff}_{a|_P} := (\mathit{eff}_a)|_P$.

→ $\Pi|_P$ removes the variables outside P from all constructs in the planning task description Π.

Theorem (Syntactic Projection is Equivalent to Projection). *Let* Π *be an FDR planning task, and let* $P \subseteq V$. *Then* $\Theta_{(\Pi|_P)}$ *is identical to* $\Theta^{\pi_P}_{\Pi}$ *except that labels* a *in the latter become labels* $a|_P$ *in the former.*

Proof. Easy from definition.

→ The state space of the syntactic projection is (modulo label renaming) the same as the abstract state space of the projection.

(I) Precomputation Step: Here's How To, ctd.

Using the Theorem on the previous slide, we can compute pattern databases for FDR tasks Π and patterns P:

Computing Pattern Databases

def compute-PDB(Π, P):
  $\Pi' := \Pi|_P$.
  Compute $\Theta' := \Theta_{\Pi'}$ by a complete forward search (e.g., breadth-first).
  In the explicit graph Θ', add a new node x with a 0-cost incoming edge from every goal node
  Run Dijkstra starting from x and traversing edges backwards, to compute all cheapest paths to x and thus the remaining costs $h^*_{\Theta'}$ in Θ'
  PDB $:=$ a table containing all remaining costs in Θ'
  return *PDB*

$\rightarrow$ This algorithm runs in time and space polynomial in $\|\Pi\| + \|\Theta'\|$.

(II) Lookup Step: Overview

Basic observations and method:

- During search, we do not need the actual abstract state space (transitions etc): The PDB is the only piece of information necessary to represent h^P.

  $\rightarrow$ We can throw away the abstract state space Θ' once the PDB is computed.

  $\rightarrow$ Space requirement for the PDB heuristic during search is linear in number of abstract states S': *PDB* has one table entry for each abstract state.

- Design a perfect hash function mapping projected states $s|_P$ to numbers in the range $\{0, \dots, |S'| - 1\}$.

  $\rightarrow$ Index *PDB* by these hash values. Given a state s during search, to compute $h^P(s)$, map $\pi_P(s) = s|_P$ to its hash value and lookup the table entry of *PDB*.

(II) Lookup Step: Here's How To

Perfect hash function $\approx$ numeral system over variable domains:

- Let $P = \{v_1, \dots, v_k\}$ be the pattern.
- Assume wlog that all variable domains are natural numbers counted from 0, i.e., $D_v = \{0, 1, \dots, |D_v| - 1\}$.
- For all $i \in \{1, \dots, k\}$, we precompute $N_i := \prod_{j=1}^{i-1} |D_{v_j}|$.

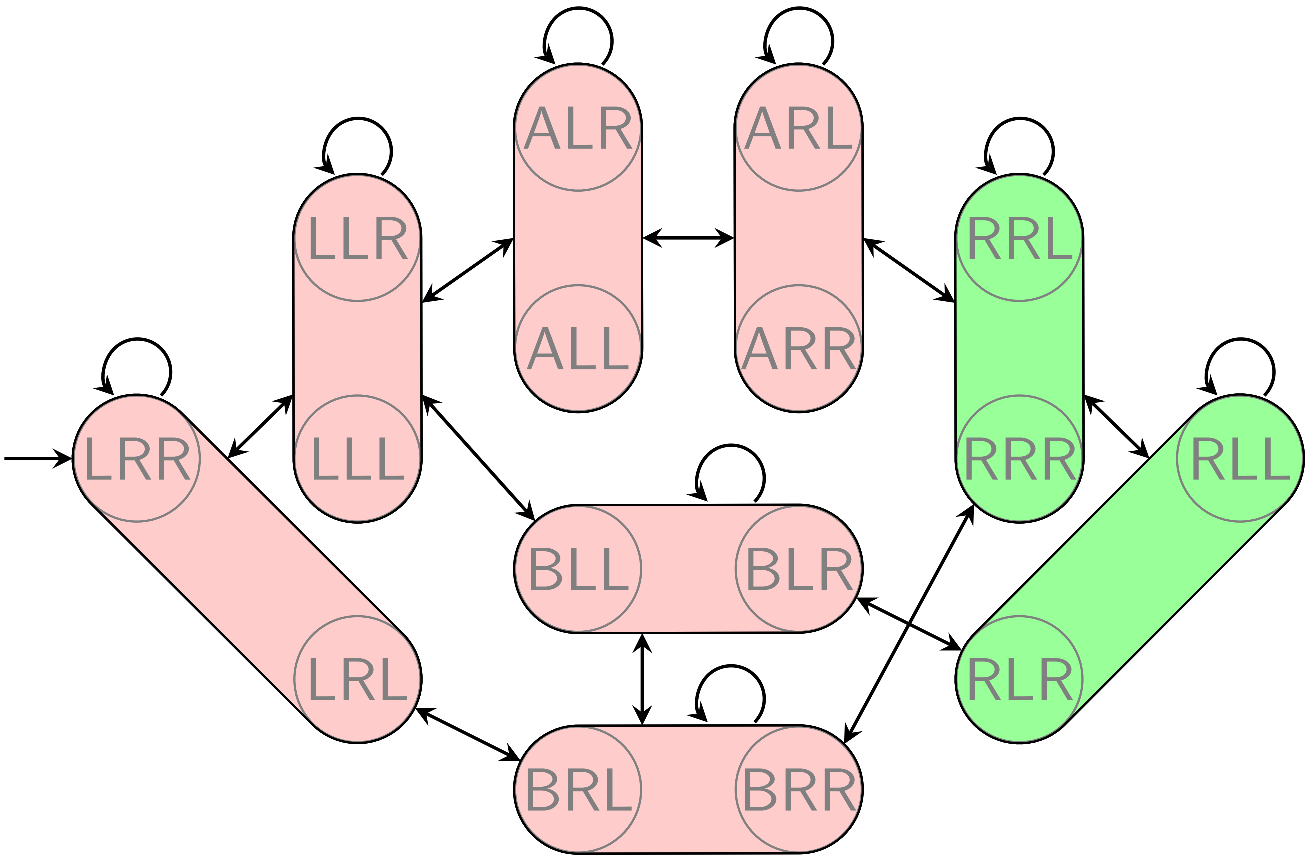
Looking Up a Pattern Database Heuristic Value

def PDB-heuristic(s):
    index $:= \sum_{i=1}^{k} N_i s(v_i)$
    return *PDB*[*index*]

Note: This lookup runs in time and space $O(k)$. This is *very* fast. For comparison, delete-relaxation heuristics need time $O(\|\Pi\|)$ per state.

(II) Lookup Step: "Logistics mal anders"

Abstraction induced by $\pi_{\{\text{package},\text{truck A}\}}$:



(II) Lookup Step: "Logistics mal anders", ctd.

Pattern variables and domains:

- $P = \{v_1, v_2\}$ with $v_1 =$ package, $v_2 =$ truck A.
- $D_{v_1} = \{L, R, A, B\} \approx \{0, 1, 2, 3\}$
- $D_{v_2} = \{L, R\} \approx \{0, 1\}$

$\rightarrow N_1 = \prod_{j=1}^{0} |D_{v_j}| = 1.$

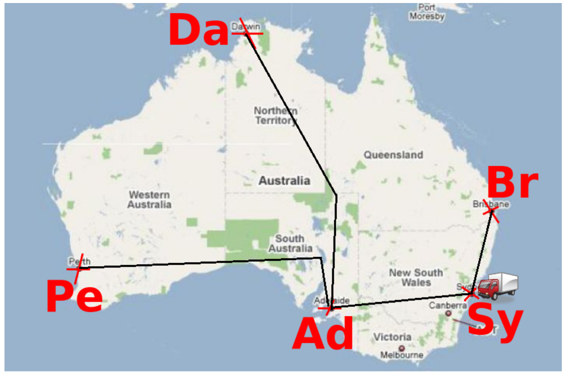
$\rightarrow N_2 = \prod_{j=1}^{1} |D_{v_j}| = 4.$

$\rightarrow$ *index*$(s) = 1 * s($package$) + 4 * s($truck A$).$

$\rightarrow$ **Pattern database:**

abstract state	LL	RL	AL	BL	LR	RR	AR	BR
index	0	1	2	3	4	5	6	7
value	2	0	2	1	2	0	1	1

And Now: The Australia Example



- Variables: $at : \{Sy, Ad, Br, Pe, Ad\}$; $v(x) : \{T, F\}$ for $x \in \{Sy, Ad, Br, Pe, Ad\}$.
- Actions: $drive(x, y)$ where x, y have a road.
- Costs: $Sy \leftrightarrow Br : 1$, $Sy \leftrightarrow Ad : 1.5$, $Ad \leftrightarrow Pe : 3.5$, $Ad \leftrightarrow Da : 4$.
- Initial state: $at = Sy, v(Sy) = T, v(x) = F$ for $x \neq Sy$.
- Goal: $at = Sy, v(x) = T$ for all x.

Question: Say our pattern P is $\{v_1 = v(Br), v_2 = v(Pe), v_3 = v(Da)\}$. What is the PDB?

$\rightarrow D_{v(Br)} = \{F, T\} \approx \{0, 1\}$, $N_1 = 1$; $D_{v(Pe)} = \{F, T\} \approx \{0, 1\}$, $N_2 = 2$; $D_{v(Da)} = \{F, T\} \approx \{0, 1\}$, $N_3 = 4$.

abstract state	FFF	TFF	FTF	TTF	FFT	TFT	FTT	TTT
index	0	1	2	3	4	5	6	7
value	8.5	7.5	5	4	4.5	3.5	1	0

Summary

- An abstraction α is a surjective function on a transition system Θ (e.g., of a planning task).
- The abstract state space Θ^α inherits the initial state, goal states, and transitions from Θ.
- Remaining cost in Θ^α is the abstraction heuristic h^α, which is safe, goal-aware, admissible, and consistent.
- Practically useful abstractions yield informative heuristics at a small computational overhead.
- Pattern database (PDB) heuristics are abstraction heuristics based on projection to a subset of variables: the pattern. For FDR tasks, they can easily be implemented via syntactic projection on the task representation.
- Pattern databases are lookup tables that store heuristic values, indexed by perfect hash values for projected states.

Motivation for Pattern Database Heuristics

→ Pattern databases are a concrete method for designing abstraction functions α, and for computing the associated heuristic functions.

There's many good reasons to be considering PDBs:

- Pattern database (PDB) heuristics are the most commonly used class of abstraction heuristics outside planning (Games, mostly).
- PDBs are the most commonly used classes of abstraction heuristics in planning
- PDBs have been a very active research area from their inception, and still are a very active research area today. (Theoretical properties, how to implement and use PDBs effectively, how to find good patterns, . . .)
- For many search problems, pattern databases are the most effective admissible heuristics currently known.

References I

Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

Klaus Dräger, Bernd Finkbeiner, and Andreas Podelski. Directed model checking with distance-preserving abstractions. In Antti Valmari, editor, *Proceedings of the 13th International SPIN Workshop (SPIN 2006)*, volume 3925 of *Lecture Notes in Computer Science*, pages 19–34. Springer-Verlag, 2006.

Stefan Edelkamp. Planning with pattern databases. In A. Cesta and D. Borrajo, editors, *Proceedings of the 6th European Conference on Planning (ECP'01)*, pages 13–24. Springer-Verlag, 2001.

Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In Adele Howe and Robert C. Holte, editors, *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, pages 1007–1012, Vancouver, BC, Canada, July 2007. AAAI Press.

References II

Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In Mark Boddy, Maria Fox, and Sylvie Thiebaux, editors, *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pages 176–183, Providence, Rhode Island, USA, 2007. Morgan Kaufmann.

Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery*, 61(3):16:1–16:63, 2014.

István T. Hernádvölgyi and Robert C. Holte. Experiments with automatically created memory-based heuristics. In Berthe Y. Choueiry and Toby Walsh, editors, *Proceedings of the 4th International Symposium on Abstraction, Reformulation and Approximation (SARA 2000)*, volume 1864 of *Lecture Notes in Artificial Intelligence*, pages 281–290. Springer-Verlag, 2000.

Michael Katz and Carmel Domshlak. Structural patterns heuristics via fork decomposition. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, editors, *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, pages 182–189. AAAI Press, 2008.

References III

Michael Katz, Jörg Hoffmann, and Malte Helmert. How to relax a bisimulation? In Blai Bonet, Lee McCluskey, José Reinaldo Silva, and Brian Williams, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, pages 101–109. AAAI Press, 2012.

Jendrik Seipp and Malte Helmert. Counterexample-guided Cartesian abstraction refinement. In Daniel Borrajo, Simone Fratini, Subbarao Kambhampati, and Angelo Oddi, editors, *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, pages 347–351, Rome, Italy, 2013. AAAI Press.