Automated (AI) Planning

Abstractions and Abstraction Heuristics

Coming up with heuristics in a principled way

General procedure for obtaining a heuristic

Solve an easier version of the problem.

Two common methods:

- relaxation: consider less constrained version of the problem
- abstraction: consider smaller version of real problem

In the previous chapter, we have studied relaxation, which has been very successfully applied to satisficing planning.

Now, we study abstraction, which is one of the most prominent techniques for optimal planning.

Outline

1. Abstractions informally
2. Abstractions formally
3. Projection abstractions (PDBs)
4. Merge-and-shrink abstractions
5. Generalized additive heuristics
6. Structural-pattern abstractions

Abstracting a transition system

Abstracting a transition system means dropping some distinctions between states, while preserving the transition behaviour as much as possible.

- An abstraction of a transition system $\mathcal{T}$ is defined by an abstraction mapping α that defines which states of $\mathcal{T}$ should be distinguished and which ones should not.
- From $\mathcal{T}$ and α, we compute an abstract transition system $\mathcal{T}'$ which is similar to $\mathcal{T}$, but smaller.
- The abstract goal distances (goal distances in $\mathcal{T}'$) are used as heuristic estimates for goal distances in $\mathcal{T}$.

Abstracting a transition system: example

Example (15-puzzle)

A 15-puzzle state is given by a permutation $\langle b, t_1, \ldots, t_{15} \rangle$ of $\{1, \ldots, 16\}$, where b denotes the blank position and the other components denote the positions of the 15 tiles.

One possible abstraction mapping ignores the precise location of tiles 8–15, i. e., two states are distinguished iff they differ in the position of the blank or one of the tiles 1–7:

$$\alpha(\langle b, t_1, \ldots, t_{15} \rangle) = \langle b, t_1, \ldots, t_7 \rangle$$

The heuristic values for this abstraction correspond to the cost of moving tiles 1–7 to their goal positions.

Abstraction example: 15-puzzle

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

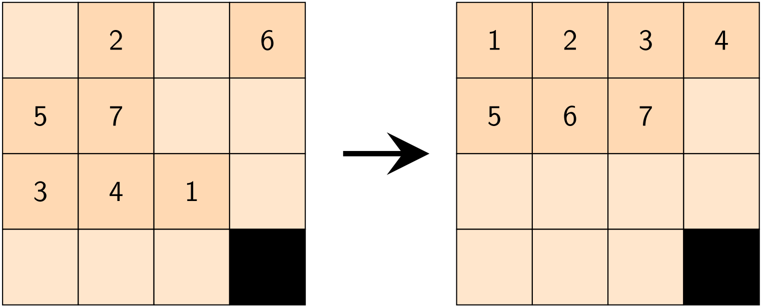
→

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

real state space

- $16! = 20922789888000 \approx 2 \cdot 10^{13}$ states
- $\frac{16!}{2} = 10461394944000 \approx 10^{13}$ reachable states

Abstraction example: 15-puzzle



abstract state space

- $16 \cdot 15 \cdot \ldots \cdot 9 = 518918400 \approx 5 \cdot 10^8$ states
- $16 \cdot 15 \cdot \ldots \cdot 9 = 518918400 \approx 5 \cdot 10^8$ reachable states

Computing the abstract transition system

Given $\mathcal{T}$ and α, how do we compute $\mathcal{T}'$?

Requirement

We want to obtain an admissible heuristic.
Hence, $h^*(\alpha(s))$ (in the abstract state space $\mathcal{T}'$) should never overestimate $h^*(s)$ (in the concrete state space $\mathcal{T}$).

An easy way to achieve this is to ensure that all solutions in $\mathcal{T}$ also exist in $\mathcal{T}'$:

- If s is a goal state in $\mathcal{T}$, then $\alpha(s)$ is a goal state in $\mathcal{T}'$.
- If $\mathcal{T}$ has a transition from s to t, then $\mathcal{T}'$ has a transition from $\alpha(s)$ to $\alpha(t)$.

Practical requirements for abstractions

To be useful in practice, an abstraction heuristic must be efficiently computable. This gives us two requirements for α:

- For a given state s, the abstract state $\alpha(s)$ must be efficiently computable.
- For a given abstract state $\alpha(s)$, the abstract goal distance $h^*(\alpha(s))$ must be efficiently computable.

There are different ways of achieving these requirements:

- pattern database heuristics (Culberson & Schaeffer, 1996)
- merge-and-shrink abstractions (Dräger, Finkbeiner & Podelski, 2006)
- structural patterns (Katz & Domshlak, 2008)

Practical requirements for abstractions: example

Example (15-puzzle)

In our running example, α can be very efficiently computed: just project the given 16-tuple to its first 8 components.

To compute abstract goal distances efficiently during search, most common algorithms precompute all abstract goal distances prior to search by performing a backward breadth-first search from the goal state(s). The distances are then stored in a table (requires about 495 MB of RAM).
During search, computing $h^*(\alpha(s))$ is just a table lookup.

This heuristic is an example of a pattern database heuristic.

Multiple abstractions

- One important practical question is how to come up with a suitable abstraction mapping α.
- Indeed, there is usually a huge number of possibilities, and it is important to pick good abstractions (i. e., ones that lead to informative heuristics).
- However, it is generally not necessary to commit to a single abstraction.

Combining multiple abstractions

Maximizing several abstractions:

- Each abstraction mapping gives rise to an admissible heuristic.
- By computing the maximum of several admissible heuristics, we obtain another admissible heuristic which dominates the component heuristics.
- Thus, we can always compute several abstractions and maximize over the individual abstract goal distances.

Adding several abstractions:

- In some cases, we can even compute the sum of individual estimates and still stay admissible.
- Summation often leads to much higher estimates than maximization, so it is important to understand when it is admissible.

Maximizing several abstractions: example

Example (15-puzzle)

- mapping to tiles 1–7 was arbitrary
  $\rightsquigarrow$ can use any subset of tiles
- with the same amount of memory required for the tables for the mapping to tiles 1–7, we could store the tables for nine different abstractions to six tiles and the blank
- use maximum of individual estimates

Adding several abstractions: example

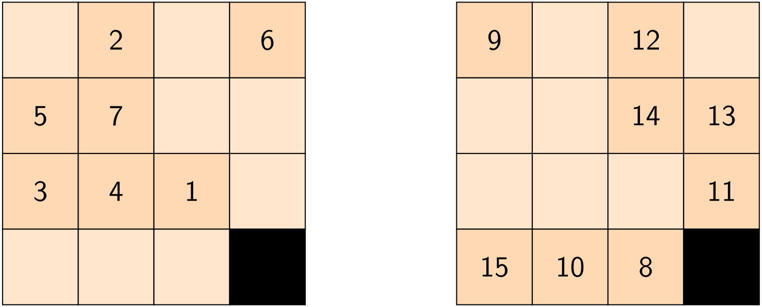
9	2	12	6
5	7	14	13
3	4	1	11
15	10	8	

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

- 1st abstraction: ignore precise location of 8–15
- 2nd abstraction: ignore precise location of 1–7

⇝ Is the sum of the abstraction heuristics admissible?

Adding several abstractions: example



- 1st abstraction: ignore precise location of 8–15
- 2nd abstraction: ignore precise location of 1–7

⇝ The sum of the abstraction heuristics is not admissible.

Adding several abstractions: example

	2		6
5	7		
3	4	1	

9		12	
		14	13
			11
15	10	8	

- 1st abstraction: ignore precise location of 8–15 and blank
- 2nd abstraction: ignore precise location of 1–7 and blank

⇝ The sum of the abstraction heuristics is admissible.

Our plan for the lecture

In the following, we take a deeper look at abstractions and their use for admissible heuristics.

- In the rest of this chapter, we formally introduce abstractions and abstraction heuristics and study some of their most important properties.
- In the following chapters, we discuss some particular classes of abstraction heuristics in detail, namely pattern database heuristics, merge-and-shrink abstractions, and structural patterns.

Outline

1. Abstractions informally
2. Abstractions formally
3. Projection abstractions (PDBs)
4. Merge-and-shrink abstractions
5. Generalized additive heuristics
6. Structural-pattern abstractions

Transition systems

Definition (transition system)

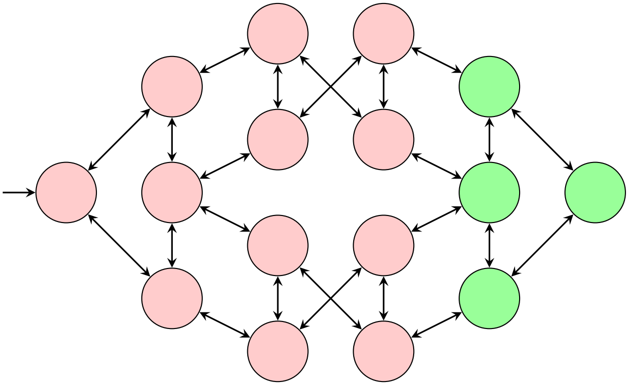
A transition system is a 5-tuple $\mathcal{T} = \langle S, L, T, I, G \rangle$ where

- S is a finite set of states (the state space),
- L is a finite set of (transition) labels,
- $T \subseteq S \times L \times S$ is the transition relation,
- $I \subseteq S$ is the set of initial states, and
- $G \subseteq S$ is the set of goal states.

We say that $\mathcal{T}$ has the transition $\langle s, l, s' \rangle$ if $\langle s, l, s' \rangle \in T$.

Note: For technical reasons, the definition slightly differs from our earlier one. (It includes explicit labels.)

Transition systems: example



Note: To reduce clutter, our figures usually omit arc labels and collapse transitions between identical states. However, these are important for the formal definition of the transition system.

Transition systems of SAS$^+$ planning tasks

Definition (transition system of an SAS$^+$ planning task)

Let $\Pi = \langle V, I, O, G \rangle$ be an SAS$^+$ planning task.
The transition system of Π, in symbols $\mathcal{T}(\Pi)$, is the transition system $\mathcal{T}(\Pi) = \langle S', L', T', I', G' \rangle$, where

- S' is the set of states over V,
- $L' = O$,
- $T' = \{\langle s', o', t' \rangle \in S' \times L' \times S' \mid \mathit{app}_{o'}(s') = t'\}$,
- $I' = \{I\}$, and
- $G' = \{s' \in S' \mid s' \models G\}$.

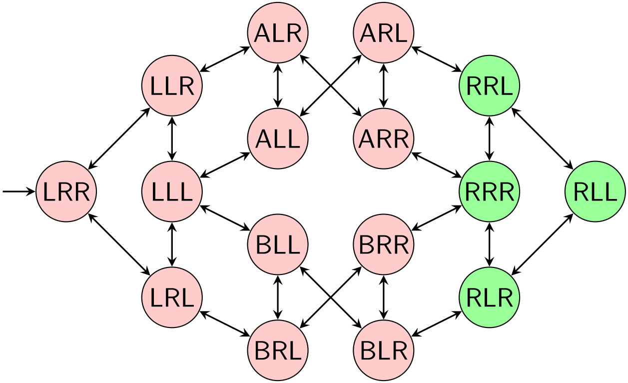
Example task: one package, two trucks

Example (one package, two trucks)

Consider the following SAS$^+$ planning task $\langle V, I, O, G \rangle$:

- $V = \{p, t_\mathsf{A}, t_\mathsf{B}\}$ with
  - $\mathcal{D}_p = \{\mathsf{L}, \mathsf{R}, \mathsf{A}, \mathsf{B}\}$
  - $\mathcal{D}_{t_\mathsf{A}} = \mathcal{D}_{t_\mathsf{B}} = \{\mathsf{L}, \mathsf{R}\}$
- $I = \{p \mapsto \mathsf{L}, t_\mathsf{A} \mapsto \mathsf{R}, t_\mathsf{B} \mapsto \mathsf{R}\}$
- $O = \{\mathsf{pickup}_{i,j} \mid i \in \{\mathsf{A}, \mathsf{B}\}, j \in \{\mathsf{L}, \mathsf{R}\}\}$
  $\cup \{\mathsf{drop}_{i,j} \mid i \in \{\mathsf{A}, \mathsf{B}\}, j \in \{\mathsf{L}, \mathsf{R}\}\}$
  $\cup \{\mathsf{move}_{i,j,j'} \mid i \in \{\mathsf{A}, \mathsf{B}\}, j, j' \in \{\mathsf{L}, \mathsf{R}\}, j \neq j'\}$, where
  - $\mathsf{pickup}_{i,j} = \langle t_i = j \land p = j, p := i \rangle$
  - $\mathsf{drop}_{i,j} = \langle t_i = j \land p = i, p := j \rangle$
  - $\mathsf{move}_{i,j,j'} = \langle t_i = j, t_i := j' \rangle$
- $G = (p = \mathsf{R})$

Transition system of example task



- State $\{p \mapsto i, t_{\mathsf{A}} \mapsto j, t_{\mathsf{B}} \mapsto k\}$ is depicted as ijk.
- Transition labels are again not shown. For example, the transition from LLL to ALL has the label $\mathsf{pickup}_{\mathsf{A,L}}$.

Abstractions

Definition (abstraction, abstraction mapping)

Let $\mathcal{T} = \langle S, L, T, I, G \rangle$ and $\mathcal{T}' = \langle S', L', T', I', G' \rangle$ be transition systems with the same label set $L = L'$, and let $\alpha : S \to S'$.

We say that $\mathcal{T}'$ is an abstraction of $\mathcal{T}$ with abstraction mapping α (or: abstraction function α) if

- for all $s \in I$, we have $\alpha(s) \in I'$,
- for all $s \in G$, we have $\alpha(s) \in G'$, and
- for all $\langle s, l, t \rangle \in T$, we have $\langle \alpha(s), l, \alpha(t) \rangle \in T'$.

Abstraction heuristics

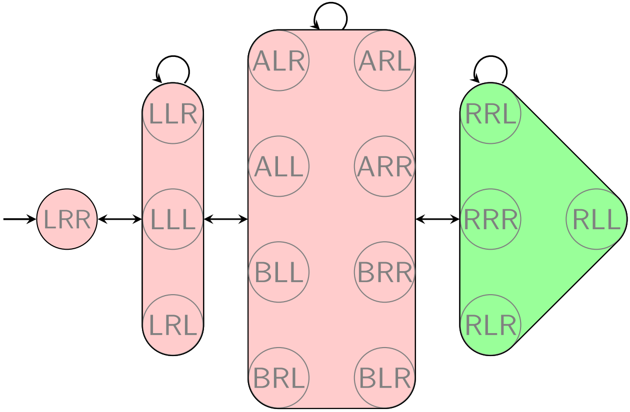
Definition (abstraction heuristic)

Let Π be an SAS$^+$ planning task with state space S, and let $\mathcal{A}$ be an abstraction of $\mathcal{T}(\Pi)$ with abstraction mapping α.

The abstraction heuristic induced by $\mathcal{A}$ and α, $h^{\mathcal{A},\alpha}$, is the heuristic function $h^{\mathcal{A},\alpha} : S \to \mathbb{N}_0 \cup \{\infty\}$ which maps each state $s \in S$ to $h^*_{\mathcal{A}}(\alpha(s))$ (the goal distance of $\alpha(s)$ in $\mathcal{A}$).

Note: $h^{\mathcal{A},\alpha}(s) = \infty$ if no goal state of $\mathcal{A}$ is reachable from $\alpha(s)$

Abstraction heuristics: example



$$h^{\mathcal{A},\alpha}(\{p \mapsto \mathsf{L}, t_{\mathsf{A}} \mapsto \mathsf{R}, t_{\mathsf{B}} \mapsto \mathsf{R}\}) = 3$$

Consistency of abstraction heuristics

Theorem (consistency and admissibility of $h^{\mathcal{A},\alpha}$)

Let Π be an SAS$^+$ planning task, and let $\mathcal{A}$ be an abstraction of $\mathcal{T}(\Pi)$ with abstraction mapping α.
Then $h^{\mathcal{A},\alpha}$ is safe, goal-aware, admissible and consistent.

Orthogonality of abstraction mappings

Definition (orthogonal abstraction mappings)

Let α_1 and α_2 be abstraction mappings on $\mathcal{T}$.

We say that α_1 and α_2 are orthogonal if for all transitions $\langle s, l, t \rangle$ of $\mathcal{T}$, we have $\alpha_i(s) = \alpha_i(t)$ for at least one $i \in \{1, 2\}$.

Affecting transition labels

Definition (affecting transition labels)

Let $\mathcal{T}$ be a transition system, and let l be one of its labels.
We say that l affects $\mathcal{T}$ if $\mathcal{T}$ has a transition $\langle s, l, t \rangle$ with $s \neq t$.

Theorem (affecting labels vs. orthogonality)

Let $\mathcal{A}_1$ be an abstraction of $\mathcal{T}$ with abstraction mapping α_1.
Let $\mathcal{A}_2$ be an abstraction of $\mathcal{T}$ with abstraction mapping α_2.

If no label of $\mathcal{T}$ affects both $\mathcal{A}_1$ and $\mathcal{A}_2$, then α_1 and α_2 are orthogonal.

Orthogonal abstraction mappings: example

	2		6
5	7		
3	4	1	

9		12	
		14	13
			11
15	10	8	

Are the abstraction mappings orthogonal?

Orthogonal abstraction mappings: example

	2		6
5	7		
3	4	1	

9		12	
		14	13
			11
15	10	8	

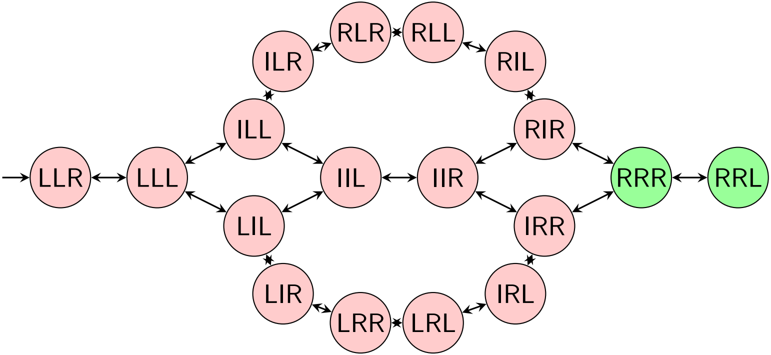
Are the abstraction mappings orthogonal?

Orthogonality and additivity

Theorem (additivity for orthogonal abstraction mappings)

Let $h^{\mathcal{A}_1,\alpha_1}, \dots, h^{\mathcal{A}_n,\alpha_n}$ *be abstraction heuristics for the same planning task* Π *such that* α_i *and* α_j *are orthogonal for all* $i \neq j$.
Then $\sum_{i=1}^{n} h^{\mathcal{A}_i,\alpha_i}$ *is a safe, goal-aware, admissible and consistent heuristic for* Π.

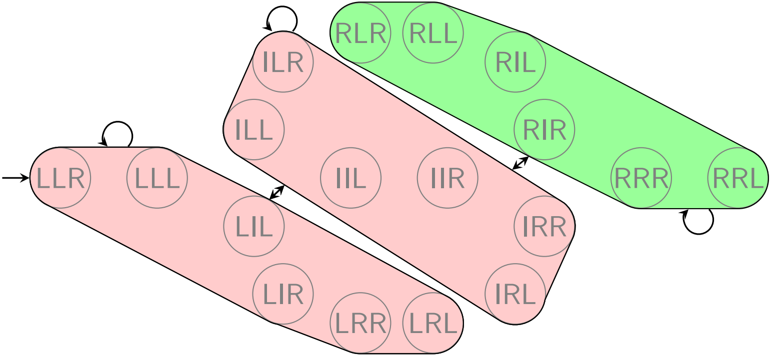
Orthogonality and additivity: example



transition system $\mathcal{T}$

state variables: first package, second package, truck

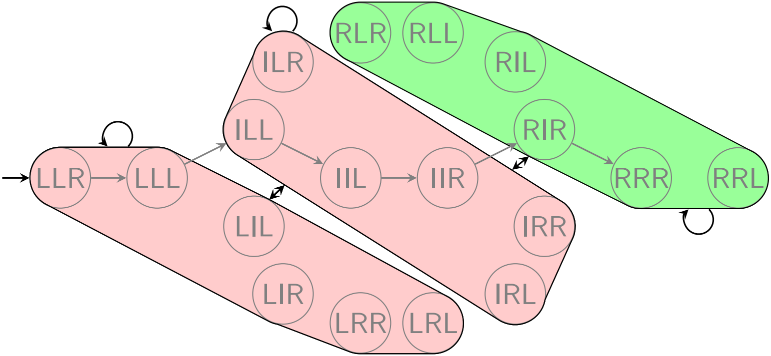
Orthogonality and additivity: example



abstraction $\mathcal{A}_1$

mapping: only consider state of first package

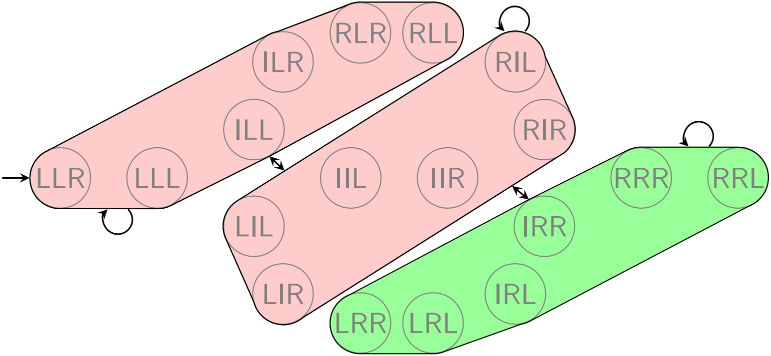
Orthogonality and additivity: example



abstraction $\mathcal{A}_1$

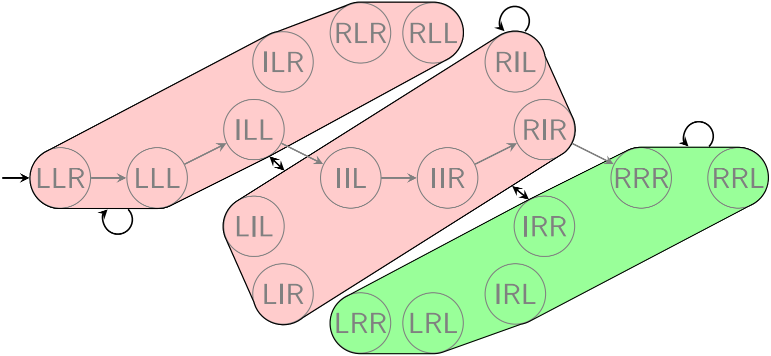
mapping: only consider state of first package

Orthogonality and additivity: example



abstraction $\mathcal{A}_2$ (orthogonal to $\mathcal{A}_1$)
mapping: only consider state of second package

Orthogonality and additivity: example



abstraction $\mathcal{A}_2$ (orthogonal to $\mathcal{A}_1$)
mapping: only consider state of second package

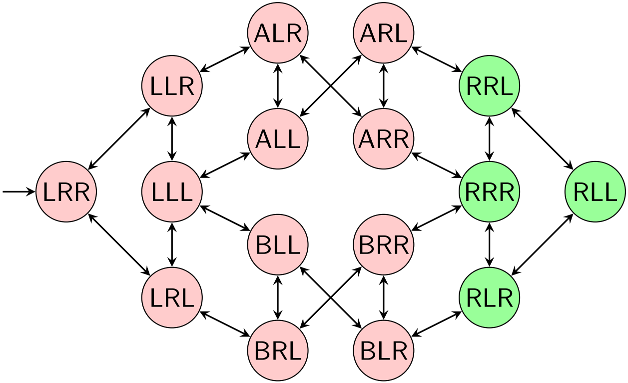
Abstractions of abstractions

Theorem (transitivity of abstractions)

Let $\mathcal{T}$, $\mathcal{T}'$ and $\mathcal{T}''$ be transition systems.

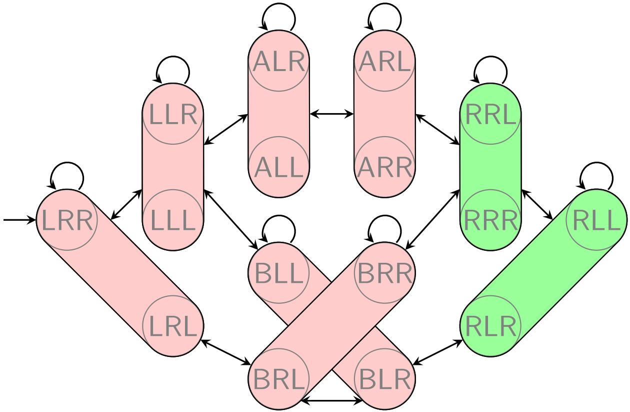
- *If $\mathcal{T}'$ is an abstraction of $\mathcal{T}$ and $\mathcal{T}''$ is an abstraction of $\mathcal{T}'$, then $\mathcal{T}''$ is an abstraction of $\mathcal{T}$.*
- *If $\mathcal{T}'$ is a homomorphic abstraction of $\mathcal{T}$ and $\mathcal{T}''$ is a homomorphic abstraction of $\mathcal{T}'$, then $\mathcal{T}''$ is a homomorphic abstraction of $\mathcal{T}$.*

Abstractions of abstractions: example



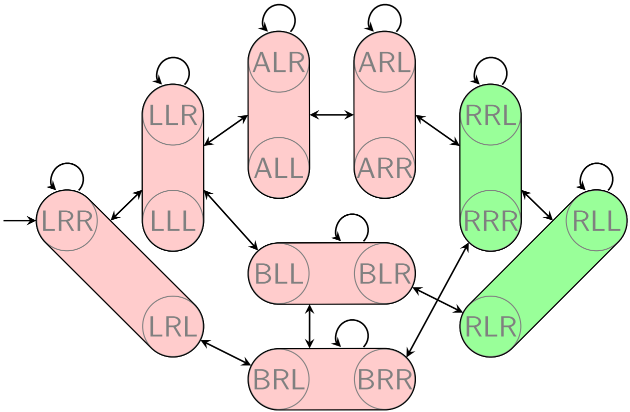
transition system $\mathcal{T}$

Abstractions of abstractions: example



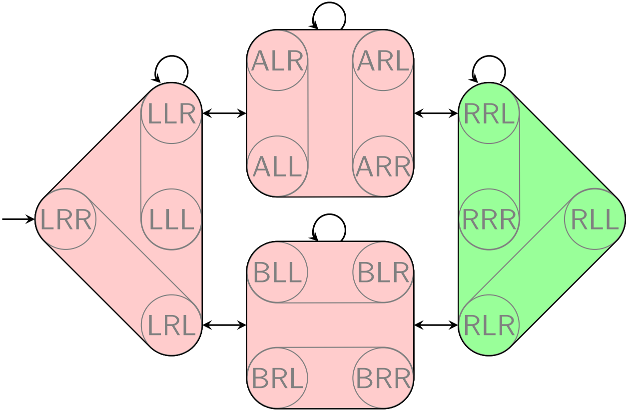
Transition system $\mathcal{T}'$ as an abstraction of $\mathcal{T}$

Abstractions of abstractions: example



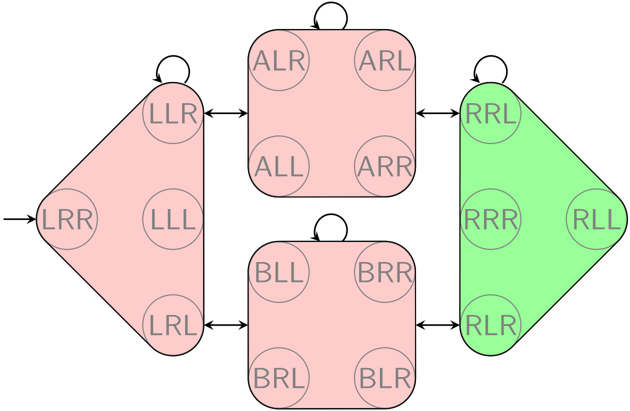
Transition system $\mathcal{T}'$ as an abstraction of $\mathcal{T}$

Abstractions of abstractions: example



Transition system $\mathcal{T}''$ as an abstraction of $\mathcal{T}'$

Abstractions of abstractions: example



Transition system $\mathcal{T}''$ as an abstraction of $\mathcal{T}$

Coarsenings and refinements

Terminology: Let $\mathcal{T}$ be a transition system,
let $\mathcal{T}'$ be an abstraction of $\mathcal{T}$ with abstraction mapping α, and
let $\mathcal{T}''$ be an abstraction of $\mathcal{T}'$ with abstraction mapping α'.

Then:

- $\langle \mathcal{T}'', \alpha' \circ \alpha \rangle$ is called a coarsening of $\langle \mathcal{T}', \alpha \rangle$, and
- $\langle \mathcal{T}', \alpha \rangle$ is called a refinement of $\langle \mathcal{T}'', \alpha' \circ \alpha \rangle$.

Theorem (heuristic quality of refinements)

Let $h^{\mathcal{A},\alpha}$ and $h^{\mathcal{B},\beta}$ be abstraction heuristics for the same planning task Π such that $\langle \mathcal{A}, \alpha \rangle$ is a refinement of $\langle \mathcal{B}, \beta \rangle$. Then $h^{\mathcal{A},\alpha}$ dominates $h^{\mathcal{B},\beta}$.

In other words, $h^{\mathcal{A},\alpha}(s) \geq h^{\mathcal{B},\beta}(s)$ for all states s of Π.

Using abstraction heuristics in practice

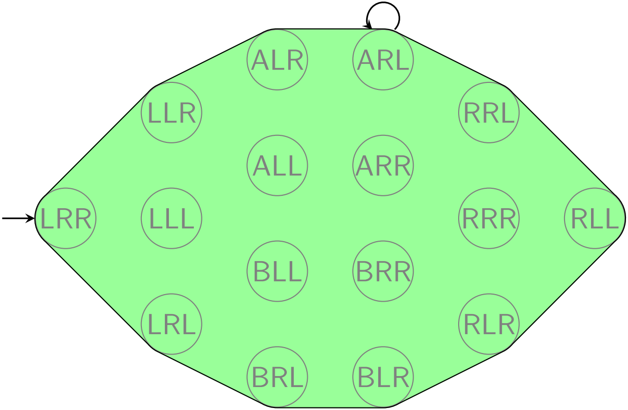
In practice, there are conflicting goals for abstractions:

- we want to obtain an informative heuristic, but
- want to keep its representation small.

Abstractions have small representations if they have

- few abstract states and
- a succinct encoding for α.

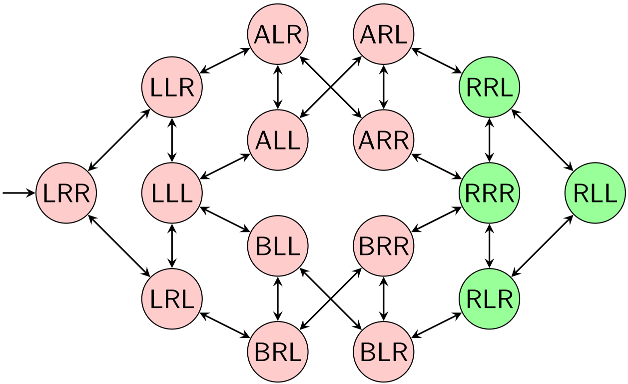
Counterexample: one-state abstraction



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

+ very few abstract states and succinct encoding for α
− completely uninformative heuristic

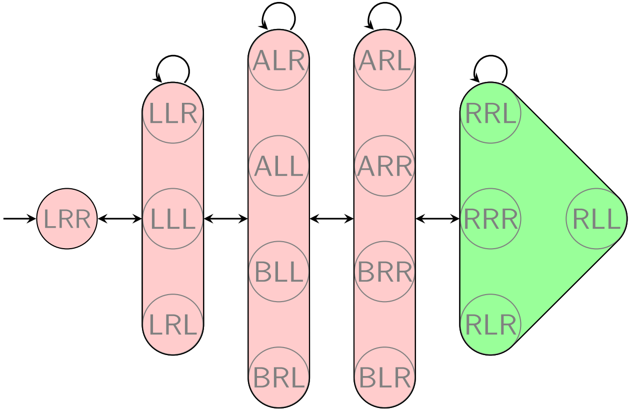
Counterexample: identity abstraction



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

+ perfect heuristic and succinct encoding for α
− too many abstract states

Counterexample: perfect abstraction



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

- $+$ perfect heuristic and usually few abstract states
- $-$ usually no succinct encoding for α

Automatically deriving good abstraction heuristics

Abstraction heuristics for planning: main research problem

Automatically derive effective abstraction heuristics
for planning tasks.

Next we

- ⇝ study three state-of-the-art approaches
  to exploiting abstractions in practice
- ⇝ consider more closely the issue of additivity

Outline

Pattern database heuristics

- The most commonly used abstraction heuristics in search and planning are pattern database (PDB) heuristics.
- PDB heuristics were originally introduced for the 15-puzzle (Culberson & Schaeffer, 1996) and for Rubik's cube (Korf, 1997).
- The first use for domain-independent planning is due to Edelkamp (2001).
- Since then, much research has focused on the theoretical properties of pattern databases, how to use pattern databases more effectively, how to find good patterns, etc.
- Pattern databases are a very active research area both in planning and in (domain-specific) heuristic search.
- For many search problems, pattern databases are the most effective admissible heuristics currently known.

Pattern database heuristics informally

Pattern databases: informally

A pattern database heuristic for a planning task is an abstraction heuristic where

- some aspects of the task are represented in the abstraction with perfect precision, while
- all other aspects of the task are not represented at all.

Example (15-puzzle)

- Choose a subset T of tiles (the pattern).
- Faithfully represent the locations of T in the abstraction.
- Assume that all other tiles and the blank can be anywhere in the abstraction.

Projections

Formally, pattern database heuristics are induced abstractions of a particular class of homomorphisms called projections.

Definition (projections)

Let Π be an SAS$^+$ planning task with variable set V and state set S. Let $P \subseteq V$, and let S' be the set of states over P.

The projection $\pi_P : S \rightarrow S'$ is defined as $\pi_P(s) := s|_P$ (with $s|_P(v) := s(v)$ for all $v \in P$).

We call P the pattern of the projection π_P.

In other words, π_P maps two states s_1 and s_2 to the same abstract state iff they agree on all variables in P.

Pattern database heuristics

Abstraction heuristics for projections are called pattern database (PDB) heuristics.

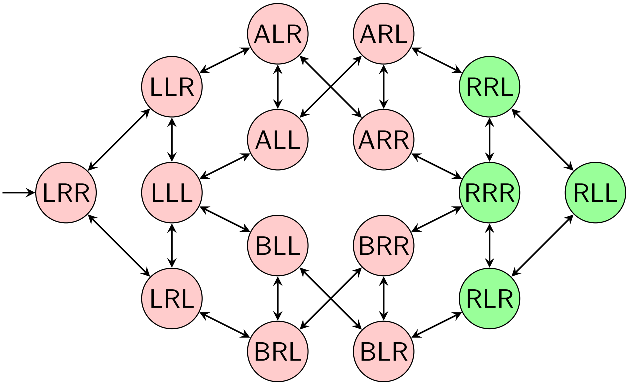
Definition (pattern database heuristic)

The abstraction heuristic induced by π_P is called a pattern database heuristic or PDB heuristic.
We write h^P as a short-hand for h^{π_P}.

Why are they called pattern database heuristics?

- Heuristic values for PDB heuristics are traditionally stored in a 1-dimensional table (array) called a pattern database (PDB). Hence the name "PDB heuristic".

Example: transition system

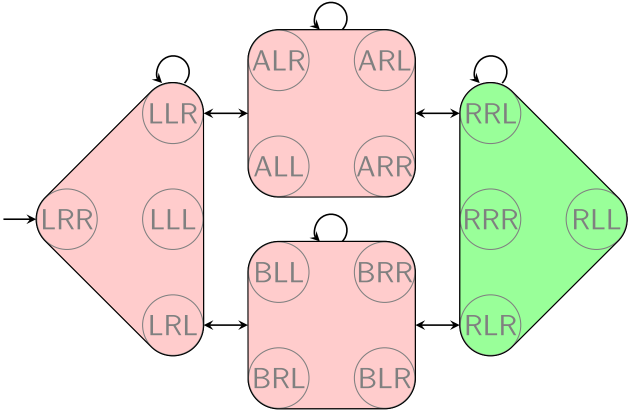


Logistics problem with one package, two trucks, two locations:

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

Example: projection

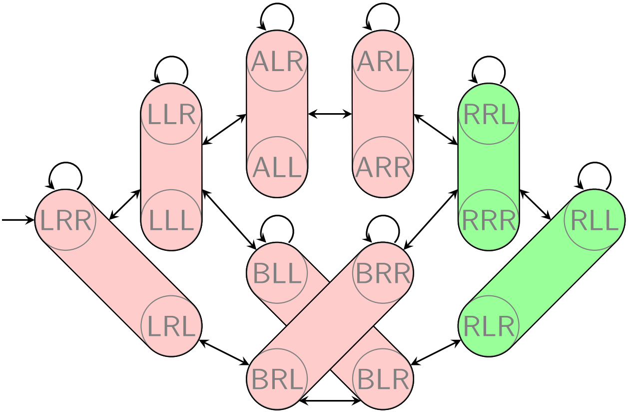
Abstraction induced by $\pi_{\{\text{package}\}}$:



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

Example: projection (2)

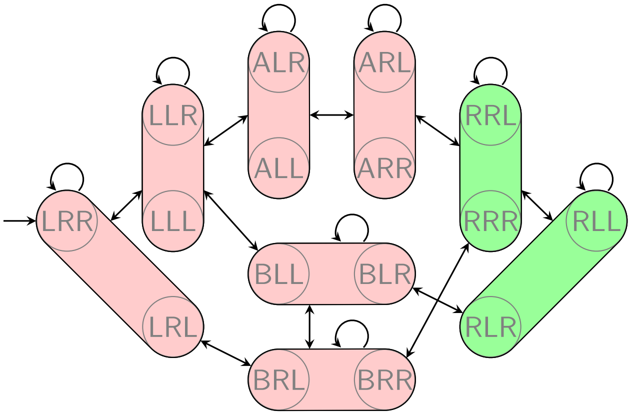
Abstraction induced by $\pi_{\{\text{package},\text{truck A}\}}$:



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

Example: projection (2)

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



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

Pattern collections

- The space requirements for a pattern database grow exponentially with the number of state variables in the pattern.
- This places severe limits on the usefulness of single PDB heuristics h^P for larger planning task.
- To overcome this limitation, planners using pattern databases work with collections of multiple patterns.
- When using two patterns P_1 and P_2, it is always possible to use the maximum of h^{P_1} and h^{P_2} as an admissible and consistent heuristic estimate.
- However, when possible, it is much preferable to use the sum of h^{P_1} and h^{P_2} as a heuristic estimate, since $h^{P_1} + h^{P_2} \geq \max\{h^{P_1}, h^{P_2}\}$.

Criterion for additive patterns

Theorem (additive pattern sets)

Let $P_1, \dots, P_k$ be patterns for an SAS$^+$ planning task Π.

If there exists no operator that has an effect on a variable $v_i \in P_i$ and on a variable $v_j \in P_j$ for some $i \neq j$, then $\sum_{i=1}^{k} h^{P_i}$ is an admissible and consistent heuristic for Π.

A pattern set $\{P_1, \dots, P_k\}$ which satisfies the criterion of the theorem is called an additive pattern set or additive set.

Finding additive pattern sets

The theorem on additive pattern sets gives us a simple criterion to decide which pattern heuristics can be admissibly added.

Given a pattern collection $\mathcal{C}$ (i. e., a set of patterns), we can use this information as follows:

1. Build the compatibility graph for $\mathcal{C}$.
   - Vertices correspond to patterns $P \in \mathcal{C}$.
   - There is an edge between two vertices iff no operator affects both incident patterns.
2. Compute all maximal cliques of the graph.
   These correspond to maximal additive subsets of $\mathcal{C}$.
   - Computing large cliques is an NP-hard problem, and a graph can have exponentially many maximal cliques.
   - However, there are output-polynomial algorithms for finding all maximal cliques (Tomita, Tanaka & Takahashi, 2004) which have led to good results in practice.

The canonical heuristic function

Definition (canonical heuristic function)

Let Π be an SAS$^+$ planning task, and let $\mathcal{C}$ be a pattern collection for Π.

The canonical heuristic $h^{\mathcal{C}}$ for pattern collection $\mathcal{C}$ is defined as

$$h^{\mathcal{C}}(s) = \max_{\mathcal{D} \in \mathrm{cliques}(\mathcal{C})} \sum_{P \in \mathcal{D}} h^P(s),$$

where $\mathrm{cliques}(\mathcal{C})$ is the set of all maximal cliques in the compatibility graph for $\mathcal{C}$.

For all choices of $\mathcal{C}$, heuristic $h^{\mathcal{C}}$ is admissible and consistent.

Canonical heuristic function: example

Example

Consider a planning task with state variables $V = \{v_1, v_2, v_3\}$ and the pattern collection $\mathcal{C} = \{P_1, \ldots, P_4\}$ with $P_1 = \{v_1, v_2\}$, $P_2 = \{v_1\}$, $P_3 = \{v_2\}$ and $P_4 = \{v_3\}$.

There are operators affecting each individual variable, and the only operators affecting several variables affect v_1 and v_3.

What are the maximal cliques in the compatibility graph for $\mathcal{C}$?

Answer: $\{P_1\}$, $\{P_2, P_3\}$, $\{P_3, P_4\}$

What is the canonical heuristic function $h^{\mathcal{C}}$?

Answer:
$$\begin{aligned} h^{\mathcal{C}} &= \max\{h^{P_1}, h^{P_2} + h^{P_3}, h^{P_3} + h^{P_4}\} \\ &= \max\{h^{\{v_1,v_2\}}, h^{\{v_1\}} + h^{\{v_2\}}, h^{\{v_2\}} + h^{\{v_3\}}\} \end{aligned}$$

Canonical heuristic function: example

Example

Consider a planning task with state variables $V = \{v_1, v_2, v_3\}$ and the pattern collection $\mathcal{C} = \{P_1, \dots, P_4\}$ with $P_1 = \{v_1, v_2\}$, $P_2 = \{v_1\}$, $P_3 = \{v_2\}$ and $P_4 = \{v_3\}$.

There are operators affecting each individual variable, and the only operators affecting several variables affect v_1 and v_3.

What are the maximal cliques in the compatibility graph for $\mathcal{C}$?

Answer: $\{P_1\}$, $\{P_2, P_3\}$, $\{P_3, P_4\}$

What is the canonical heuristic function $h^{\mathcal{C}}$?

Answer:

$$h^{\mathcal{C}} = \max\{h^{P_1}, h^{P_2} + h^{P_3}, h^{P_3} + h^{P_4}\}$$
$$= \max\{h^{\{v_1,v_2\}}, h^{\{v_1\}} + h^{\{v_2\}}, h^{\{v_2\}} + h^{\{v_3\}}\}$$

Canonical heuristic function: example

Example

Consider a planning task with state variables $V = \{v_1, v_2, v_3\}$ and the pattern collection $\mathcal{C} = \{P_1, \dots, P_4\}$ with $P_1 = \{v_1, v_2\}$, $P_2 = \{v_1\}$, $P_3 = \{v_2\}$ and $P_4 = \{v_3\}$.

There are operators affecting each individual variable, and the only operators affecting several variables affect v_1 and v_3.

What are the maximal cliques in the compatibility graph for $\mathcal{C}$?

Answer: $\{P_1\}$, $\{P_2, P_3\}$, $\{P_3, P_4\}$

What is the canonical heuristic function $h^{\mathcal{C}}$?

Answer:
$$h^{\mathcal{C}} = \max\{h^{P_1}, h^{P_2} + h^{P_3}, h^{P_3} + h^{P_4}\}$$
$$= \max\{h^{\{v_1,v_2\}}, h^{\{v_1\}} + h^{\{v_2\}}, h^{\{v_2\}} + h^{\{v_3\}}\}$$

Canonical heuristic function: example

Example

Consider a planning task with state variables $V = \{v_1, v_2, v_3\}$ and the pattern collection $\mathcal{C} = \{P_1, \dots, P_4\}$ with $P_1 = \{v_1, v_2\}$, $P_2 = \{v_1\}$, $P_3 = \{v_2\}$ and $P_4 = \{v_3\}$.

There are operators affecting each individual variable, and the only operators affecting several variables affect v_1 and v_3.

What are the maximal cliques in the compatibility graph for $\mathcal{C}$?

Answer: $\{P_1\}$, $\{P_2, P_3\}$, $\{P_3, P_4\}$

What is the canonical heuristic function $h^{\mathcal{C}}$?

Answer:
$$\begin{aligned} h^{\mathcal{C}} &= \max\{h^{P_1}, h^{P_2} + h^{P_3}, h^{P_3} + h^{P_4}\} \\ &= \max\{h^{\{v_1,v_2\}}, h^{\{v_1\}} + h^{\{v_2\}}, h^{\{v_2\}} + h^{\{v_3\}}\} \end{aligned}$$

How good is the canonical heuristic function?

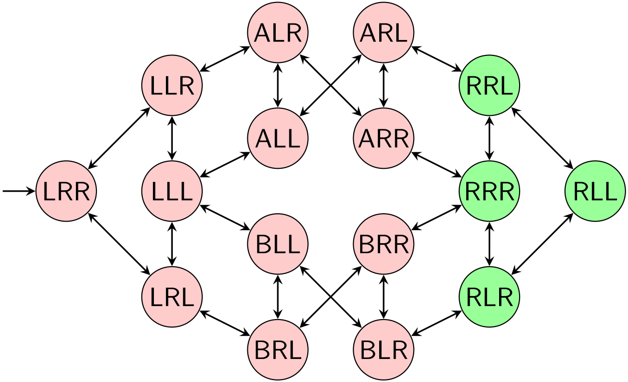
- The canonical heuristic function is the best possible admissible heuristic we can derive from $\mathcal{C}$ using the additivity criterion of orthogonality.
- However, even better heuristic estimates can be obtained from projection heuristics using a more general additivity criterion based on an idea called cost partitioning.
  $\rightsquigarrow$ *more on that later*.

Outline

Beyond pattern databases

- Despite their popularity, pattern databases have some fundamental limitations ($\rightsquigarrow$ example on next slides).
- In this chapter, we study a recently introduced class of abstractions called merge-and-shrink abstractions.
- Merge-and-shrink abstractions can be seen as a proper generalization of pattern databases.
  - They can do everything that pattern databases can do (modulo polynomial extra effort).
  - They can do some things that pattern databases cannot.
- Initial experiments with merge-and-shrink abstractions have shown very promising results.
- They have provably greater representational power than pattern databases for many common planning domains.

Back to the running example

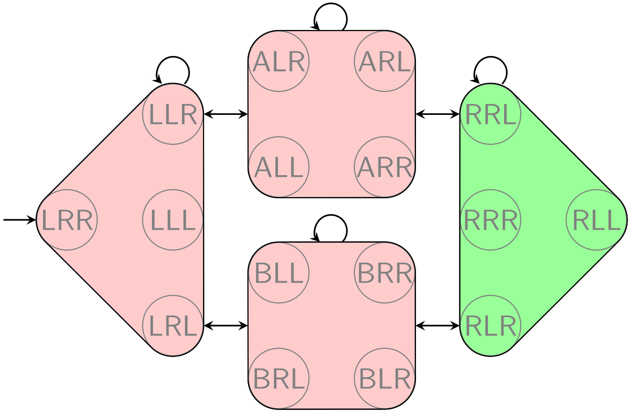


Logistics problem with one package, two trucks, two locations:

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

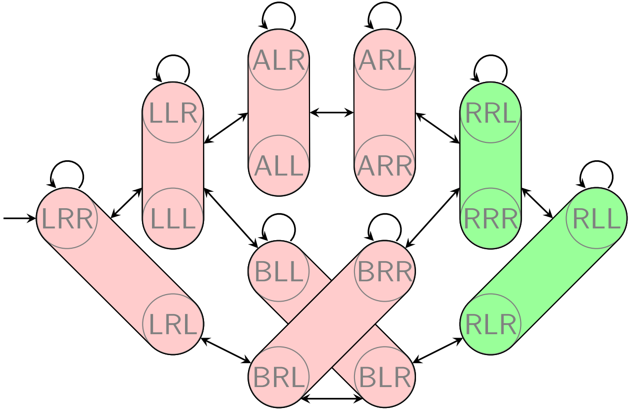
Example: projection

Project to {package}:



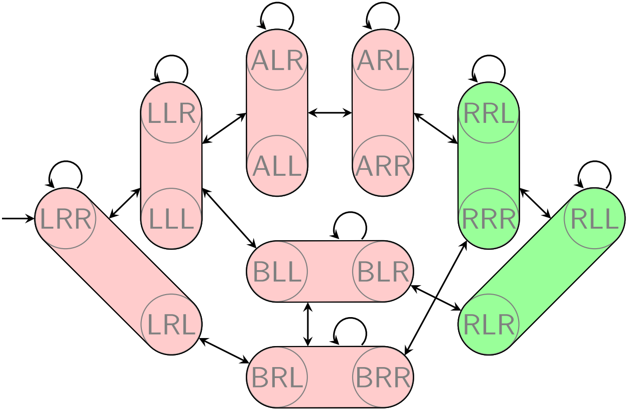
Example: projection (2)

Project to {package, truck A}:



Example: projection (2)

Project to {package, truck A}:



Limitations of projections

How accurate is the PDB heuristic?

- consider generalization of the example:
  N trucks, M locations (fully connected), still one package
- consider any pattern that is proper subset of variable set V
- $h(s_0) \leq 2 \rightsquigarrow$ no better than atomic projection to package

These values cannot be improved by maximizing over several patterns or using additive patterns.

Merge-and-shrink abstractions can represent heuristics with $h(s_0) \geq 3$ for tasks of this kind of any size.
Time and space requirements are polynomial in N and M.

Merge-and-shrink abstractions: main idea

Main idea of merge-and-shrink abstractions

(due to Dräger, Finkbeiner & Podelski, 2006):

Instead of perfectly reflecting a few state variables, reflect all state variables, but in a potentially lossy way.

The need for succinct abstraction mappings

- One major difficulty for non-PDB abstractions is to succinctly represent the abstraction mapping.
- For pattern databases, this is easy because the abstraction mappings – projections – are very structured.
- For less rigidly structured abstraction mappings, we need another idea.

Merge-and-shrink abstractions: idea

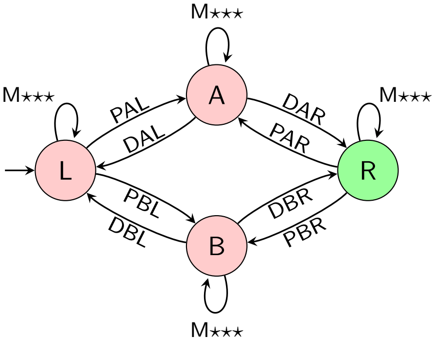
- The main idea underlying merge-and-shrink abstractions is that given two abstractions $\mathcal{A}$ and $\mathcal{A}'$, we can merge them into a new product abstraction.
  - The product abstraction captures all information of both abstractions and can be better informed than either.
  - It can even be better informed than their sum.
- By merging a set of very simple abstractions, we can in theory represent arbitrary abstractions of an SAS^+ task.
- In practice, due to memory limitations, such abstractions can become too large. In that case, we can shrink them by abstracting them further using any abstraction on an intermediate result, then continue the merging process.

Running example: explanations

- Atomic projections – projections to a single state variable – play an important role in this chapter.
- Unlike previous chapters, transition labels are critically important in this chapter.
- Hence we now look at the transition systems for atomic projections of our example task, including transition labels.
- We abbreviate operator names as in these examples:
  - MALR: move truck A from left to right
  - DAR: drop package from truck A at right location
  - PBL: pick up package with truck B at left location
- We abbreviate parallel arcs with commas and wildcards ($\star$) in the labels as in these examples:
  - PAL, DAL: two parallel arcs labeled PAL and DAL
  - MA$\star\star$: two parallel arcs labeled MALR and MARL

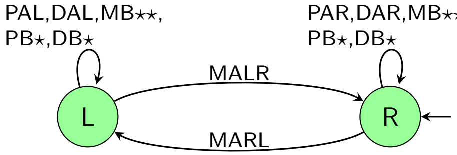
Running example: atomic projection for package

$\mathcal{T}^{\pi_{\{\text{package}\}}}$:



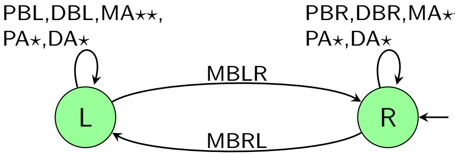
Running example: atomic projection for truck A

$\mathcal{T}^{\pi_{\{\text{truck A}\}}}$:



Running example: atomic projection for truck B

$\mathcal{T}^{\pi_{\{\text{truck B}\}}}$:



Synchronized product of transition systems

Definition (synchronized product of transition systems)

For $i \in \{1, 2\}$, let $\mathcal{T}_i = \langle S_i, L, T_i, I_i, G_i \rangle$ be transition systems with identical label set.

The synchronized product of $\mathcal{T}_1$ and $\mathcal{T}_2$, in symbols $\mathcal{T}_1 \otimes \mathcal{T}_2$, is the transition system $\mathcal{T}_\otimes = \langle S_\otimes, L, T_\otimes, I_\otimes, G_\otimes \rangle$ with

- $S_\otimes := S_1 \times S_2$
- $T_\otimes := \{\langle\langle s_1, s_2\rangle, l, \langle t_1, t_2\rangle\rangle \mid \langle s_1, l, t_1\rangle \in T_1$ and $\langle s_2, l, t_2\rangle \in T_2\}$
- $I_\otimes := I_1 \times I_2$
- $G_\otimes := G_1 \times G_2$

Synchronized product of functions

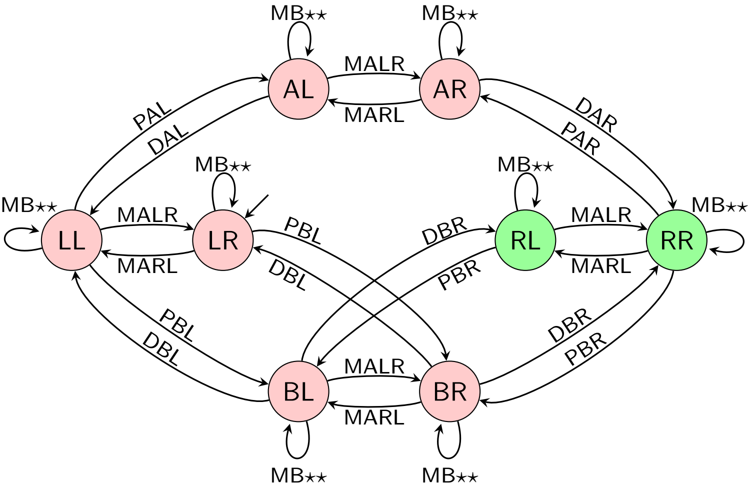
Definition (synchronized product of functions)

Let $\alpha_1 : S \to S_1$ and $\alpha_2 : S \to S_2$ be functions with identical domain.

The synchronized product of α_1 and α_2, in symbols $\alpha_1 \otimes \alpha_2$, is the function $\alpha_\otimes : S \to S_1 \times S_2$ defined as $\alpha_\otimes(s) = \langle \alpha_1(s), \alpha_2(s) \rangle$.

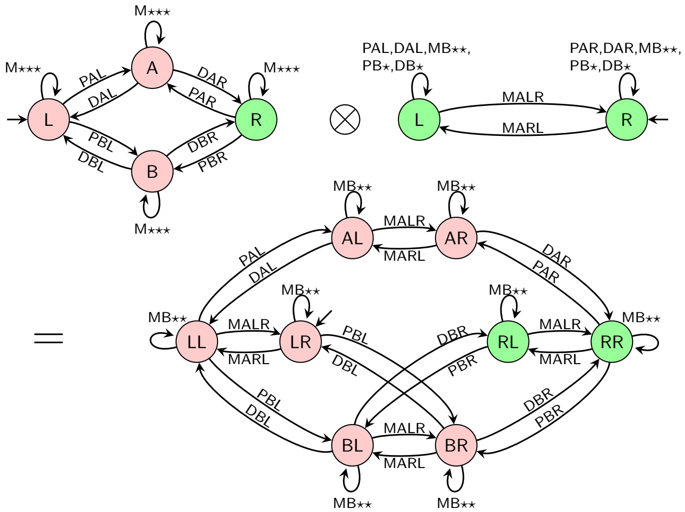
Example: synchronized product

$\mathcal{T}^{\pi_{\{\text{package}\}}} \otimes \mathcal{T}^{\pi_{\{\text{truck A}\}}}$:



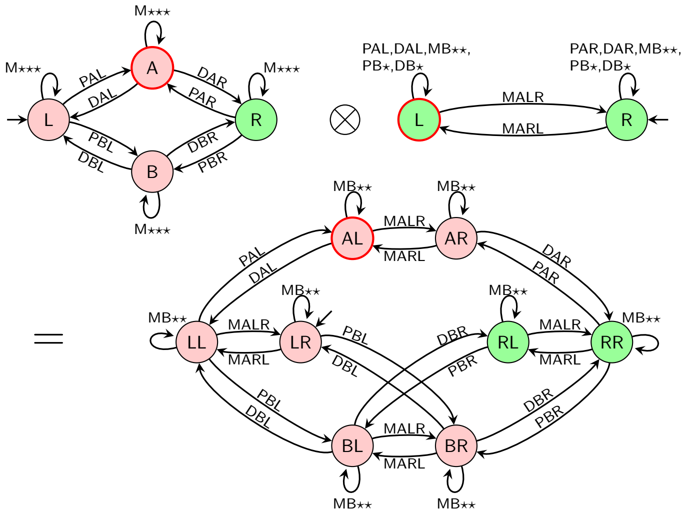
Example: computation of synchronized product

$\mathcal{T}^{\pi_{\{\text{package}\}}} \otimes \mathcal{T}^{\pi_{\{\text{truck A}\}}}$:



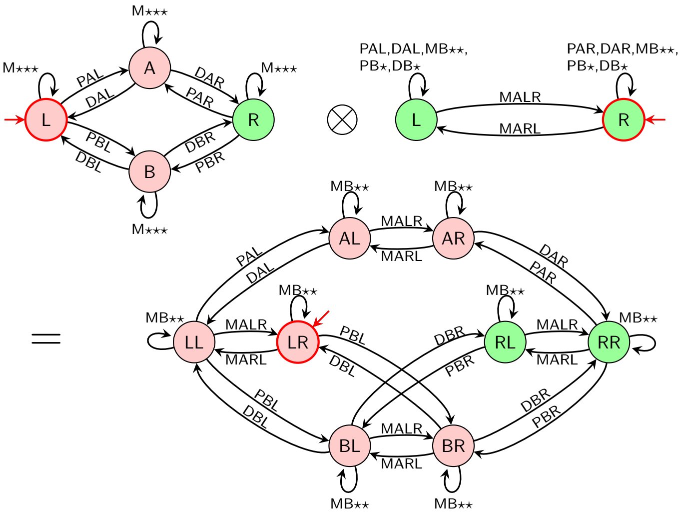
Example: computation of synchronized product

$\mathcal{T}^{\pi_{\{\text{package}\}}} \otimes \mathcal{T}^{\pi_{\{\text{truck A}\}}}$: $S_\otimes = S_1 \times S_2$



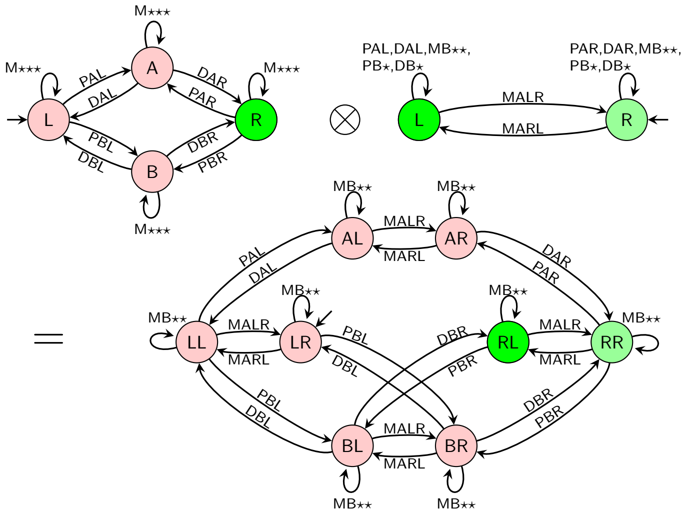
Example: computation of synchronized product

$\mathcal{T}^{\pi\{\text{package}\}} \otimes \mathcal{T}^{\pi\{\text{truck A}\}}$: $I_{\otimes} = I_1 \times I_2$



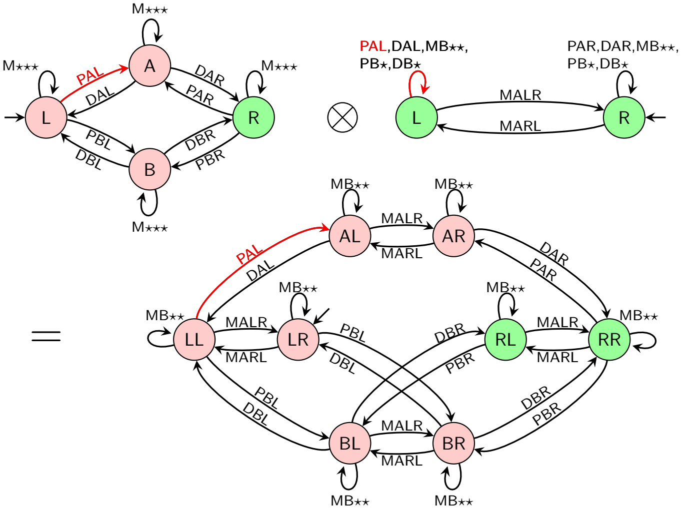
Example: computation of synchronized product

$\mathcal{T}^{\pi_{\{\text{package}\}}} \otimes \mathcal{T}^{\pi_{\{\text{truck A}\}}}$: $G_\otimes = G_1 \times G_2$



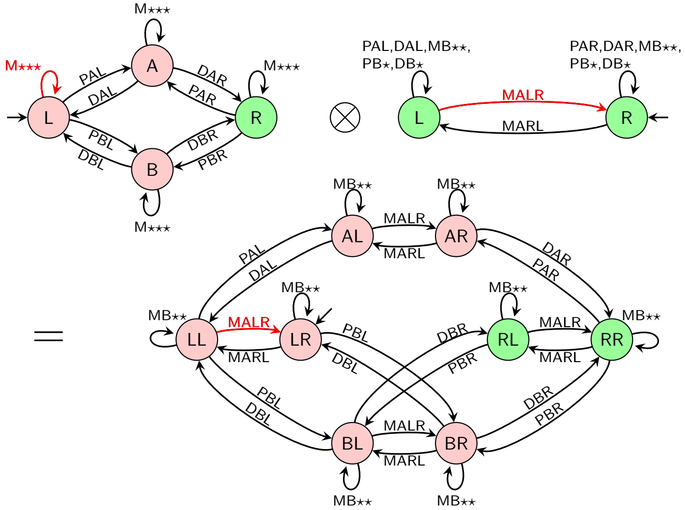
Example: computation of synchronized product

$\mathcal{T}^{\pi_{\{\text{package}\}}} \otimes \mathcal{T}^{\pi_{\{\text{truck A}\}}}$: $T_\otimes := \{\langle\langle s_1, s_2\rangle, l, \langle t_1, t_2\rangle\rangle \mid \ldots\}$



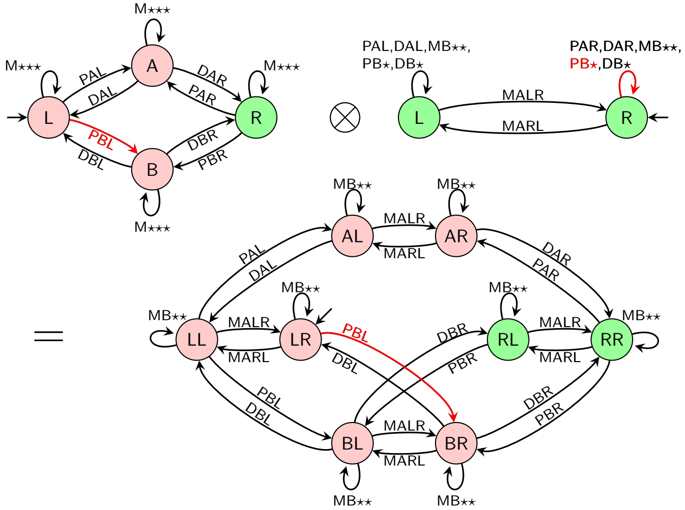
Example: computation of synchronized product

$\mathcal{T}^{\pi_{\{\text{package}\}}} \otimes \mathcal{T}^{\pi_{\{\text{truck A}\}}}$: $T_\otimes := \{\langle\langle s_1, s_2\rangle, l, \langle t_1, t_2\rangle\rangle \mid \ldots\}$



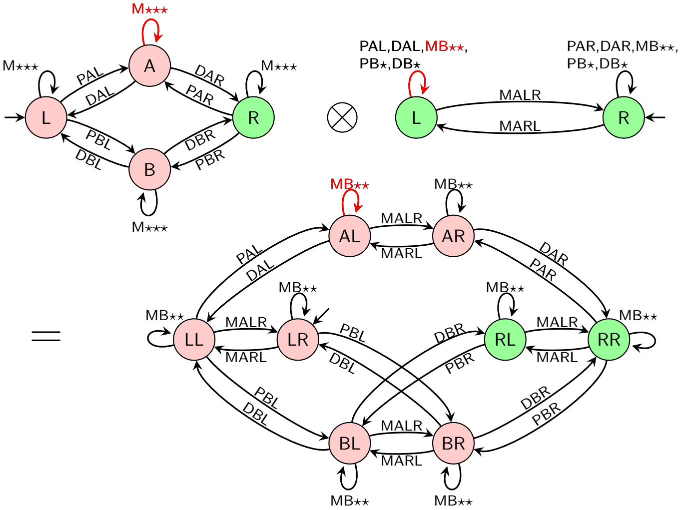
Example: computation of synchronized product

$\mathcal{T}^{\pi_{\{\text{package}\}}} \otimes \mathcal{T}^{\pi_{\{\text{truck A}\}}}$: $T_\otimes := \{\langle\langle s_1, s_2\rangle, l, \langle t_1, t_2\rangle\rangle \mid \dots\}$



Example: computation of synchronized product

$\mathcal{T}^{\pi_{\{\text{package}\}}} \otimes \mathcal{T}^{\pi_{\{\text{truck A}\}}}$: $T_\otimes := \{\langle\langle s_1, s_2\rangle, l, \langle t_1, t_2\rangle\rangle \mid \dots\}$



Synchronized products are abstractions

Theorem (synchronized products are abstractions)

For $i \in \{1, 2\}$, let $\mathcal{T}_i$ be an abstraction of transition system $\mathcal{T}$ with abstraction mapping α_i.

Then $\mathcal{T}_\otimes := \mathcal{T}_1 \otimes \mathcal{T}_2$ is an abstraction of $\mathcal{T}$ with abstraction mapping $\alpha_\otimes := \alpha_1 \otimes \alpha_2$, and $\langle \mathcal{T}_\otimes, \alpha_\otimes \rangle$ is a refinement of $\langle \mathcal{T}_1, \alpha_1 \rangle$ and of $\langle \mathcal{T}_2, \alpha_2 \rangle$.

Synchronized products of projections

Corollary (Synchronized products of projections)

Let Π be an SAS$^+$ planning task with variable set V, and let V_1 and V_2 be disjoint subsets of V.
Then $\mathcal{T}^{\pi_{V_1}} \otimes \mathcal{T}^{\pi_{V_2}} = \mathcal{T}^{\pi_{V_1 \cup V_2}}$.

Recovering $\mathcal{T}(\Pi)$ from the atomic projections

- By repeated application of the corollary, we can recover all pattern database abstractions of an SAS$^+$ planning task from the abstractions for atomic projections.
- In particular, by computing the product of all atomic projections, we can recover the abstraction for the identity abstraction id $= \pi_V$.

Corollary (Recovering $\mathcal{T}(\Pi)$ from the atomic projections)

Let Π be an SAS$^+$ planning task with variable set V.
Then $\mathcal{T}(\Pi) = \bigotimes_{v \in V} \mathcal{T}^{\pi_{\{v\}}}$.

- This is an important result because it shows that the abstractions for atomic projections contain all information of an SAS$^+$ task.

Generic merge-and-shrink abstractions: outline

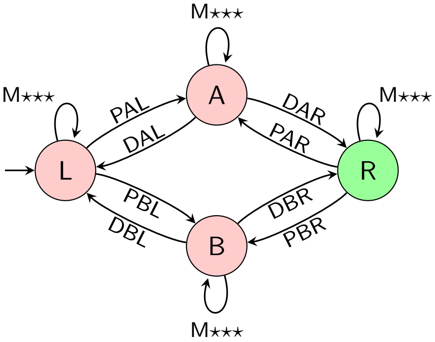
Using the results from the previous section, we can develop the ideas of a generic abstraction computation procedure that takes all state variables into account:

- Initialization step: Compute all abstract transition systems for atomic projections to form the initial abstraction collection.
- Merge steps: Combine two abstractions in the collection by replacing them with their synchronized product. (Stop once only one abstraction is left.)
- Shrink steps: If the abstractions in the collection are too large to compute their synchronized product, make them smaller by abstracting them further (applying an arbitrary homomorphism to them).

We explain these steps with our running example.

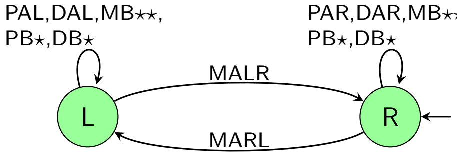
Initialization step: atomic projection for package

$\mathcal{T}^{\pi_{\{\text{package}\}}}$:



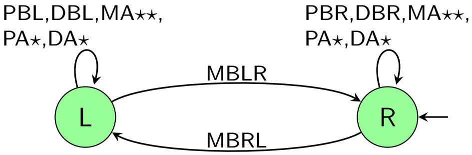
Initialization step: atomic projection for truck A

$\mathcal{T}^{\pi_{\{\text{truck A}\}}}$:



Initialization step: atomic projection for truck B

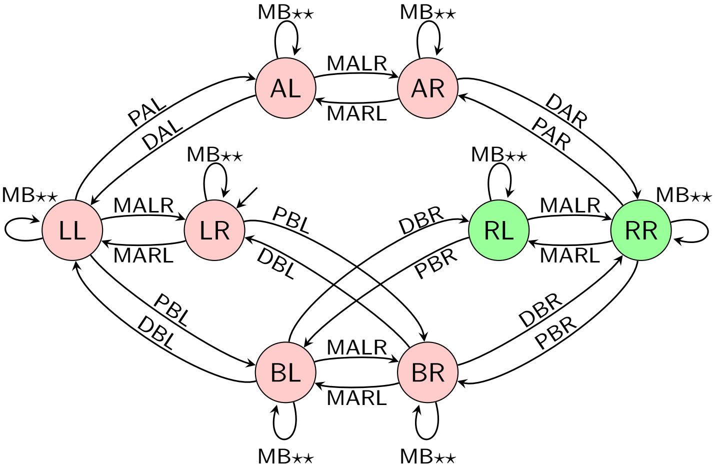
$\mathcal{T}^{\pi_{\{\text{truck B}\}}}$:



current collection: $\{\mathcal{T}^{\pi_{\{\text{package}\}}}, \mathcal{T}^{\pi_{\{\text{truck A}\}}}, \mathcal{T}^{\pi_{\{\text{truck B}\}}}\}$

First merge step

$\mathcal{T}_1 := \mathcal{T}^{\pi_{\{\text{package}\}}} \otimes \mathcal{T}^{\pi_{\{\text{truck A}\}}}$:



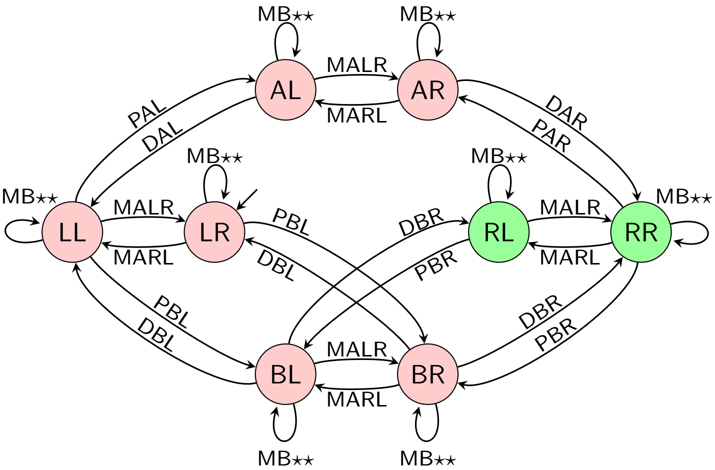
current collection: $\{\mathcal{T}_1, \mathcal{T}^{\pi_{\{\text{truck B}\}}}\}$

Need to simplify?

- If we have sufficient memory available, we can now compute $\mathcal{T}_1 \otimes \mathcal{T}^{\pi_{\{\text{truck B}\}}}$, which would recover the complete transition system of the task.
- However, to illustrate the general idea, let us assume that we do not have sufficient memory for this product.
- More specifically, we will assume that after each product operation we need to reduce the result abstraction to four states to obey memory constraints.
- So we need to reduce $\mathcal{T}_1$ to four states. We have a lot of leeway in deciding how exactly to abstract $\mathcal{T}_1$.
- In this example, we simply use an abstraction that leads to a good result in the end.

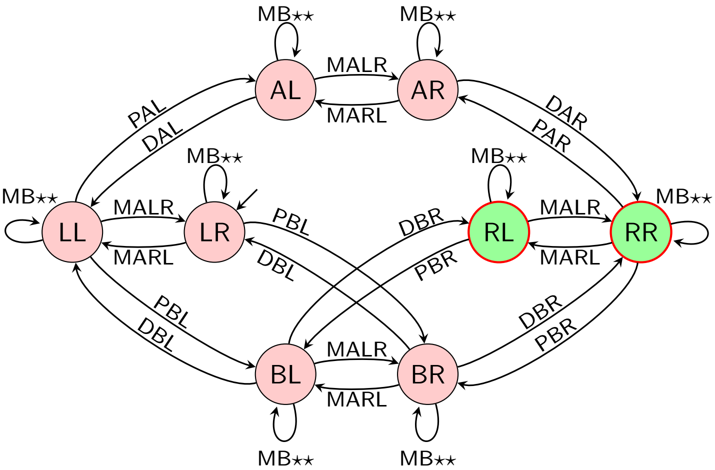
First shrink step

$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



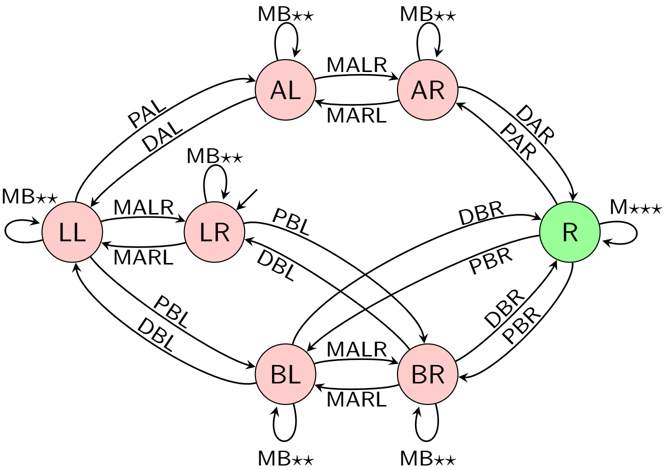
First shrink step

$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



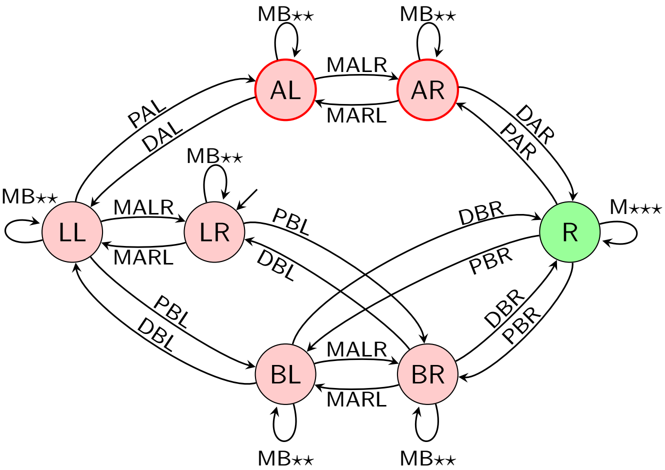
First shrink step

$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



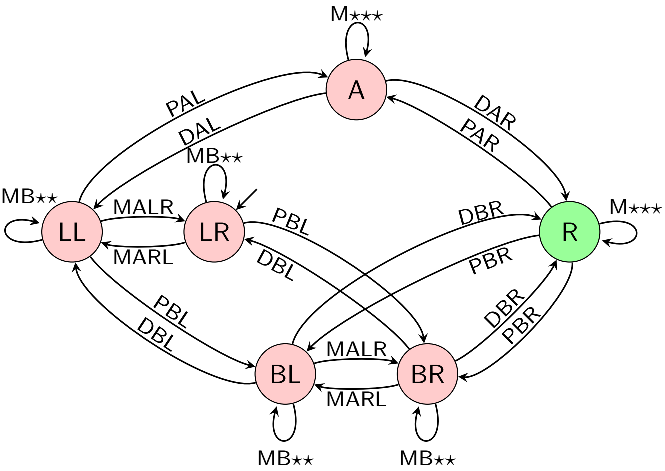
First shrink step

$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



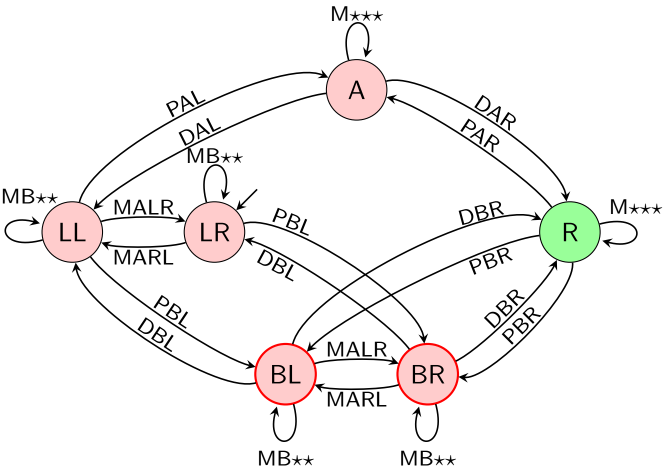
First shrink step

$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



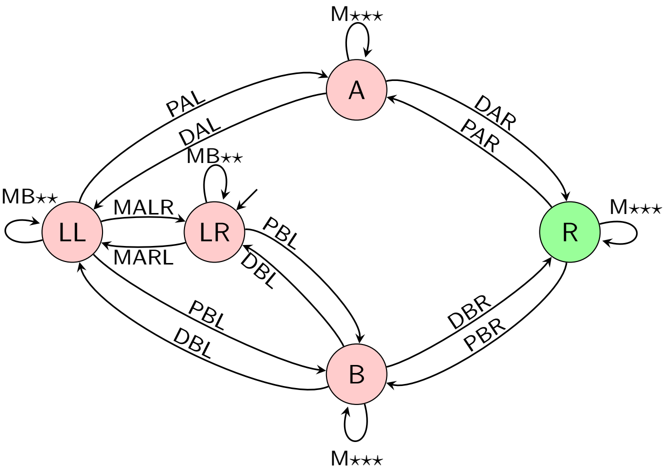
First shrink step

$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



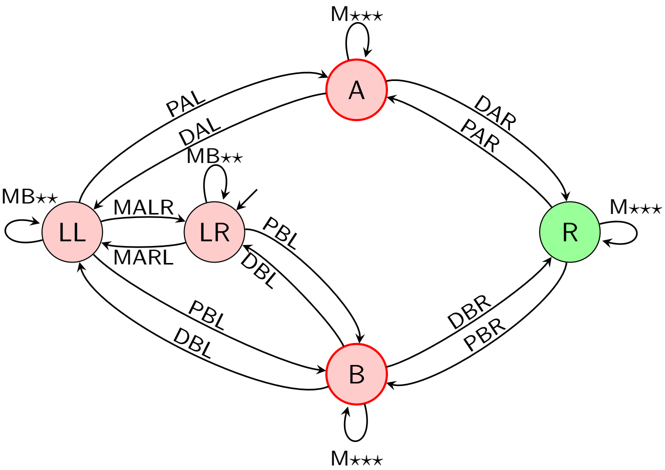
First shrink step

$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



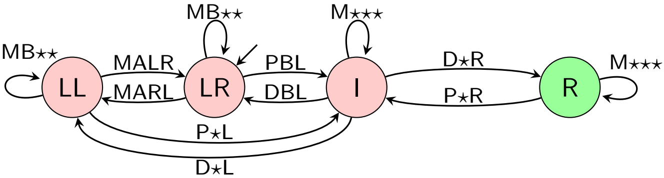
First shrink step

$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



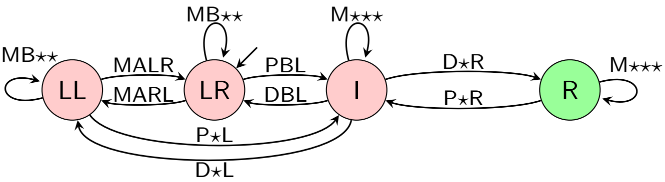
First shrink step

$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



First shrink step

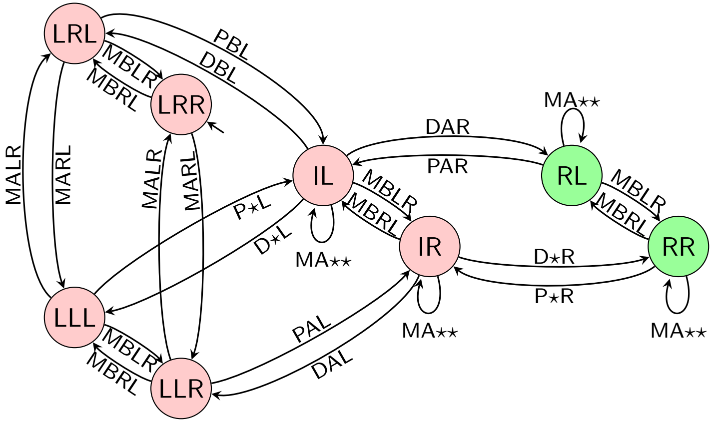
$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



current collection: $\{\mathcal{T}_2, \mathcal{T}^{\pi_{\{\text{truck B}\}}}\}$

Second merge step

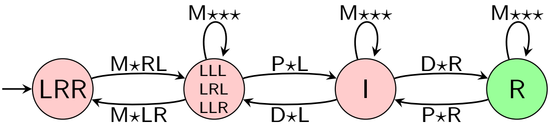
$\mathcal{T}_3 := \mathcal{T}_2 \otimes \mathcal{T}^{\pi_{\{\text{truck B}\}}}$:



current collection: $\{\mathcal{T}_3\}$

Another shrink step?

- Normally we could stop now and use the distances in the final abstraction as our heuristic function.
- However, if there were further state variables to integrate, we would simplify further, e. g. leading to the following abstraction (again with four states):



- We get a heuristic value of 3 for the initial state, better than any PDB heuristic that is a proper abstraction.
- The example generalizes to more locations and trucks, even if we stick to the size limit of 4 (after merging).

How to represent the abstraction mapping?

Idea: the computation of the abstraction mapping follows the sequence of product computations

- For the atomic abstractions for $\pi_{\{v\}}$, we generate a one-dimensional table that denotes which value in $\mathcal{D}_v$ corresponds to which abstract state.
- During the merge (product) step $\mathcal{A} := \mathcal{A}_1 \otimes \mathcal{A}_2$, we generate a two-dimensional table that denotes which pair of states of $\mathcal{A}_1$ and $\mathcal{A}_2$ corresponds to which state of $\mathcal{A}$.
- During the shrink (abstraction) steps, we make sure that the simplified table stays in sync with each individual merge step.

How to represent the abstraction mapping? (ctd.)

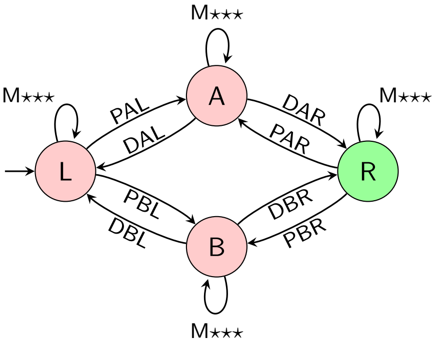
Idea: the computation of the abstraction mapping follows the sequence of product computations

- Once we have computed the final abstraction, we compute all abstract goal distances and store them in a one-dimensional table.
- At this point, we can throw away all the abstractions – we just need to keep the tables.
- During search, we do a sequence of table lookups to navigate from the atomic abstraction states to the final abstraction state and heuristic value $\rightsquigarrow 2|V|$ lookups, $O(|V|)$ time

Again, we illustrate the process with our running example.

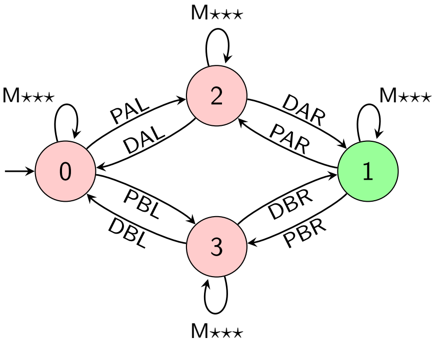
Abstraction mapping example: atomic abstractions

Computing abstraction mappings for the atomic abstractions is simple. Just number the states (domain values) consecutively and generate a table of references to the states:



Abstraction mapping example: atomic abstractions

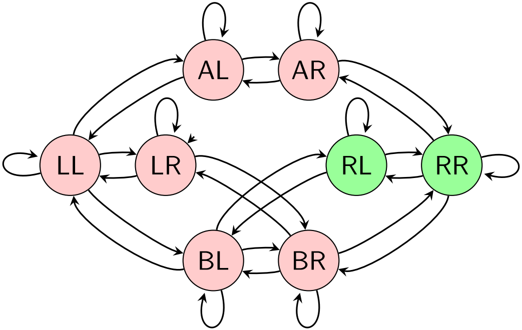
Computing abstraction mappings for the atomic abstractions is simple. Just number the states (domain values) consecutively and generate a table of references to the states:



L	R	A	B
0	1	2	3

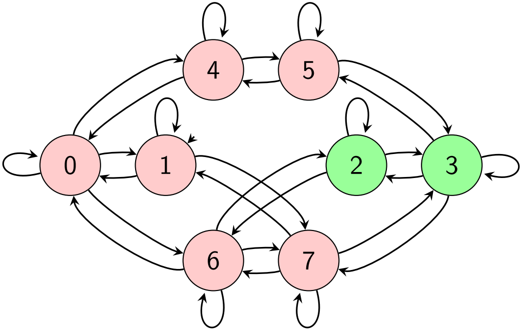
Abstraction mapping example: merge step

For product abstractions $\mathcal{A}_1 \otimes \mathcal{A}_2$, we again number the product states consecutively and generate a table that links state pairs of $\mathcal{A}_1$ and $\mathcal{A}_2$ to states of $\mathcal{A}$:



Abstraction mapping example: merge step

For product abstractions $\mathcal{A}_1 \otimes \mathcal{A}_2$, we again number the product states consecutively and generate a table that links state pairs of $\mathcal{A}_1$ and $\mathcal{A}_2$ to states of $\mathcal{A}$:



	$s_2 = 0$	$s_2 = 1$
$s_1 = 0$	0	1
$s_1 = 1$	2	3
$s_1 = 2$	4	5
$s_1 = 3$	6	7

Maintaining the mapping when shrinking

- The hard part in representing the abstraction mapping is to keep it consistent when shrinking.
- In theory, this is easy to do:
  - When combining states i and j, arbitrarily use one of them (say i) as the number of the new state.
  - Find all table entries in the table for this abstraction which map to the other state j and change them to i.
- However, doing a table scan each time two states are combined is very inefficient.
- Fortunately, there also is an efficient implementation which takes constant time per combination.

Towards a concrete algorithm

- We have now described how merge-and-shrink abstractions work in general.
- However, we have not said how exactly to decide
  - which abstractions to combine in a merge step and
  - when and how to further abstract in a shrink step.
- There are many possibilities here (just like there are many possible PDB heuristics).
- Only one concrete method, called h_{HHH}, has been explored so far in planning, which we will now discuss briefly.

Generic algorithm template

Generic abstraction computation algorithm

abs $:= \{\mathcal{T}^{\pi_{\{v\}}} \mid v \in V\}$
while *abs* contains more than one abstraction:
  select $\mathcal{A}_1$, $\mathcal{A}_2$ from *abs*
  shrink $\mathcal{A}_1$ and/or $\mathcal{A}_2$ until $\mathit{size}(\mathcal{A}_1) \cdot \mathit{size}(\mathcal{A}_2) \leq N$
  abs $:=$ *abs* $\setminus \{\mathcal{A}_1, \mathcal{A}_2\} \cup \{\mathcal{A}_1 \otimes \mathcal{A}_2\}$
return the remaining abstraction in *abs*

N: parameter bounding number of abstract states

Questions for practical implementation:

- Which abstractions to select? $\rightsquigarrow$ merging strategy
- How to shrink an abstraction? $\rightsquigarrow$ shrinking strategy
- How to choose N? $\rightsquigarrow$ usually: as high as memory allows

Merging strategy

Which abstractions to select?

h_{HHH}: Linear merging strategy

In each iteration after the first, choose the abstraction computed in the previous iteration as $\mathcal{A}_1$.
$\rightsquigarrow$ fully defined by an ordering of atomic projections

Rationale: only maintains one "complex" abstraction at a time

h_{HHH}: Ordering of atomic projections

- Start with a goal variable.
- Add variables that appear in preconditions of operators affecting previous variables.
- If that is not possible, add a goal variable.

Rationale: increases h quickly

Shrinking strategy

Which abstractions to shrink?

h_{HHH}: only shrink the product

If at all possible, don't shrink atomic abstractions, but only product abstractions.

Rationale: Product abstractions are more likely to contain significant redundancies and symmetries.

Shrinking strategy (ctd.)

How to shrink an abstraction?

h_{HHH}: f-preserving shrinking strategy

Repeatedly combine abstract states with identical abstract goal distances (h values) and identical abstract initial state distances (g values).

Rationale: preserves heuristic value and overall graph shape

h_{HHH}: Tie-breaking criterion

Prefer combining states where $g + h$ is high.
In case of ties, combine states where h is high.

Rationale: states with high $g + h$ values are less likely to be explored by A*, so inaccuracies there matter less

Outline

1. Abstractions informally
2. Abstractions formally
3. Projection abstractions (PDBs)
4. Merge-and-shrink abstractions
5. Generalized additive heuristics
6. Structural-pattern abstractions

Transition systems of SAS$^+$ planning tasks
Extension

Definition (transition system of an SAS$^+$ planning task)

Let $\Pi = \langle V, I, O, G \rangle$ be an SAS$^+$ planning task.
The transition system of Π, in symbols $\mathcal{T}(\Pi)$, is the transition system $\mathcal{T}(\Pi) = \langle S, L, T, I, G \rangle$, where

- S is the set of states over V,
- $L = O$,
- $T = \{\langle s, o, t \rangle \in S \times L \times S \mid \mathit{app}_o(s) = t\}$,
- $I = I$, and
- $G = \{s \in S \mid s \models G\}$.

Transition systems of SAS$^+$ planning tasks
Extension

Definition (transition system of an SAS$^+$ planning task)

Let $\Pi = \langle V, I, O, G, cost \rangle$ be an SAS$^+$ planning task with $cost : O \rightarrow \mathbb{R}^{0+} \cup \{\infty\}$.

The transition system of Π, in symbols $\mathcal{T}(\Pi)$, is the transition system $\mathcal{T}(\Pi) = \langle S, L, T, I, G \rangle$, where

- S is the set of states over V,
- $L = O$,
- $T = \{\langle s, o, t \rangle \in S \times L \times S \mid \mathit{app}_o(s) = t\}$,
- $I = I$, and
- $G = \{s \in S \mid s \models G\}$.

In short: labels of $\mathcal{T}(\Pi)$ are getting annotated with operator costs in Π.

Orthogonality of abstraction mappings
Reminder

Definition (orthogonal abstraction mappings)

Let α_1 and α_2 be abstraction mappings on $\mathcal{T}$.

We say that α_1 and α_2 are orthogonal if for all transitions $\langle s, l, t\rangle$ of $\mathcal{T}$, we have $\alpha_i(s) = \alpha_i(t)$ for at least one $i \in \{1, 2\}$.

What if α_1 and α_2 are non-orthogonal?

Definition (orthogonal action counting)

Let $\Pi = \langle V, I, O, G, cost\rangle$ be an SAS$^+$ planning task, and $\mathcal{T}_1$ and $\mathcal{T}_2$ be two abstractions of $\mathcal{T}(\Pi)$.

We say that action counting in $\mathcal{T}_1$ and $\mathcal{T}_2$ is orthogonal if for all operators $o \in O$, we have $cost_i(o) = 0$ for at least one $i \in \{1, 2\}$.

Orthogonality of action counting

Definition (orthogonal abstraction mappings)

Let α_1 and α_2 be abstraction mappings on $\mathcal{T}$.

We say that α_1 and α_2 are orthogonal if for all transitions $\langle s, l, t \rangle$ of $\mathcal{T}$, we have $\alpha_i(s) = \alpha_i(t)$ for at least one $i \in \{1, 2\}$.

What if α_1 and α_2 are non-orthogonal?

Definition (orthogonal action counting)

Let $\Pi = \langle V, I, O, G, cost \rangle$ be an SAS$^+$ planning task, and $\mathcal{T}_1$ and $\mathcal{T}_2$ be two abstractions of $\mathcal{T}(\Pi)$.

We say that action counting in $\mathcal{T}_1$ and $\mathcal{T}_2$ is orthogonal if for all operators $o \in O$, we have $cost_i(o) = 0$ for at least one $i \in \{1, 2\}$.

Action counting orthogonality and additivity

Theorem (additivity for orthogonal abstraction mappings)

Let $h^{\mathcal{T}_1,\alpha_1}, \ldots, h^{\mathcal{T}_n,\alpha_n}$ be abstraction heuristics for the same planning task Π such that action counting in $\mathcal{T}_i$ and $\mathcal{T}_j$ is orthogonal for all $i \neq j$.
Then $\sum_{i=1}^{n} h^{\mathcal{T}_i,\alpha_i}$ is a safe, goal-aware, admissible and consistent heuristic for Π.

What next?

1. Can we further generalize this (sufficient) condition for additivity?
2. If so, can it be practical?

Action counting orthogonality and additivity

Theorem (additivity for orthogonal abstraction mappings)

Let $h^{\mathcal{T}_1,\alpha_1}, \ldots, h^{\mathcal{T}_n,\alpha_n}$ be abstraction heuristics for the same planning task Π such that action counting in $\mathcal{T}_i$ and $\mathcal{T}_j$ is orthogonal for all $i \neq j$.
Then $\sum_{i=1}^{n} h^{\mathcal{T}_i,\alpha_i}$ is a safe, goal-aware, admissible and consistent heuristic for Π.

What next?

1. Can we further generalize this (sufficient) condition for additivity?
2. If so, can it be practical?

Additive sets of heuristics

Theorem (action cost partitioning)

Let $\Pi, \Pi_1, \ldots, \Pi_k$ be planning tasks, identical except for the operator costs $cost, cost_1, \ldots, cost_k$. Let $\{h_i\}_{i=1}^k$ be a set of arbitrary *admissible heuristic functions for $\{\Pi_i\}_{i=1}^k$, respectively.*

If holds $cost(o) \geq \sum_{i=1}^k cost_i(o)$ for all operators o, then $\sum_{i=1}^k h_i$ is an admissible heuristic for Π.

Additive sets of heuristics

Theorem (action cost partitioning)

Let $\Pi, \Pi_1, \ldots, \Pi_k$ *be planning tasks, identical except for the operator costs* $cost, cost_1, \ldots, cost_k$*. Let* $\{h_i\}_{i=1}^{k}$ *be a set of* arbitrary *admissible heuristic functions for* $\{\Pi_i\}_{i=1}^{k}$*, respectively.*

If holds $cost(o) \geq \sum_{i=1}^{k} cost_i(o)$ *for all operators* o*, then* $\sum_{i=1}^{k} h_i$ *is an admissible heuristic for* Π*.*

Observations

- Generalizes action counting orthogonality
- No idea what partition is better? $\rightsquigarrow$ Uniform partition?

Additive sets of heuristics

Theorem (action cost partitioning)

Let $\Pi, \Pi_1, \ldots, \Pi_k$ *be planning tasks, identical except for the operator costs* $cost, cost_1, \ldots, cost_k$*. Let* $\{h_i\}_{i=1}^k$ *be a set of* arbitrary *admissible heuristic functions for* $\{\Pi_i\}_{i=1}^k$*, respectively.*

If holds $cost(o) \geq \sum_{i=1}^k cost_i(o)$ *for all operators* o*, then* $\sum_{i=1}^k h_i$ *is an admissible heuristic for* Π*.*

Observations

- Generalizes action counting orthogonality
- No idea what partition is better? $\rightsquigarrow$ Uniform partition?
- Still, how to choose among the alternative cost partitions?

Optimal action cost partitioning

Problem statement

Given

1. a (costs attached) transition system $\mathcal{T}$,
2. a set of (costs attached) abstractions $\{\mathcal{T}_i\}_{i=1}^k$ of $\mathcal{T}$ with abstraction mappings $\{\alpha_i\}_{i=1}^k$, respectively, and
3. a state s in $\mathcal{T}$,

determine optimal additive heuristic for $\mathcal{T}$ on the basis of $\{\mathcal{T}_i\}_{i=1}^k$, that is

$$h_{\mathsf{opt}}(s) = \max_{\{cost_i\}} \sum_{i=1}^{k} h_i^*(\alpha_i(s)).$$

Problems on the way

Optimal additive heuristic for $\mathcal{T}$ on the basis of $\{\mathcal{T}_i\}_{i=1}^k$

$$h_{\mathsf{opt}}(s) = \max_{\{cost_i\}} \sum_{i=1}^{k} h_i^*(\alpha_i(s)).$$

Challenges

1. Infinite space of alternative choices $\{cost_i\}_{i=1}^k$
2. The optimal choice is state-dependent
3. The process is fully unsupervised

The LP trick

Main Idea

Instead of, given an action cost partition $\{cost_i\}_{i=1}^{k}$, independently searching each abstraction $\mathcal{T}_i$ using dynamic programming

1. compile SSSP problem over each $\mathcal{T}_i$ into a linear program $\mathcal{L}_i$ with action costs being free variables
2. combine $\mathcal{L}_1, \ldots, \mathcal{L}_k$ with additivity constraints $cost(o) \geq \sum_{i=1}^{k} cost_i(a)$
3. solution of the joint LP $\leadsto h_{\text{opt}}(s)$

The LP trick

Main Idea

Instead of, given an action cost partition $\{cost_i\}_{i=1}^{k}$, independently searching each abstraction $\mathcal{T}_i$ using dynamic programming

1. compile SSSP problem over each $\mathcal{T}_i$ into a linear program $\mathscr{L}_i$ with action costs being free variables
2. combine $\mathscr{L}_1, \ldots, \mathscr{L}_k$ with additivity constraints $cost(o) \geq \sum_{i=1}^{k} cost_i(a)$
3. solution of the joint LP $\rightsquigarrow$ $h_{\text{opt}}(s)$

Single-Source Shortest Paths: LP Formulation

LP formulation

Given: digraph $G = (N, E)$, source node $v \in N$

LP variables: $d(v') \rightsquigarrow$ shortest-path length from v to v'

LP:

$$\max_{\vec{d(\cdot)}} \sum_{v'} d(v')$$

$$\text{s.t. } d(v) = 0$$

$$d(v'') \leq d(v') + w(v', v''), \;\; \forall (v', v'') \in E$$

Step 1: Compile each SSSP over $\mathcal{T}_i$ into $\mathscr{L}_i$

LP formulation

Given: abstraction $\mathcal{T}_i$, state s of concrete system $\mathcal{T}$

LP variables: $\{d(s') \mid s' \in S_i\} \cup \{d(G_i)\} \cup \{cost(o,i)\}$

LP:

$$\max \quad d(G_i)$$

$$\text{s.t.} \quad \begin{cases} d(s') \leq d(s'') + cost(o,i), & \forall \langle s', o, s'' \rangle \in \mathcal{T}_i \\ d(s') = 0, & s' = \alpha_i(s) \\ d(G_i) \leq d(s'), & s' \in G(i) \end{cases}$$

Step 2: Properly combine $\{\mathscr{L}_i\}_{i=1}^k$

LP formulation

Given: abstractions $\{\mathcal{T}_i\}_{i=1}^k$ state s of $\mathcal{T}$

LP variables: $\bigcup_{i=1}^k \{d(s') \mid s' \in S_i\} \cup \{d(G_i)\} \cup \{cost(o,i)\}$

LP:

$$\max \sum_{i=1}^{k} d(G_i)$$

$$\text{s.t. } \forall i \begin{cases} d(s') \leq d(s'') + cost(o,i), & \forall \langle s', o, s'' \rangle \in \mathcal{T}_i \\ d(s') = 0, & s' = \alpha_i(s) \\ d(G_i) \leq d(s'), & s' \in G(i) \end{cases}$$

$$\forall o \in O : \quad cost(o) \geq \sum_{i=1}^{k} cost(o,i)$$

Outline

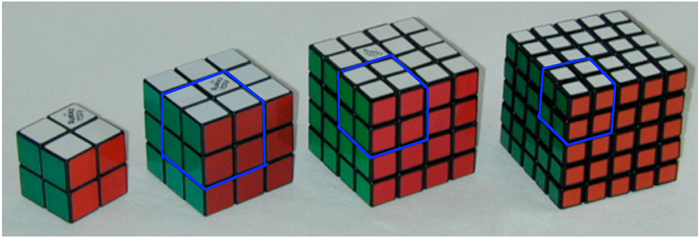
Limitations of Explicit Abstractions

Both PDBs and merge-and-shrink are explicit abstractions: abstract spaces are searched exhaustively

PDBs dimensionality $= O(1)$, size of the abstract space is $O(1)$

M&S dimensionality $= \Theta(|V|)$, size of the abstract space is $O(1)$

$\rightsquigarrow$ (often) price in heuristic accuracy in long-run



Abstractions: Extending the definition

Definition (abstraction, abstraction mapping)

Let $\mathcal{T} = \langle S, L, T, I, G, \rangle$ and $\mathcal{T}' = \langle S', L', T', I', G', \rangle$ be transition systems with the same label set $L = L'$, , and let $\alpha : S \rightarrow S'$.

We say that $\mathcal{T}'$ is an abstraction of $\mathcal{T}$ with abstraction mapping α if

- for all $s \in I$, we have $\alpha(s) \in I'$,
- for all $s \in G$, we have $\alpha(s) \in G'$, and
- for all $\langle s, l, t \rangle \in T$, we have $\langle \alpha(s), l, \alpha(t) \rangle \in T'$.

Abstractions: Extending the definition

Definition (abstraction, abstraction mapping)

Let $\mathcal{T} = \langle S, L, T, I, G, \mathcal{C} \rangle$ and $\mathcal{T}' = \langle S', L', T', I', G', \mathcal{C}' \rangle$ be transition systems with the same label set $L = L'$, $\mathcal{C} : S \rightarrow \mathbb{R}^{0+}$, $\mathcal{C}' : S' \rightarrow \mathbb{R}^{0+}$, and let $\alpha : S \rightarrow S'$.

We say that $\mathcal{T}'$ is an abstraction of $\mathcal{T}$ with abstraction mapping α if

- for all $s \in I$, we have $\alpha(s) \in I'$,
- for all $s \in G$, we have $\alpha(s) \in G'$, and
- for all $\langle s, l, t \rangle \in T$, we have $h^*(\alpha(s), \alpha(t)) \leq \mathcal{C}(l)$.

Structural Abstraction Heuristics: Main Idea

Objective (departing from PDBs)

Instead of perfectly reflecting a few state variables,
reflect many (up to $\Theta(|V|)$) state variables, BUT

- guarantee abstract space can be searched (implicitly) in poly-time

Structural Abstraction Heuristics: Main Idea

Objective (departing from PDBs)

Instead of perfectly reflecting a few state variables, reflect many (up to $\Theta(|V|)$) state variables, BUT

- ♠ guarantee abstract space can be searched (implicitly) in poly-time

How

Abstracting Π by an instance of a tractable fragment of cost-optimal planning

- ☹ not many such known tractable fragments
- ☺ should find more, and useful for us!

Structural Abstraction Heuristics: Main Idea

Objective (departing from PDBs)

Instead of perfectly reflecting a few state variables, reflect many (up to $\Theta(|V|)$) state variables, BUT

- ♠ guarantee abstract space can be searched (implicitly) in poly-time

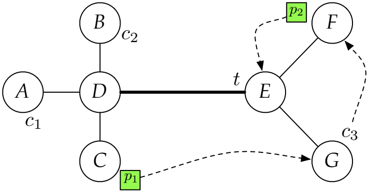
How

Abstracting Π by an instance of a tractable fragment of cost-optimal planning

- ☹ not many such known tractable fragments
- ☺ should find more, and useful for us!

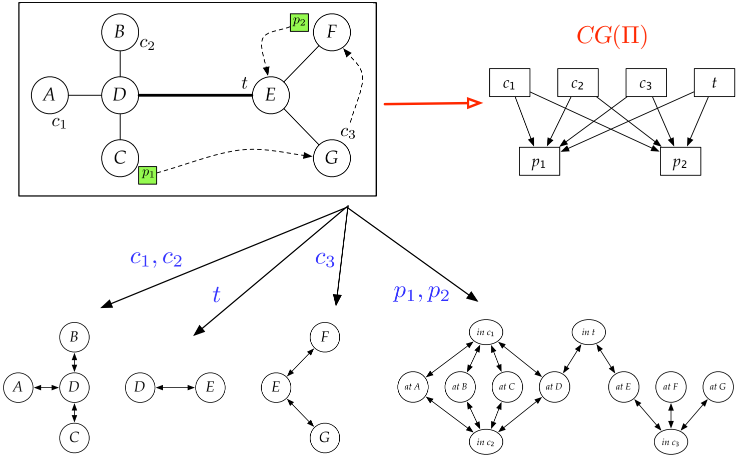
Here Come the Forks!

Running Example

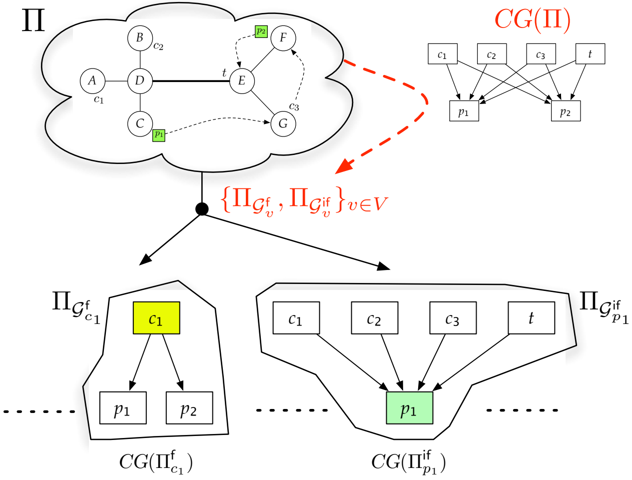


$$
\begin{aligned}
V &= \{p_1, p_2, c_1, c_2, c_3, t\} \\
dom(p_1) &= dom(p_2) = \{A, B, C, D, E, F, G, c_1, c_2, c_3, t\} \\
dom(c_1) &= dom(c_2) = \{A, B, C, D\} \\
dom(c_3) &= \{E, F, G\} \\
dom(t) &= \{D, E\} \\
s^0, G &\mapsto \text{see picture} \\
A &\mapsto \text{loads, unloads, single-segment movements}
\end{aligned}
$$

Causal Graph + Domain Transition Graphs



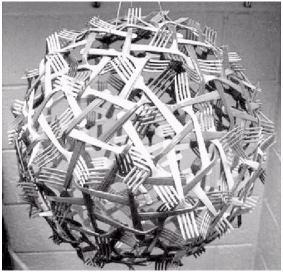
Fork-Decomposition (Additive Abstractions)



+ ensuring proper action cost partitioning

Action Cost Partitioning = Gluing Things Together

$\rightsquigarrow$



Works?
Problem!

Forks and Inverted Forks are Hard …

- ☹ Even non-optimal planning for problems with fork and inverted fork causal graphs is NP-complete (D & Dinitz, 2001).
- ☹ Even if the domain-transition graphs of all variables are strongly connected, optimal planning for forks and inverted forks remains NP-hard (Helmert, 2003-04).

⇝ Shall we give up?

Tractable Cases of Planning with Forks

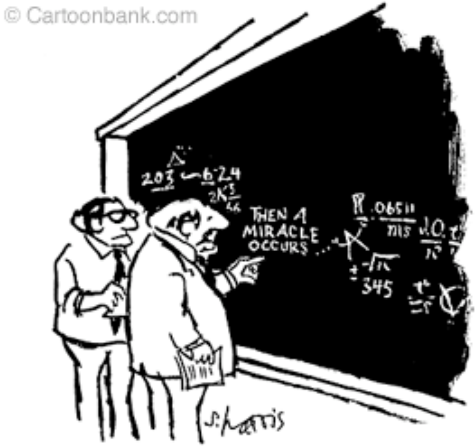
Theorem (forks)

Cost-optimal planning for fork problems with root $r \in V$ is poly-time if $|dom(r)| = 2$.

Theorem (inverted forks)

Cost-optimal planning for inverted fork problems with root $r \in V$ is poly-time if $|dom(r)| = O(1)$.

Tractable Cases of Planning with Forks



"I think you should be more explicit here in step two."

Theorem (inverted forks)

Theorem (inverted forks)

Cost-optimal planning for inverted fork problems with root $r \in V$ *is poly-time if* $|dom(r)| = d = O(1)$.

Proof sketch (Construction)

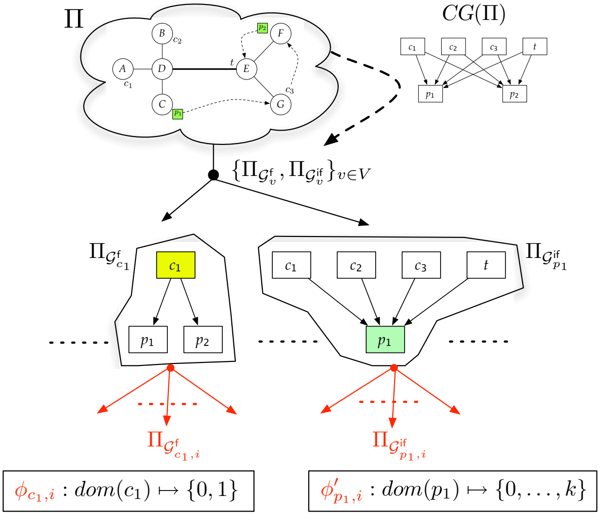
(1) Create all $\Theta(d^d)$ cycle-free paths from $s^0[r]$ to $G[r]$ in $DTG(r, \Pi)$.

(2) For each $u \in \text{pred}(r)$, and each $x, y \in dom(u)$, compute the cost-minimal path from x to y in $DTG(u, \Pi)$.

(3) For each path in $DTG(r, \Pi)$ generated in step (1), construct a plan for Π based on that path for r, and the shortest paths computed in (2).

(4) Take minimal cost plan from (3).

Mixing Causal-Graph & Variable-Domain Decompositions



+ ensuring proper action cost partitioning

Planning / Logistics-00
Expanded nodes

#	h^*	HHH_{10^5} nodes	time	$h^{\mathcal{F}}$ nodes	time	$h^{\mathcal{FI}}$ + opt nodes	time
01	20	21	0.05	21	10.49	21	20.82
02	19	20	0.04	20	10.4	20	20.36
03	15	16	0.05	16	5.18	16	10.85
04	27	28	0.33	28	22.81	28	47.42
05	17	18	0.34	18	11.72	18	21.63
06	8	9	0.33	9	2.99	9	8.89
07	25	26	1.11	26	26.88	26	53.81
08	14	15	1.12	15	10.37	15	21.19
09	25	26	1.14	26	27.78	26	51.52
10	36	37	4.55	37	426.07	37	973.46
11	44	2460	4.65	1689	14259.8	45	1355.23
12	31	32	6.5	32	374.48	32	876.9
13	44	7514	6.84	45	702.29	45	1621.74
14	36	37	8.94	37	474.8	37	1153.85
15	30	31	8.84	31	448.86	31	1052.46
16	45	29319	17.35	46	3517.25	46	7635.96
17	42	1561610	45.61	43	3297.69	43	7192.51
18	48	199428	24.95			49	10014.3
19	60					61	15625.5
20	42	6095	24.9	43	4325.45	43	9470.85
21	68					69	22928.4

Planning / Logistics-00
Expanded nodes and Time

#	h^*	*HHH*$_{10^5}$		$h^{\mathcal{F}}$		$h^{\mathcal{FI}}$ + opt	
		nodes	*time*	*nodes*	*time*	*nodes*	*time*
01	20	21	*0.05*	21	*10.49*	21	*20.82*
02	19	20	*0.04*	20	*10.4*	20	*20.36*
03	15	16	*0.05*	16	*5.18*	16	*10.85*
04	27	28	*0.33*	28	*22.81*	28	*47.42*
05	17	18	*0.34*	18	*11.72*	18	*21.63*
06	8	9	*0.33*	9	*2.99*	9	*8.89*
07	25	26	*1.11*	26	*26.88*	26	*53.81*
08	14	15	*1.12*	15	*10.37*	15	*21.19*
09	25	26	*1.14*	26	*27.78*	26	*51.52*
10	36	37	*4.55*	37	*426.07*	37	*973.46*
11	44	2460	*4.65*	1689	*14259.8*	45	*1355.23*
12	31	32	*6.5*	32	*374.48*	32	*876.9*
13	44	7514	*6.84*	45	*702.29*	45	*1621.74*
14	36	37	*8.94*	37	*474.8*	37	*1153.85*
15	30	31	*8.84*	31	*448.86*	31	*1052.46*
16	45	29319	*17.35*	46	*3517.25*	46	*7635.96*
17	42	1561610	*45.61*	43	*3297.69*	43	*7192.51*
18	48	199428	*24.95*			49	*10014.3*
19	60					61	*15625.5*
20	42	6095	*24.9*	43	*4325.45*	43	*9470.85*
21	68					69	*22928.4*

Planning / Logistics-00

Shall we redefine the notion of success?...

#	h^*	HHH_{10^5}		$h^{\mathcal{F}}$			$h^{\mathcal{FI}}$ + opt	
		nodes	*time*	*nodes*	*time*	♠	*nodes*	*time*
01	20	21	*0.05*	21	*10.49*		21	*20.82*
02	19	20	*0.04*	20	*10.4*		20	*20.36*
03	15	16	*0.05*	16	*5.18*		16	*10.85*
04	27	28	*0.33*	28	*22.81*		28	*47.42*
05	17	18	*0.34*	18	*11.72*		18	*21.63*
06	8	9	*0.33*	9	*2.99*		9	*8.89*
07	25	26	*1.11*	26	*26.88*		26	*53.81*
08	14	15	*1.12*	15	*10.37*		15	*21.19*
09	25	26	*1.14*	26	*27.78*		26	*51.52*
10	36	37	*4.55*	37	*426.07*		37	*973.46*
11	44	2460	*4.65*	1689	*14259.8*		45	*1355.23*
12	31	32	*6.5*	32	*374.48*		32	*876.9*
13	44	7514	*6.84*	45	*702.29*		45	*1621.74*
14	36	37	*8.94*	37	*474.8*		37	*1153.85*
15	30	31	*8.84*	31	*448.86*		31	*1052.46*
16	45	29319	*17.35*	46	*3517.25*		46	*7635.96*
17	42	1561610	*45.61*	43	*3297.69*		43	*7192.51*
18	48	199428	*24.95*				49	*10014.3*
19	60						61	*15625.5*
20	42	6095	*24.9*	43	*4325.45*		43	*9470.85*
21	68						69	*22928.4*

Planning / Logistics-00

No. Structural pattern databases!

#	h^*	HHH_{10^5}		$h^{\mathcal{F}}$			$h^{\mathcal{FJ}}$ + opt	
		nodes	*time*	*nodes*	*time*	♠	*nodes*	*time*
01	20	21	*0.05*	21	*10.49*	*0.27*	21	*20.82*
02	19	20	*0.04*	20	*10.4*	*0.27*	20	*20.36*
03	15	16	*0.05*	16	*5.18*	*0.27*	16	*10.85*
04	27	28	*0.33*	28	*22.81*	*0.33*	28	*47.42*
05	17	18	*0.34*	18	*11.72*	*0.33*	18	*21.63*
06	8	9	*0.33*	9	*2.99*	*0.33*	9	*8.89*
07	25	26	*1.11*	26	*26.88*	*0.41*	26	*53.81*
08	14	15	*1.12*	15	*10.37*	*0.43*	15	*21.19*
09	25	26	*1.14*	26	*27.78*	*0.41*	26	*51.52*
10	36	37	*4.55*	37	*426.07*	*3.96*	37	*973.46*
11	44	2460	*4.65*	1689	*14259.8*	*4.25*	45	*1355.23*
12	31	32	*6.5*	32	*374.48*	*4.68*	32	*876.9*
13	44	7514	*6.84*	45	*702.29*	*4.63*	45	*1621.74*
14	36	37	*8.94*	37	*474.8*	*5.12*	37	*1153.85*
15	30	31	*8.84*	31	*448.86*	*5.12*	31	*1052.46*
16	45	29319	*17.35*	46	*3517.25*	*24.73*	46	*7635.96*
17	42	1561610	*45.61*	43	*3297.69*	*24.13*	43	*7192.51*
18	48	199428	*24.95*	697		*24.73*	49	*10014.3*
19	60			21959		*33.61*	61	*15625.5*
20	42	6095	*24.9*	43	*4325.45*	*29.61*	43	*9470.85*
21	68			106534		*61.54*	69	*22928.4*

Empirical Evaluation

domain	solved	$h^{\mathcal{F}}$	$h^{\mathcal{J}}$	$h^{\mathcal{FJ}}$	MS_{10^4}	MS_{10^5}	HSP^*_F	Gamer	blind	$h_{\max}$
airport	20	16	17	16	16	16	15	11	17	20
blocks	30	21	18	18	18	20	30	30	18	18
depots	7	**7**	4	4	7	4	4	4	4	4
driverlog	12	11	**12**	11	12	12	9	11	7	8
freecell	5	**5**	4	4	5	1	5	2	4	5
grid	2	1	1	1	2	2	0	2	1	2
gripper	20	7	7	7	7	7	6	20	7	7
logistics	22	**22**	16	16	16	21	16	20	10	10
logistics	7	**6**	4	5	4	5	3	6	2	2
miconic	85	51	50	50	54	55	45	85	50	50
mprime	25	21	18	21	21	12	8	9	19	24
mystery	20	**20**	16	**20**	16	12	11	8	17	17
openstacks	7	**7**	**7**	**7**	7	7	7	7	7	7
pathways	4	**4**	**4**	**4**	3	4	4	4	4	4
pipes-notank	22	14	15	14	20	12	13	11	14	17
pipes-tank	14	10	9	7	13	7	7	6	10	10
psr-small	50	48	49	48	50	50	50	47	48	49
rovers	7	6	**7**	6	6	7	6	5	5	6
satellite	6	**6**	**6**	**6**	6	6	5	6	4	5
schedule	44	**43**	34	39	20	0	11	3	28	30
tpp	6	**6**	**6**	**6**	6	6	5	5	5	6
trucks	9	6	7	7	6	5	9	3	5	7
zenotravel	11	**11**	**11**	**11**	11	11	8	10	7	8
solved	435	**349**	322	328	326	282	277	315	293	316