Abstractions in Planning: Pattern Databases & Merge and Shrink

Stefan Edelkamp

PUI - CTU

AI CENTER
FEE CTU

Coming up with a heuristic in a principled way

General procedure for obtaining a heuristic

Solve an easier version of the problem.

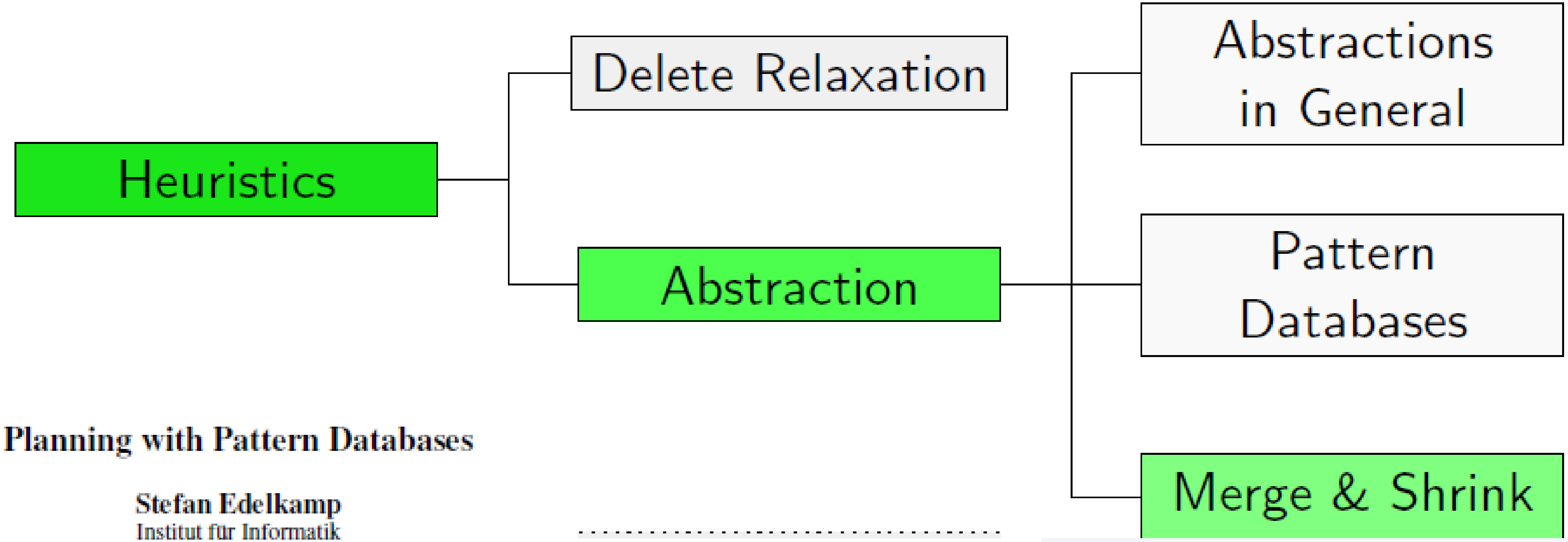
Two common methods:

- 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.

Some heuristics in AI Planning



Planning with Pattern Databases

Stefan Edelkamp
Institut für Informatik
Albert-Ludwigs-Universität
Georges-Köhler-Allee, Gebäude 51
D-79110 Freiburg
eMail: *edelkamp@informatik.uni-freiburg.de**

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	

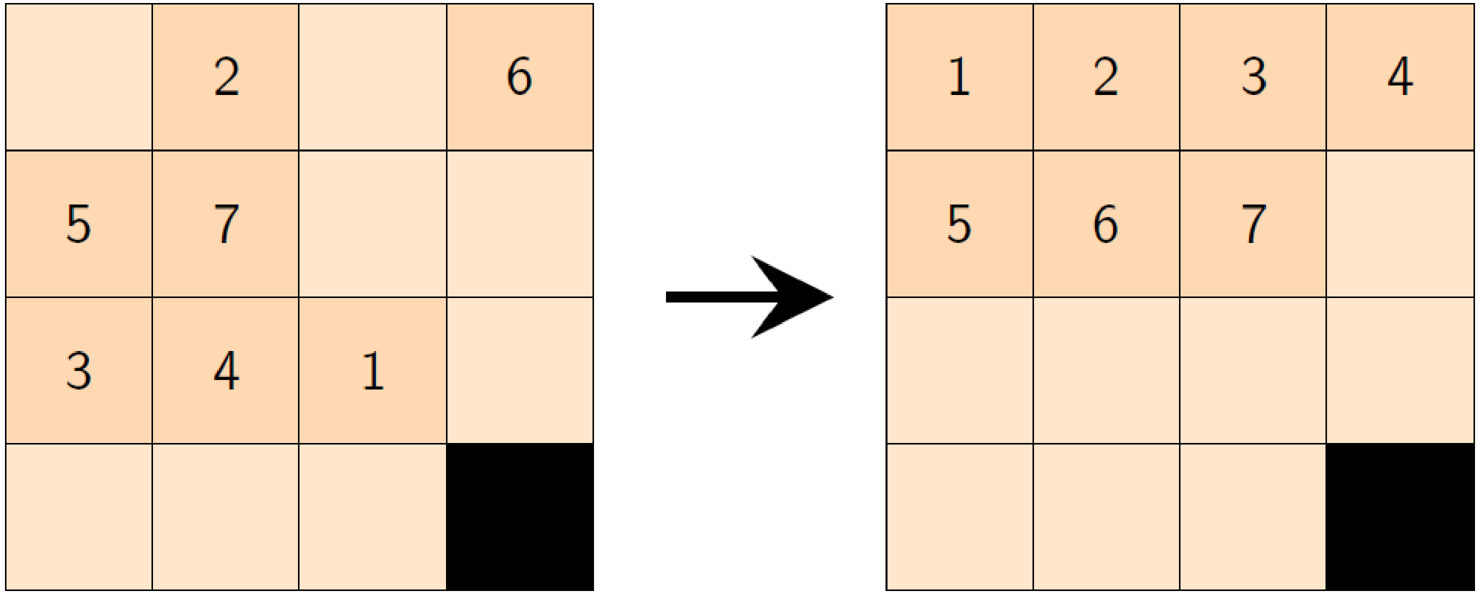
→

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

Korf‘s conjecture

- n: number of states in the entire problem space
- b: brute-force branching factor of the space
- d: be the average optimal solution length for a random problem instance
- e: be the expected value of the heuristic
- m: the amount of memory unsed, in terms of heuristic values stored
- t: running time of IDA*, in terms of nodes generated

The average optimal solution length d of a random instance, which is the depth to which IDA* must serach, can be estimated as $\log_b n$ or $d \approx \log_b n$. We expect $e \approx \log_b m$ and $t \approx b^{d-e}$

Substituting the values for d and e into this formula gives:

$$t \approx b^{d-e} \approx b^{\log_b n - \log_b m} = n/m$$

Abstract Planning Problem

An *abstract planning problem* $\mathcal{P}|_R = < \mathcal{S}|_R, \mathcal{O}|_R, \mathcal{I}|_R, \mathcal{G}|_R >$ of a propositional planning problem $< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$ with respect to a set of propositional atoms R is defined by

1. $\mathcal{S}|_R = \{S|_R \mid S \in \mathcal{S}\}$,
2. $\mathcal{G}|_R = \{G|_R \mid G \in \mathcal{G}\}$,
3. $\mathcal{O}|_R = \{O|_R \mid O \in \mathcal{O}\}$, with $O|_R$ for $O = (P, A, D) \in \mathcal{O}$ is given as $(P|_R, A|_R, D|_R)$

Sequential solutions for the abstract planning problem $\mathcal{P}|_R$ are denoted by π_R and optimal abstract sequential plan length is denoted by δ_R.

Pattern Databases for Strips

A planning pattern database $\mathcal{D}_R$ with respect to a set of propositions R and a propositional planning problem $< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$ is a collection of pairs (v, S) with $v \in I\!R$ and $S \in \mathcal{S}|_R$, such that $v = \delta_R(S)$

Therefore,

$$\mathcal{D}_R = \{(\delta_R(S), S) \mid S \in \mathcal{S}|_R\}.$$

An optimal sequential abstract plan π_R^{opt} for $\mathcal{P}|_R$ is always shorter than an optimal sequential plan π^{opt} for $\mathcal{P}$, i.e. $\delta_R(S|_R) \leq \delta(S)$, for all $S \in \mathcal{S}$

Proof

Let $\pi = (O_1, \ldots, O_k)$ be a sequential plan for $< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$. Then $\pi|_R = (O_1|_R, \ldots, O_k|_R)$ is a solution for $\mathcal{P}|_R = < \mathcal{S}|_R, \mathcal{O}|_R, \mathcal{I}|_R, \mathcal{G}|_R >$.

Now suppose, that $\delta_R(S|_R) > \delta(S)$ for some $S \in \mathcal{S}$

Let $\pi^{opt} = (O_1, \ldots, O_t)$ be the optimal sequential plan from S to $\mathcal{G}$ in the original planning space $\mathcal{P}$ then

$\pi^{opt}|_R = (O_1|_R, \ldots, O_t|_R)$ is a valid plan in $\mathcal{P}|_R$ with plan length less or equal to $t = \delta(S)$

Contradiction

Remark: Strict inequality $\delta_R(S|_R) < \delta(S)$ is given if some operators $O_i|_R$ are void, or if there are alternative even shorter paths in the abstract space.

AI CENTER FEE CTU

Example: Blocksworld

((clear a),1)
((holding a),2)
((on b a),2)
((on d a),2)

((on d c) (clear b),1) ((on a b) (clear c),1)
((on d c) (holding b),2) ((clear c) (clear b),2)
((on d c) (on d b),2) ((on a b) (holding c),2)
((on a c) (on a b) ,2) ((clear c) (holding b),3)
((clear b) (holding c),3) ((on a c) (clear b),3)
((on d b) (clear c),3) ((holding c) (holding b),4)
((on b c) (clear b),4) ((on a c) (holding b),4)
((on c b) (clear c),4) ((on d b) (holding c),4)
((on a c) (on d b),4) ((on b c) (holding b),5)
((on a b) (on b c),5) ((on d b) (on b c),5)
((on c b) (holding c),5) ((on a c) (on c b),5)
((on c b) (on d c),5)

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)

Practical requirements for abstractions

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.

Some observations

Observation the use of maximized smaller pattern databases reduces the number of nodes generated by IDA*

Eight-puzzle: 20 pattern databases of size 252 perform less state expansions (318) then 1 pattern database of size 5,040 (yielding 2,160 state expansions)

1. The use of smaller pattern databases instead of one large pattern database usually reduces the number of patterns with high h-value, but maximization of the smaller pattern databases can make the number of patterns with low h-values significantly smaller than the number of low-valued patterns in the larger pattern database
2. Eliminating low h values is more important for improving search performance than for retaining large h-values

Maximizing several abstractions: example

Example (15-puzzle)

- mapping to tiles 1–7 was arbitrary
  $\leadsto$ 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

Maximizing 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

$\rightsquigarrow$ Is the sum of the abstraction heuristics admissible?

Maximizing 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
- 2nd abstraction: ignore precise location of 1–7

⤳ The sum of the abstraction heuristics is not admissible.

Maximizing 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.

Transition systems

Definition (transition system)

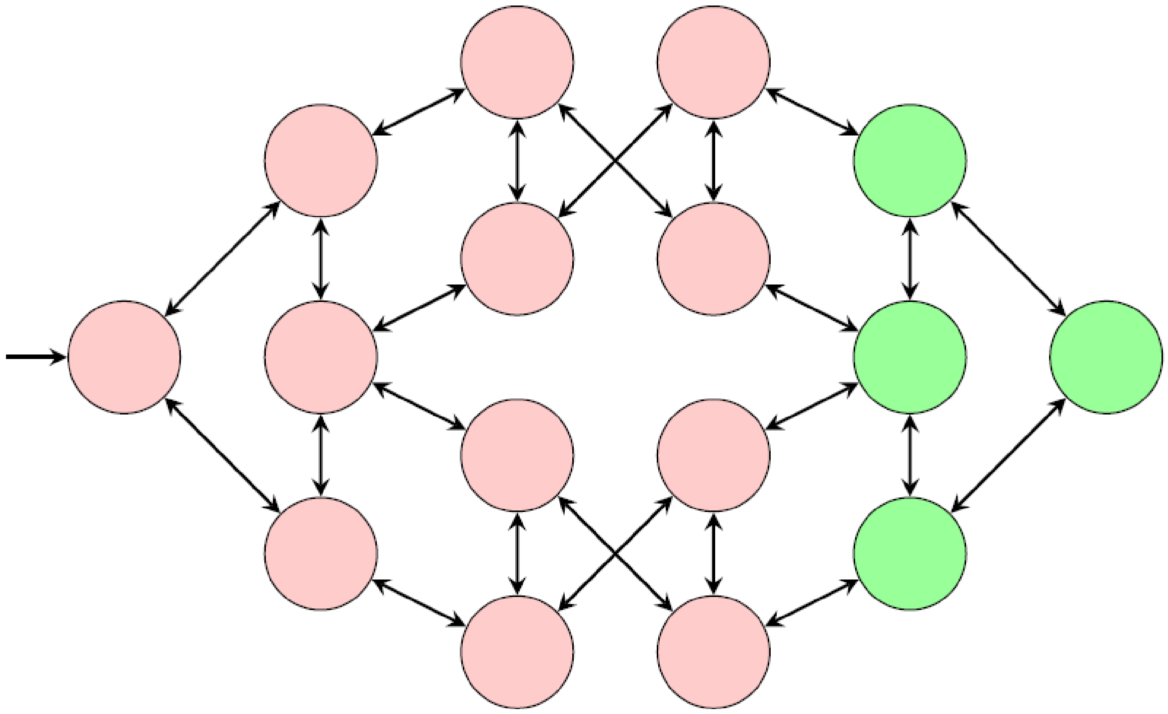
A transition system is a 5-tuple $\mathcal{T} = \langle S, L, T, I, G \rangle$ where

- 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 system: 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.

SAS+ planning task

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
- $G' = \{s' \in S' \mid s' \models G\}$.

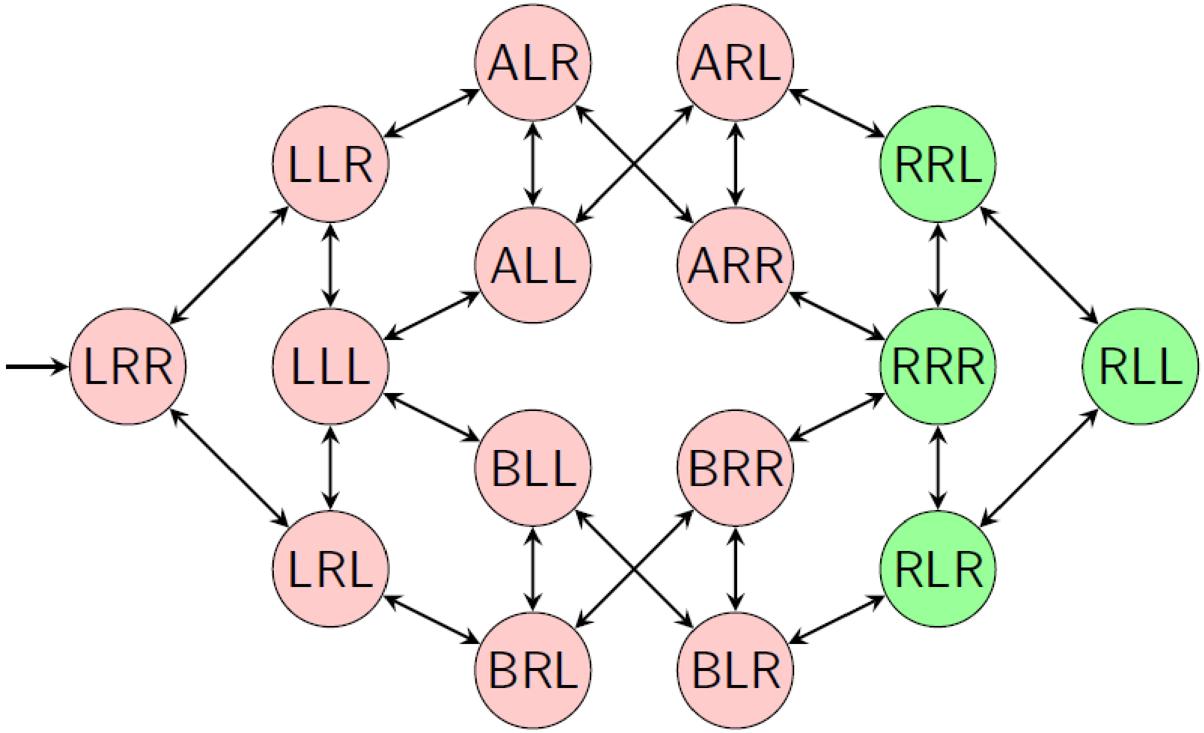
SAS+ planning task: example

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
  - $\mathsf{pickup}_{i,j} = \langle t_i = j \wedge p = j, p := i \rangle$
  - $\mathsf{drop}_{i,j} = \langle t_i = j \wedge p = i, p := j \rangle$
  - $\mathsf{move}_{i,j,j'} = \langle t_i = j, t_i := j' \rangle$
- $G = (p = \mathsf{R})$

Transition system of example task



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

AI CENTER
FEE CTU

Abstraction

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 heuristic

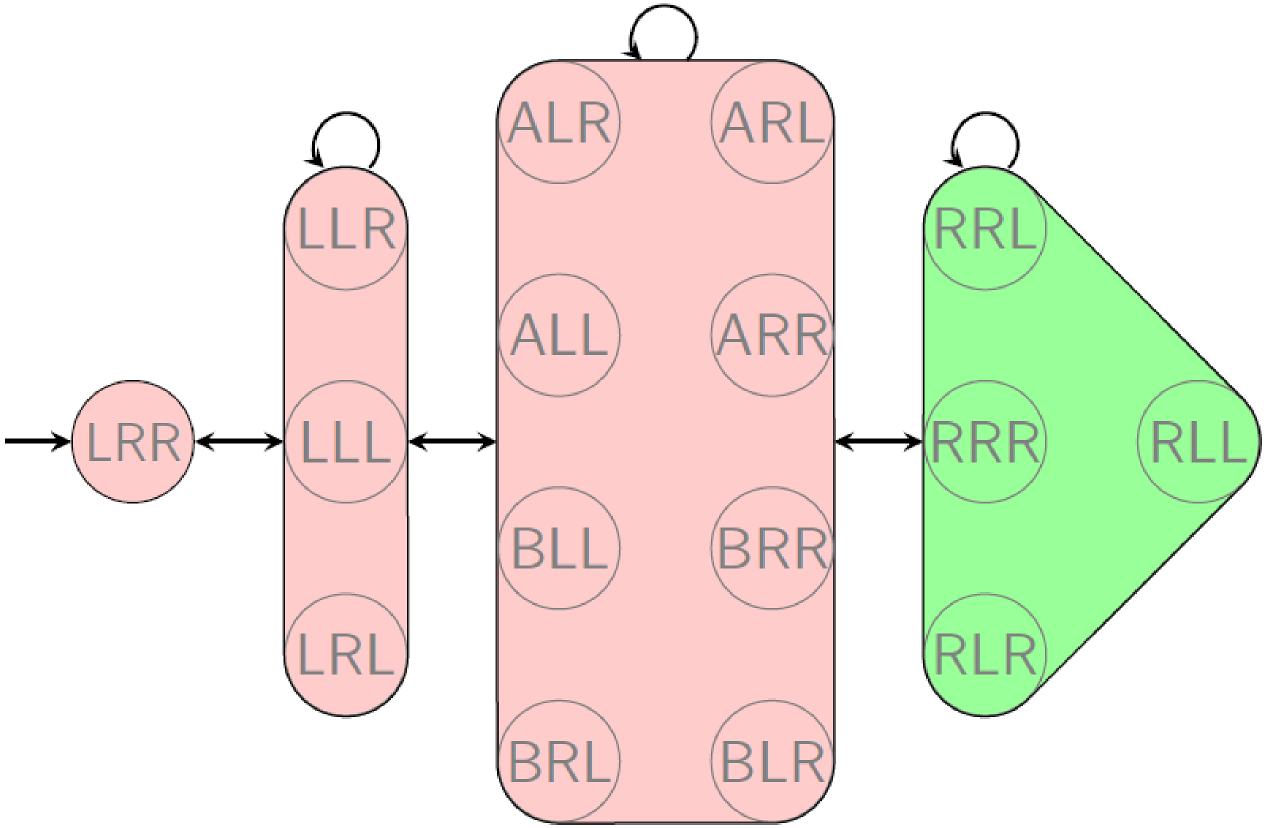
Definition (abstraction heuristic)

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 heuristic: 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 heuristic

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.

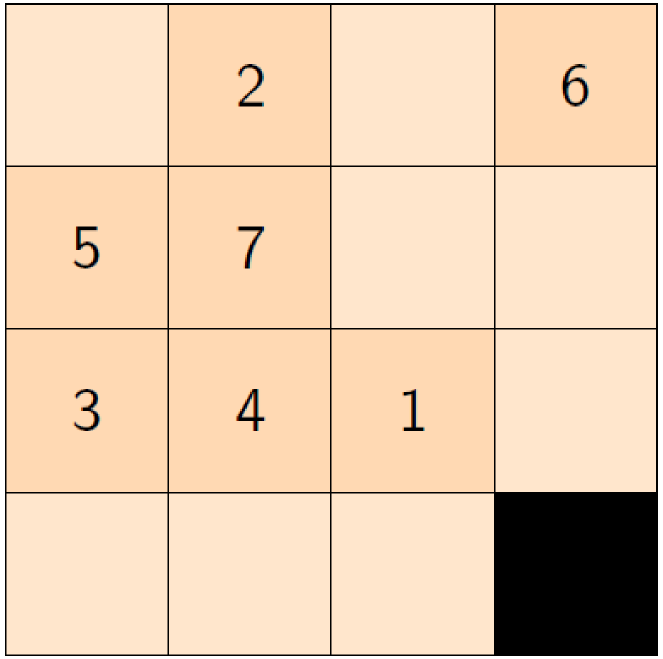
Orthogonal abstraction mapping

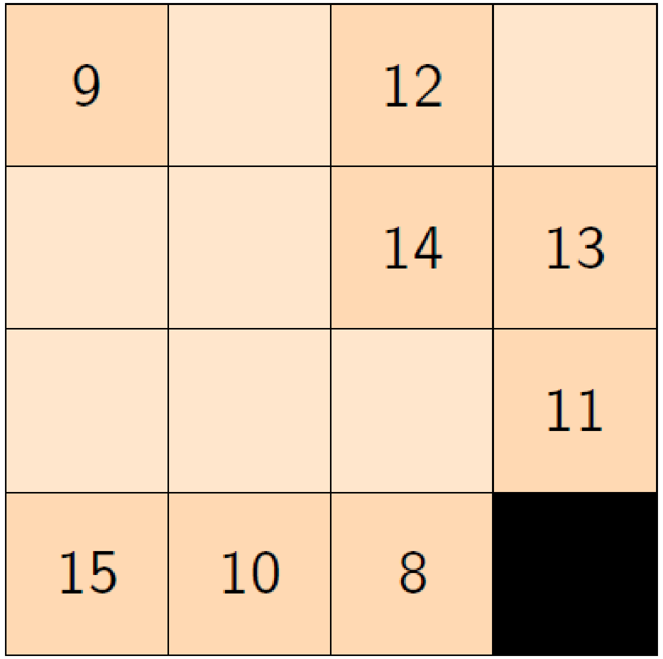
Definition (orthogonal abstraction mappings)

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 $i \in \{1, 2\}$.

Orthogonal abstraction mapping: example





Are the abstraction mappings orthogonal?

Orthogonal abstraction mapping: example

	2		6
5	7		
3	4	1	

9		12	
		14	13
			11
15	10	8	

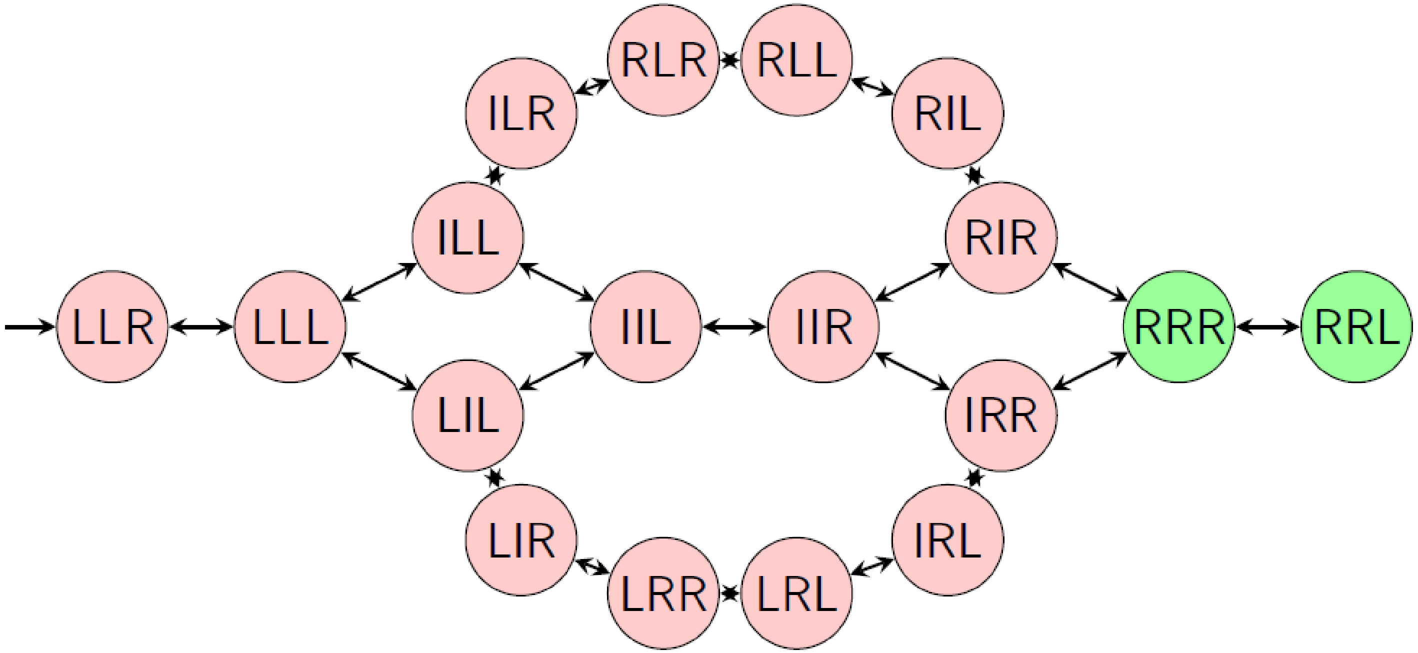
Are the abstraction mappings orthogonal?

Orthogonality and additivity

Theorem (additivity for orthogonal abstraction mappings)

Let $h^{\mathcal{A}_1,\alpha_1}, \ldots, h^{\mathcal{A}_n,\alpha_n}$ *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

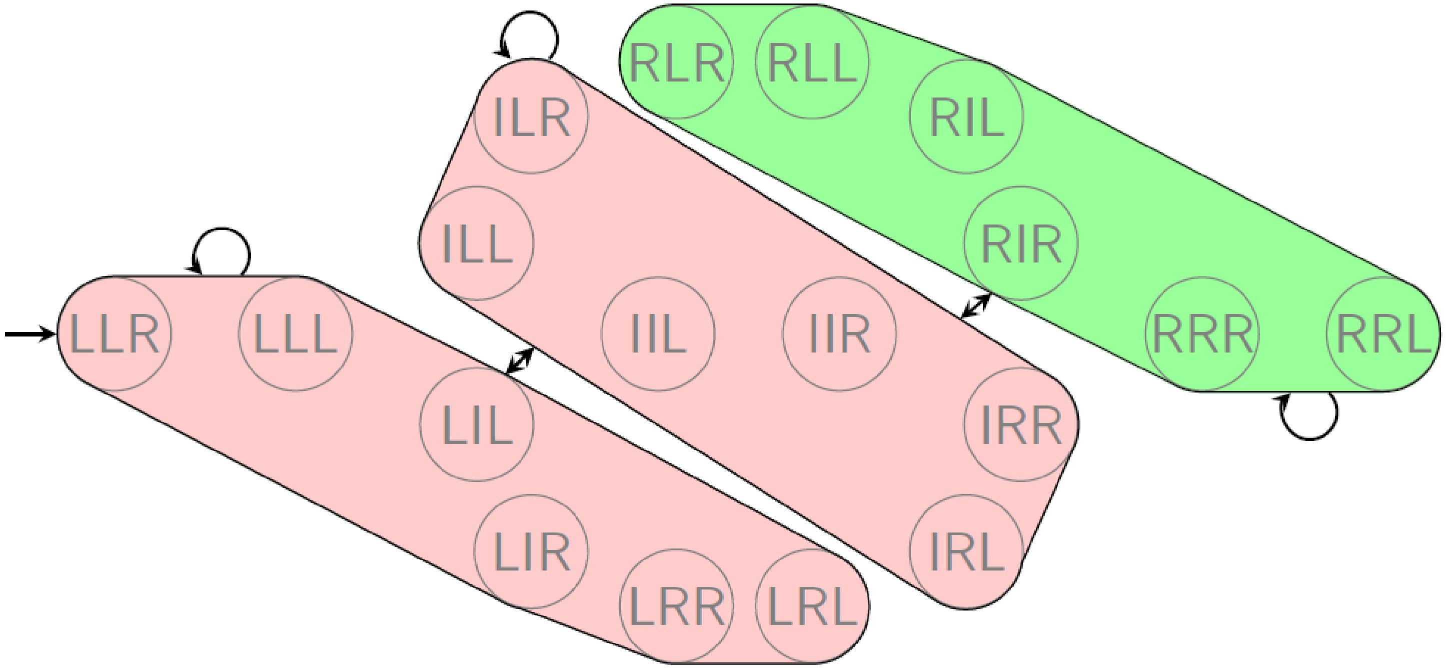


transition system $\mathcal{T}$

state variables: first package, second package, truck

AI CENTER FEE CTU

Orthogonality and additivity: example

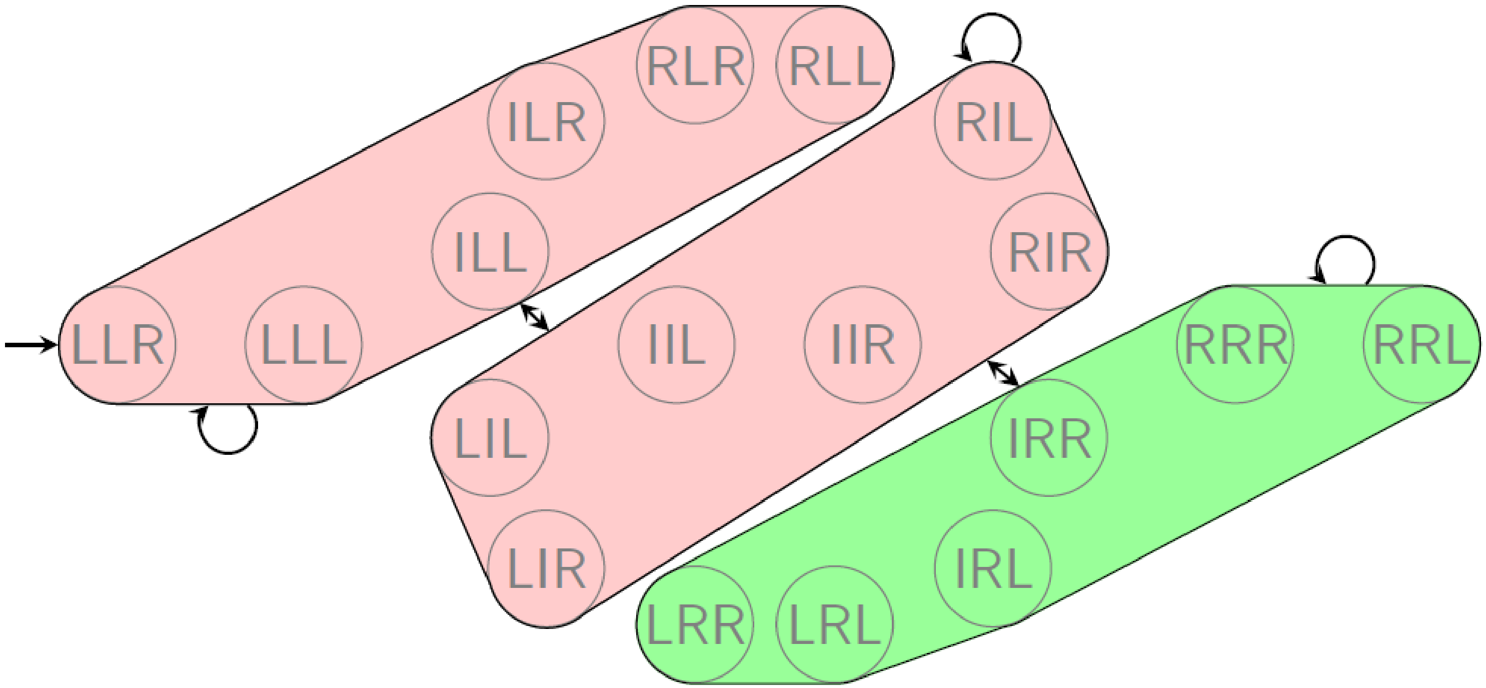


abstraction $\mathcal{A}_1$

mapping: only consider state of first package

AI CENTER FEE CTU

Orthogonality and additivity: example



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

Using abstraction heuristics in practice

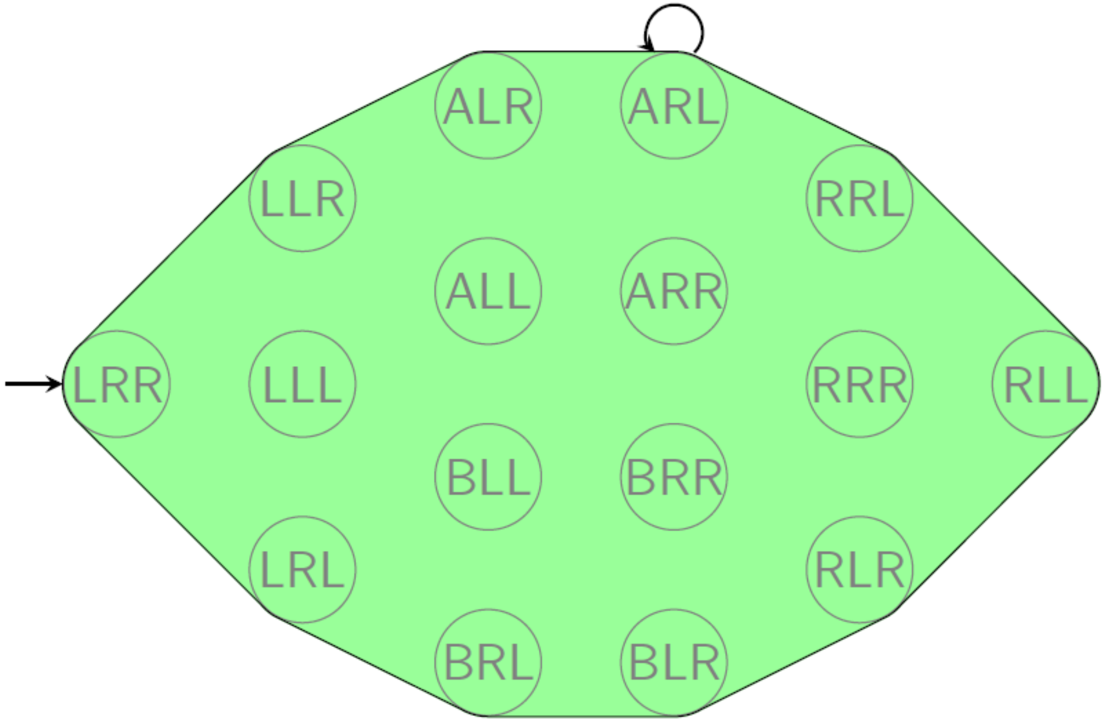
In practice, there are conflicting goals for abstractions:

- 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.

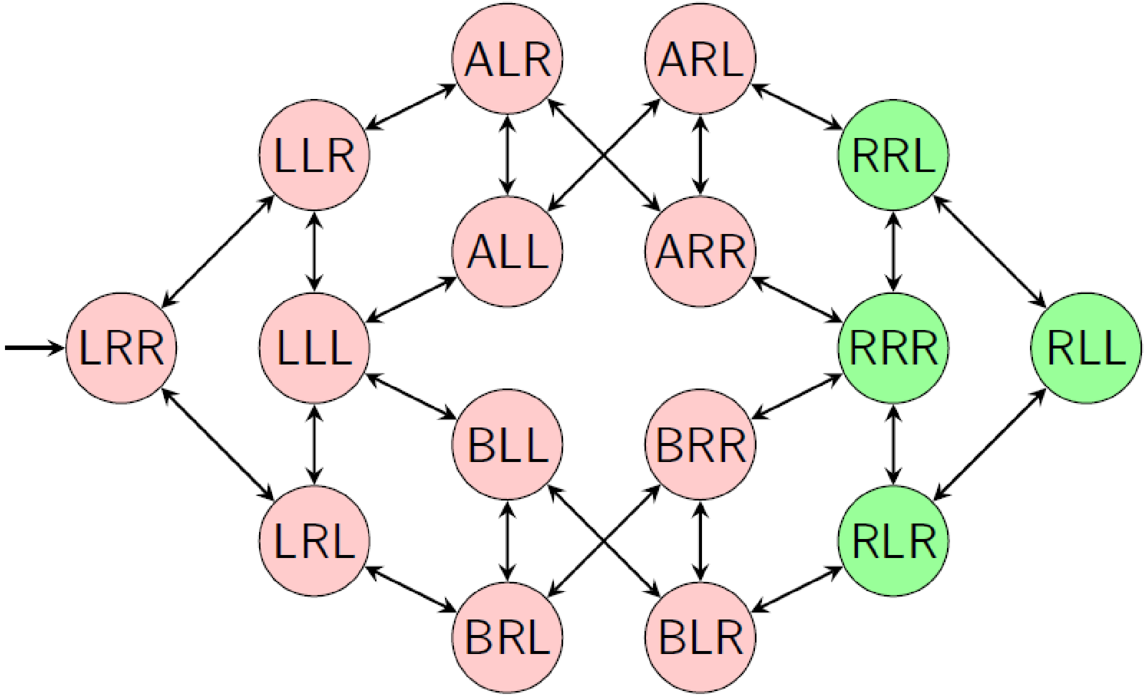
Counterexample: one-state abstraction



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

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

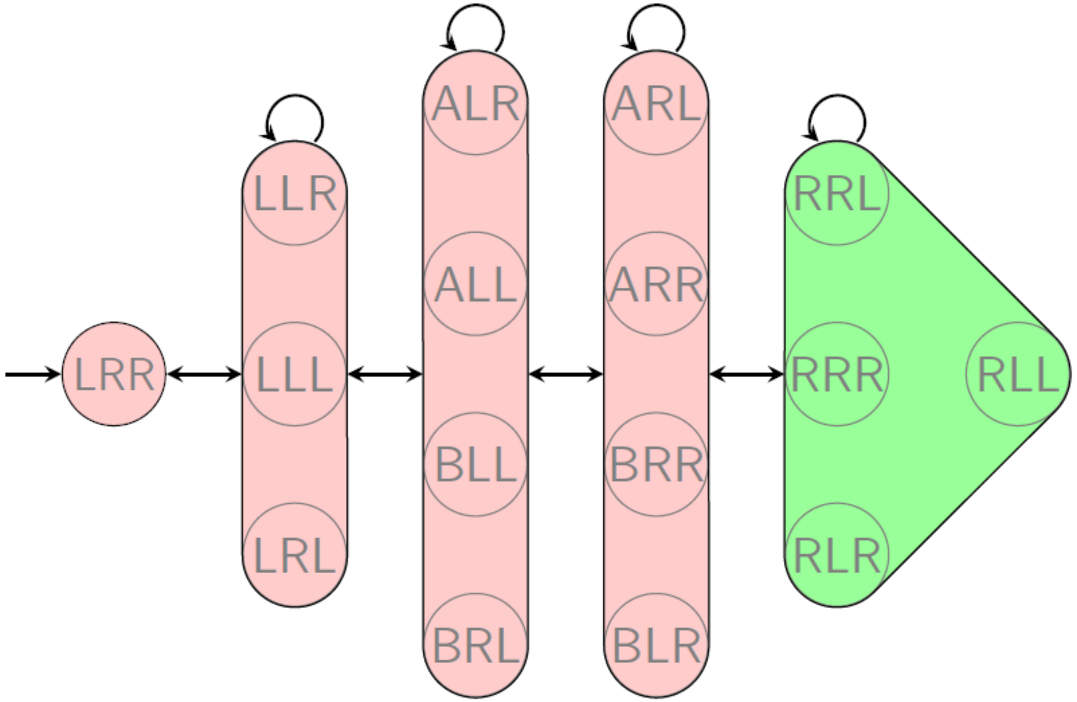
Counterexample: identity abstraction



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

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

Counterexample: perfect heuristic



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

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

Pattern database heuristic 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 $\pi_P : S \to S'$ 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.

PDBs

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

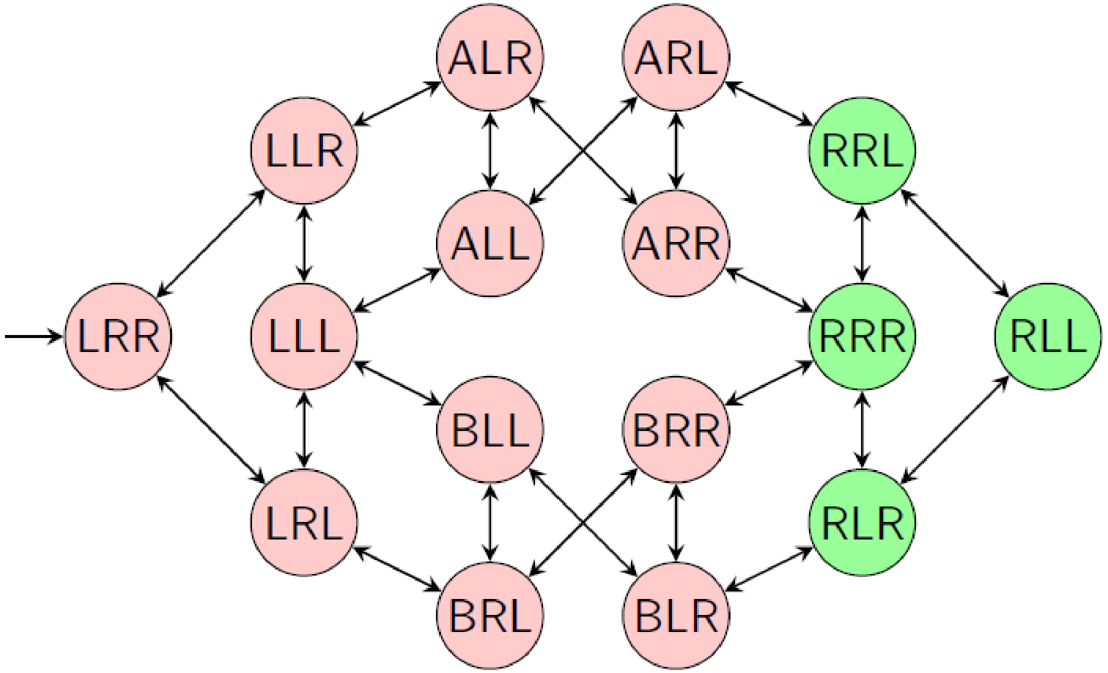
Definition (pattern database heuristic)

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".

PDBs: example

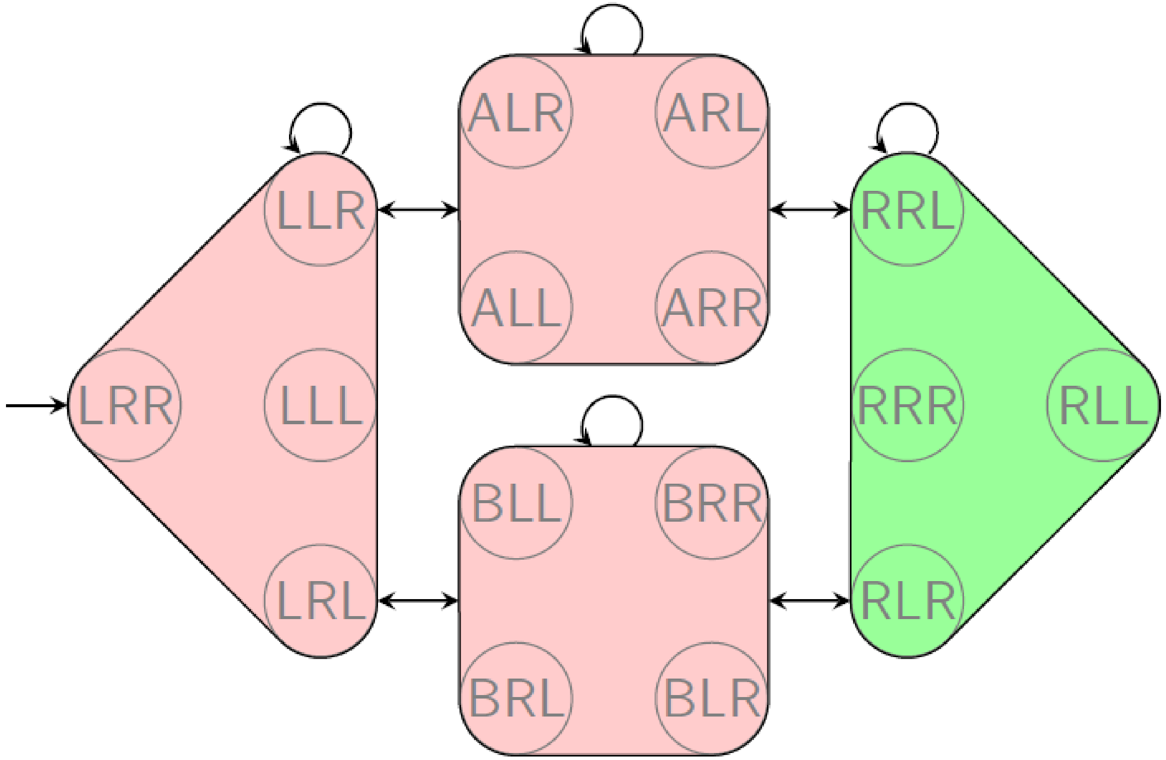


Logistics problem with one package, two trucks, two locations:

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

Example: projection

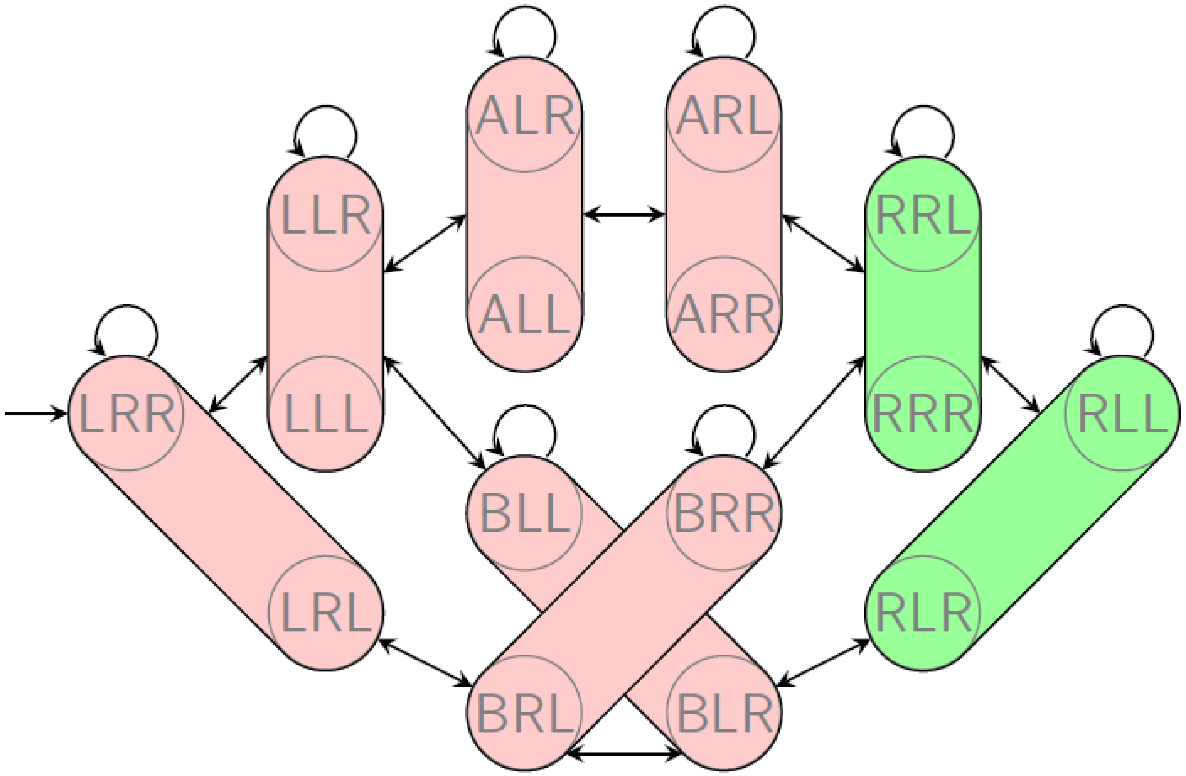
Project to {package}:



AI CENTER FEE CTU

Example: projection

Project to {package, truck A}:



Limits 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 & Shrink heuristic: general 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 a succinct abstraction mapping

- 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 abstraction: 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.

Merge-and-shrink in pseudo-code

Generic Merge & Shrink Algorithm for planning task Π

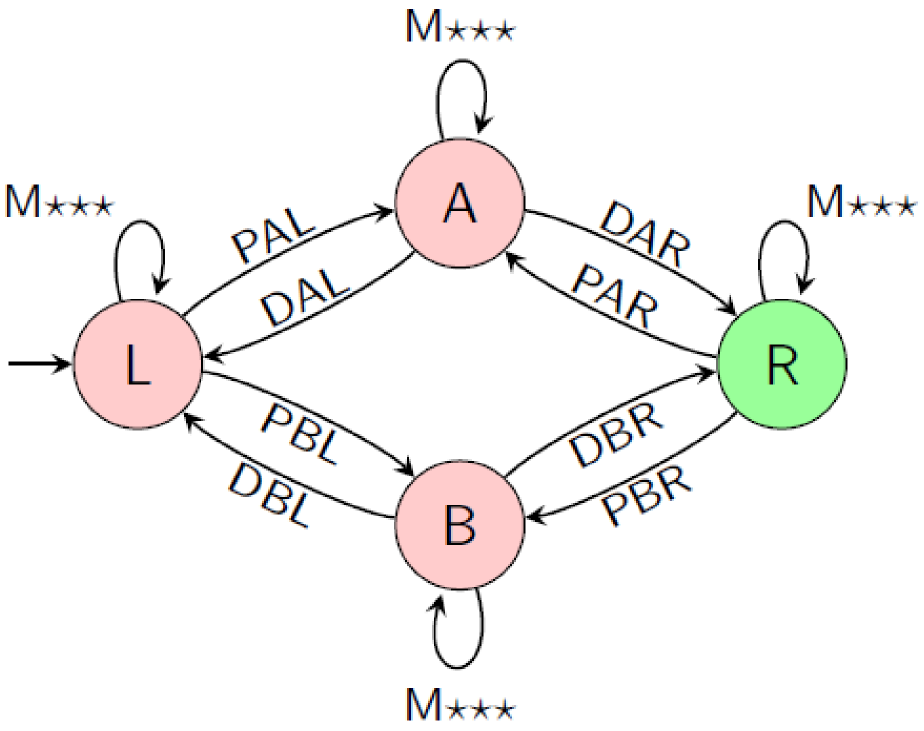
$F := F(\Pi)$
while $|F| > 1$:
  select $type \in \{\text{merge}, \text{shrink}\}$
  if $type = \text{merge}$:
    select $\mathcal{T}_1, \mathcal{T}_2 \in F$
    $F := (F \setminus \{\mathcal{T}_1, \mathcal{T}_2\}) \cup \{\mathcal{T}_1 \otimes \mathcal{T}_2\}$
  if $type = \text{shrink}$:
    select $\mathcal{T} \in F$
    choose an abstraction mapping β on $\mathcal{T}$
    $F := (F \setminus \{\mathcal{T}\}) \cup \{\mathcal{T}^{\beta}\}$
return the remaining factor $\mathcal{T}^{\alpha}$ in F

Running example: explanation

- 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

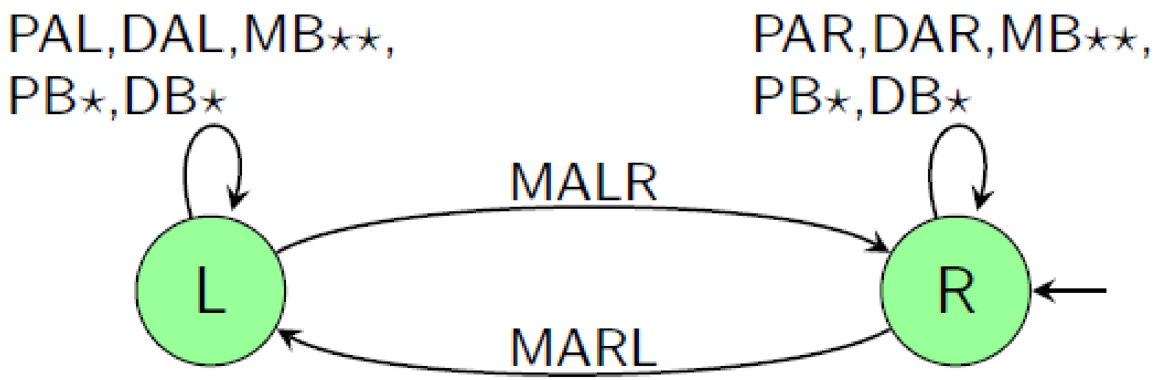
$\mathcal{T}^{\pi_{\{\text{package}\}}}$:



AI CENTER FEE CTU

Running example: atomic projection for truck A

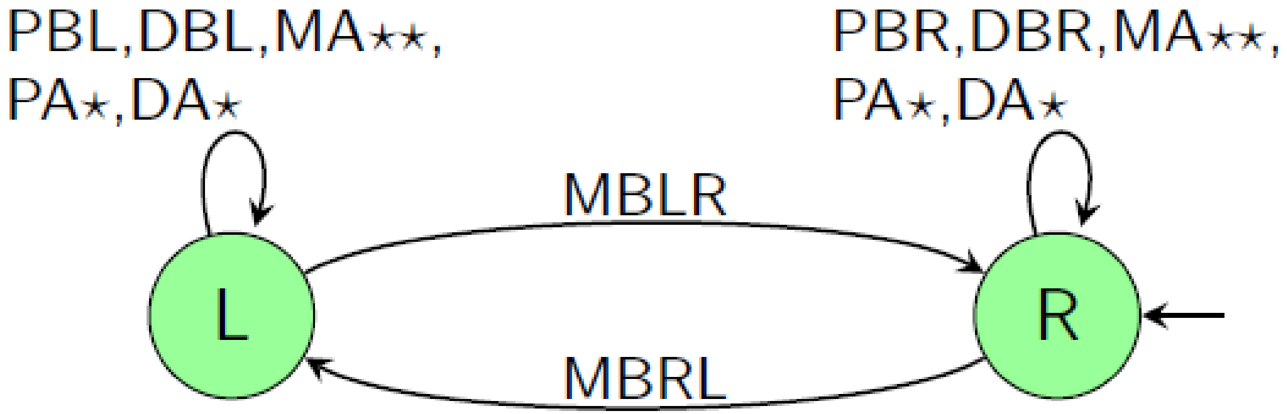
$\mathcal{T}^{\pi_{\{\text{truck A}\}}}$:



AI CENTER
FEE CTU

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

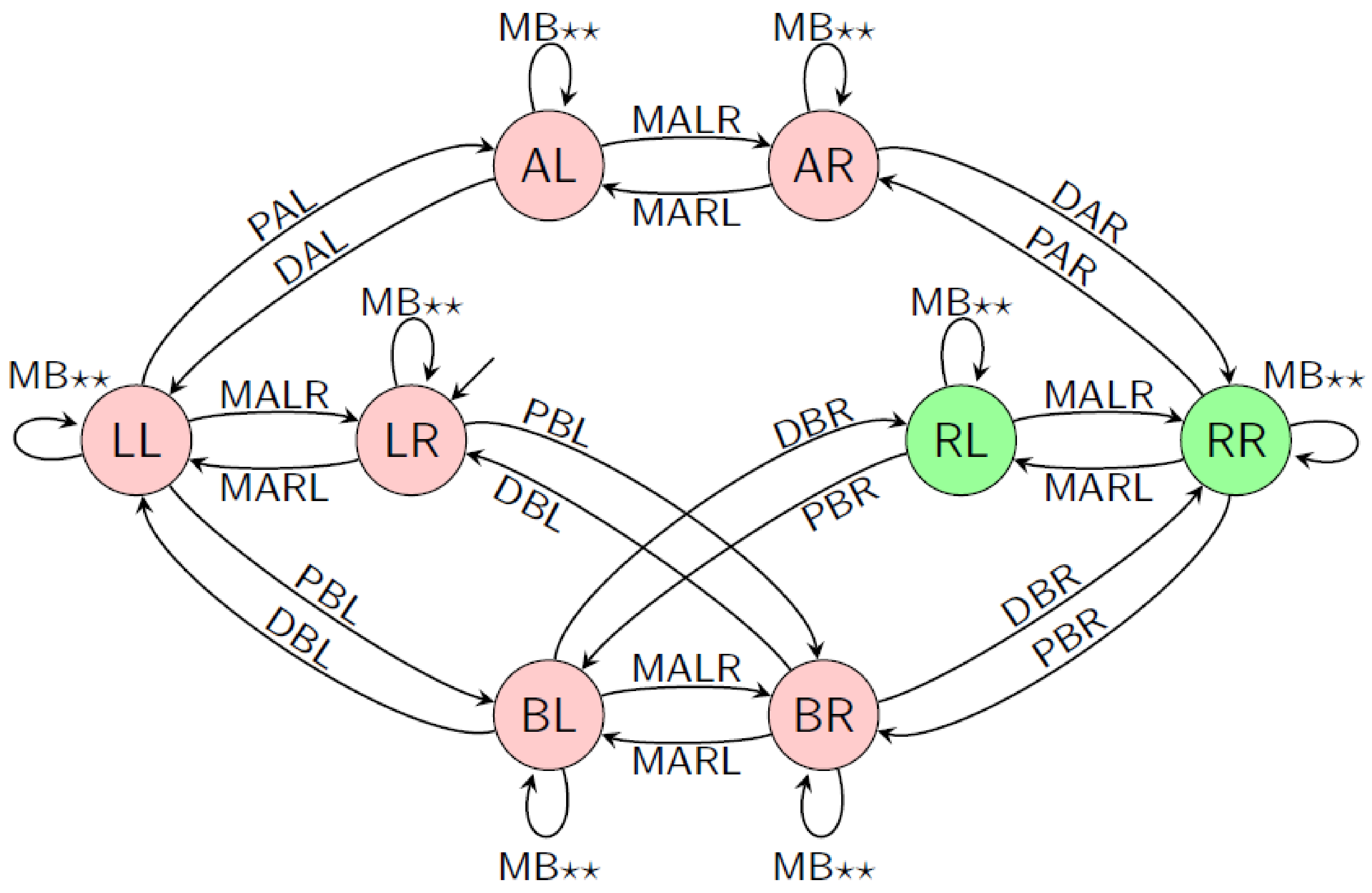
Definition (synchronized product of functions)

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 $\alpha_\otimes(s) = \langle \alpha_1(s), \alpha_2(s) \rangle$.

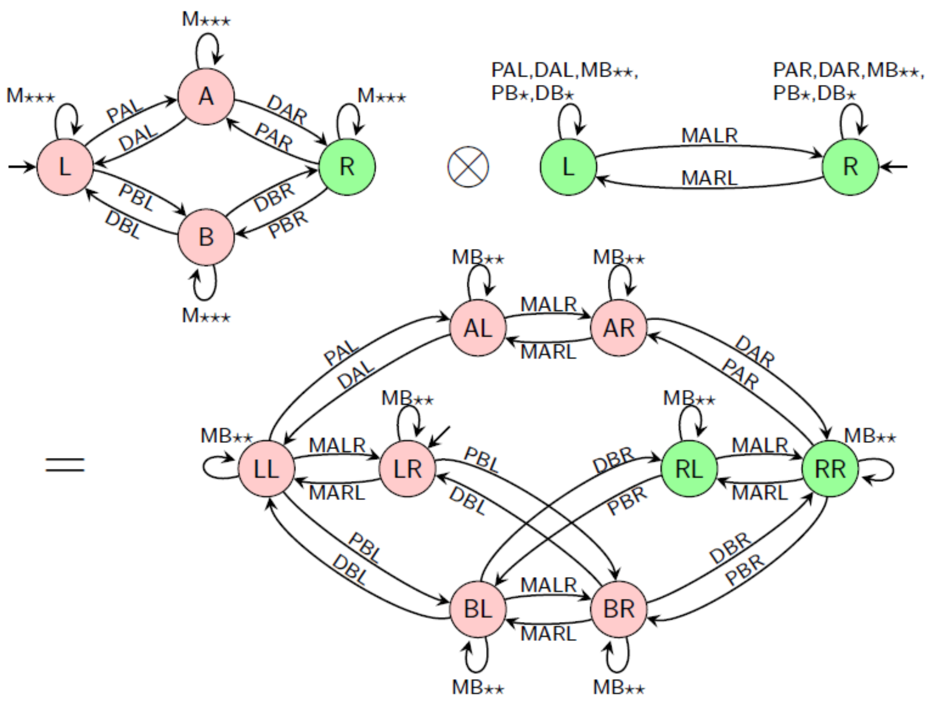
Synchronized product: example

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



Example: computation of synchronized product

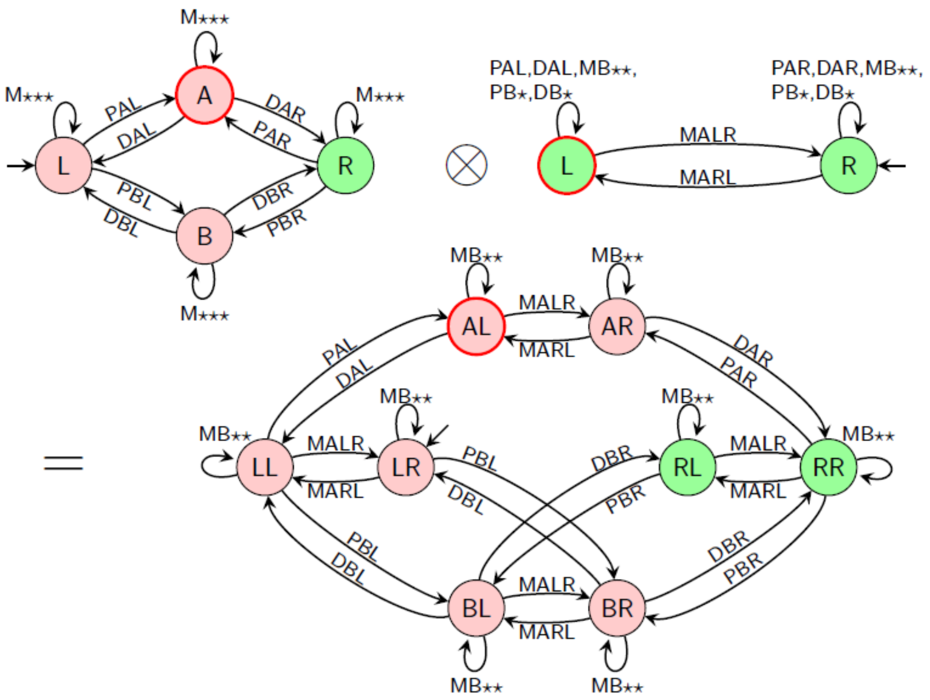
$\mathcal{T}^{\pi}\{\text{package}\} \otimes \mathcal{T}^{\pi}\{\text{truck A}\}$:



AI CENTER
FEE CTU

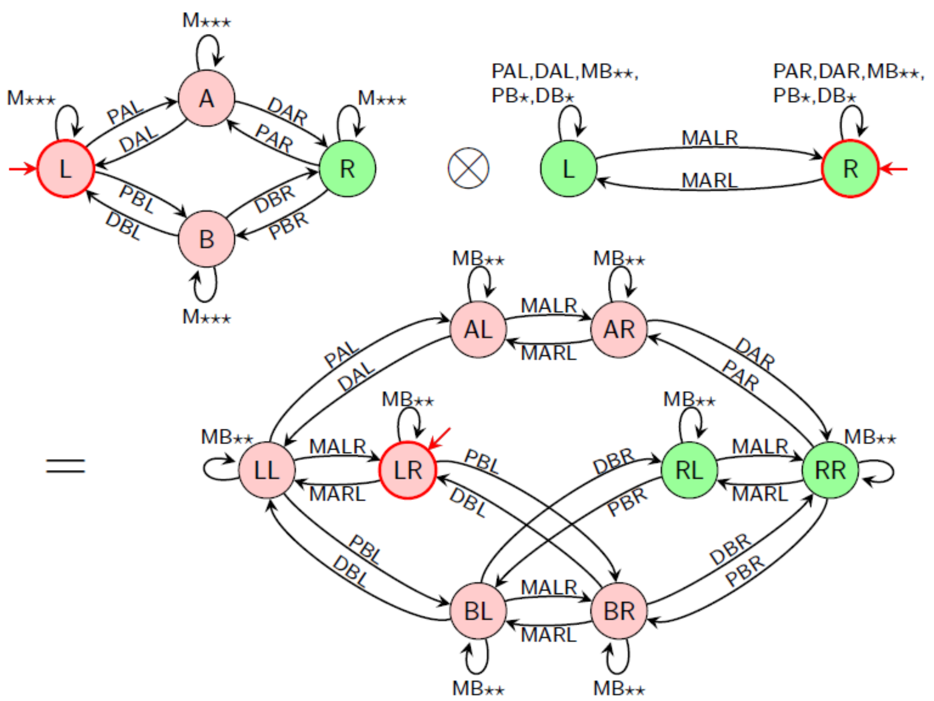
Example: computation of synchronized product

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



Example: computation of synchronized product

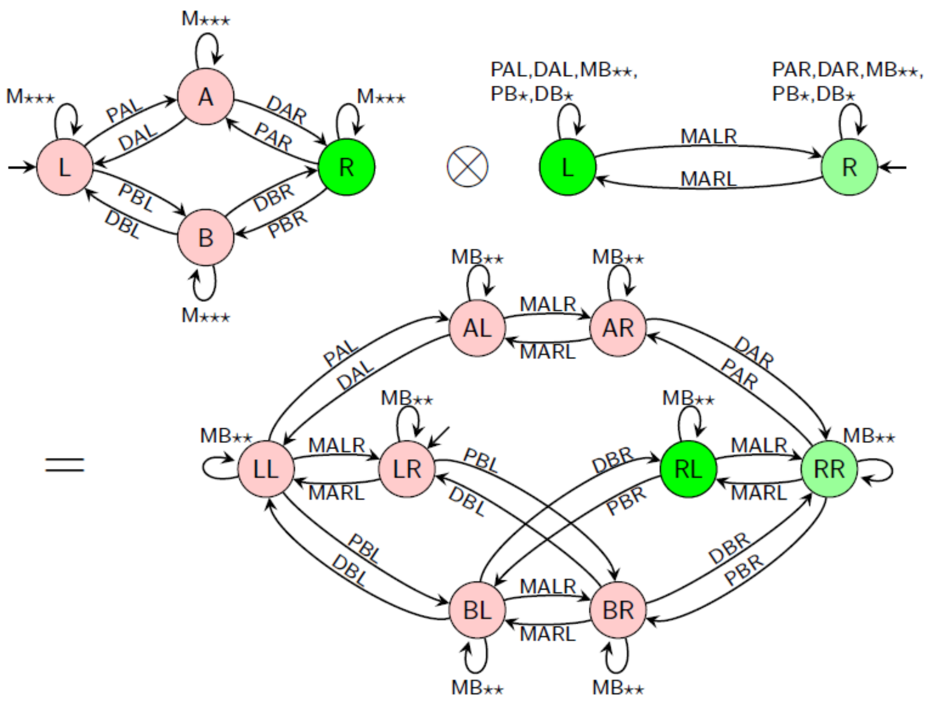
$\mathcal{T}^{\pi\{\text{package}\}} \otimes \mathcal{T}^{\pi\{\text{truck A}\}}: I_{\otimes} = I_1 \times I_2$



AI CENTER
FEE CTU

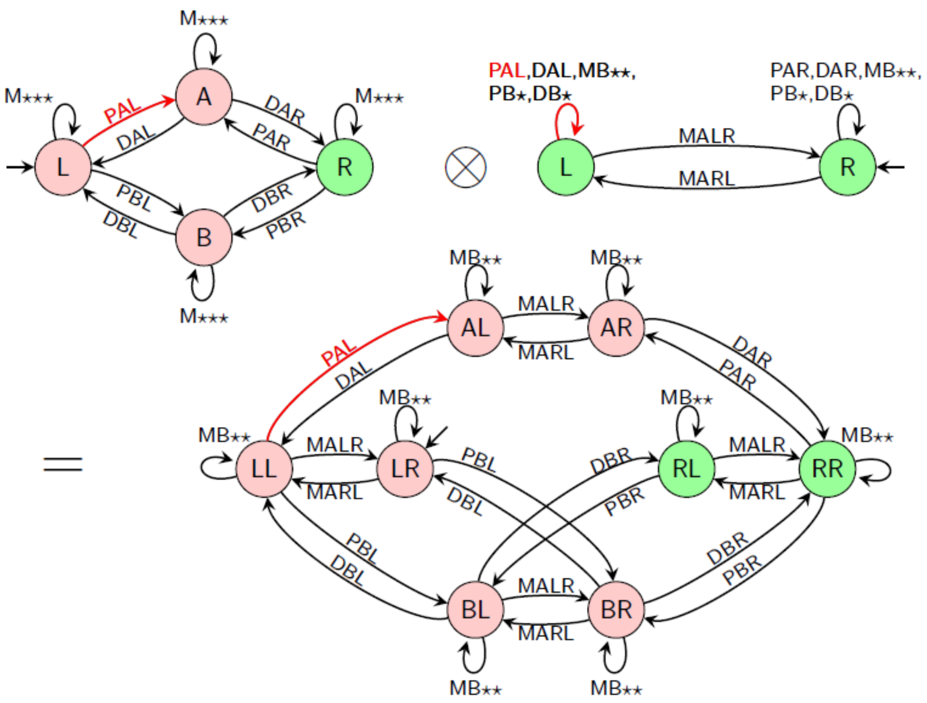
Example: computation of synchronized product

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



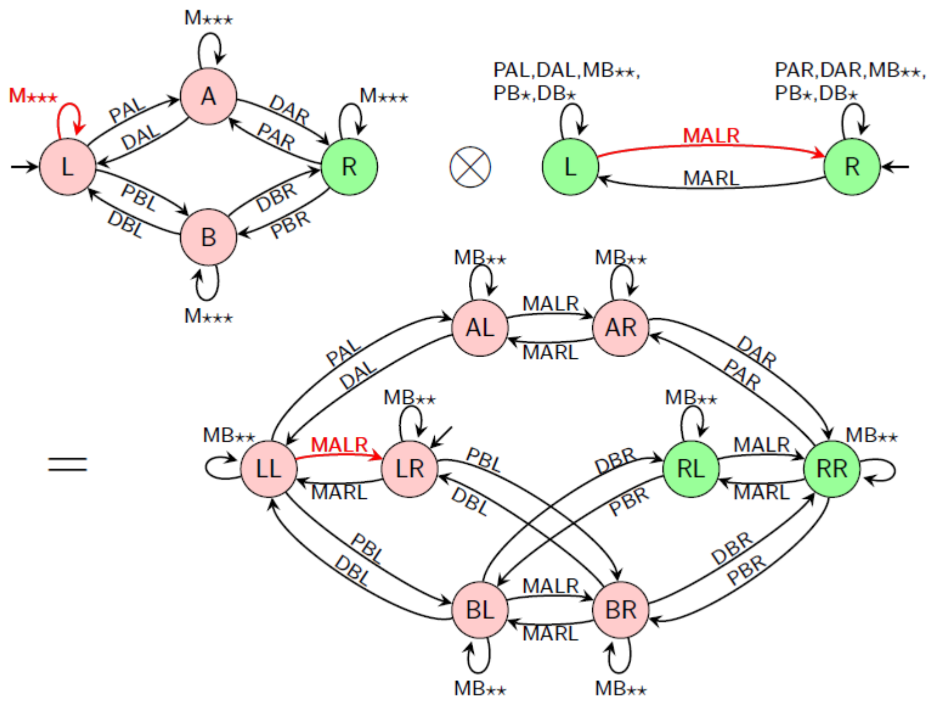
Example: computation of synchronized product

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



Example: computation of synchronized product

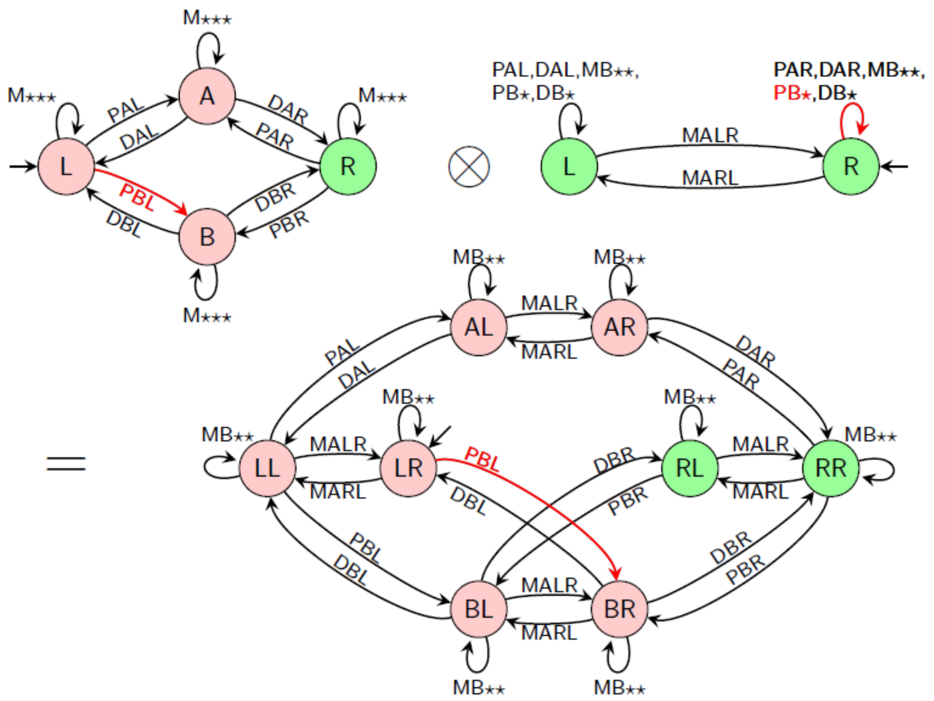
$\mathcal{T}^{\pi\{\text{package}\}} \otimes \mathcal{T}^{\pi\{\text{truck A}\}}$: $T_\otimes := \{\langle\langle s_1, s_2\rangle, l, \langle t_1, t_2\rangle\rangle \mid \dots\}$



AI CENTER
FEE CTU

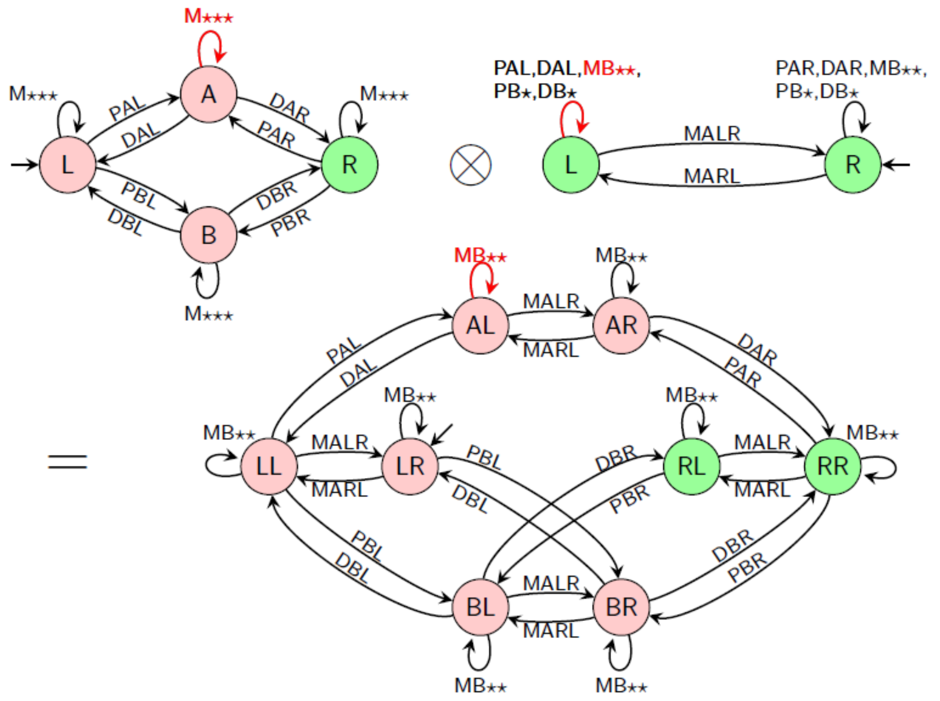
Example: computation of synchronized product

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



Example: computation of synchronized product

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



AI CENTER
FEE CTU

Generic merge-and-shrink procedure

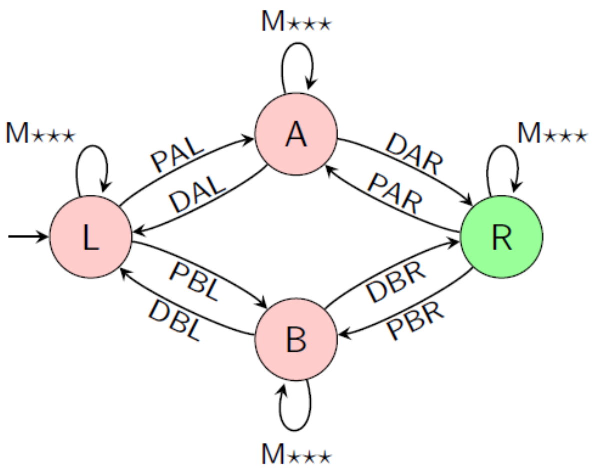
Using the results from the previous section, we can develop the ideas of a generic abstraction computation procedure that takes all state variables into account:

- 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.

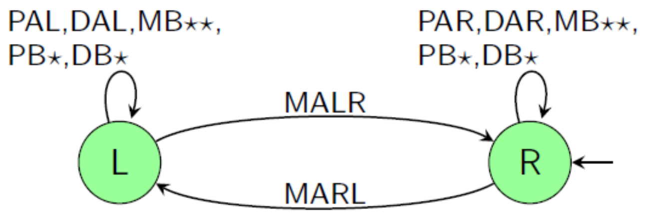
Initialization step: projection for package

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



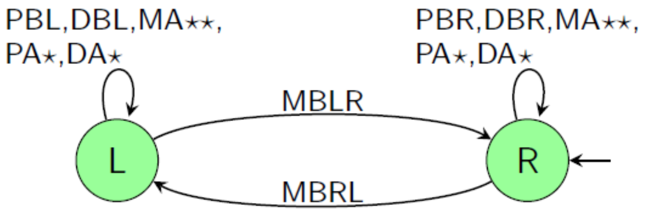
Initialization step: projection for truck A

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



Initialization step: projection for truck B

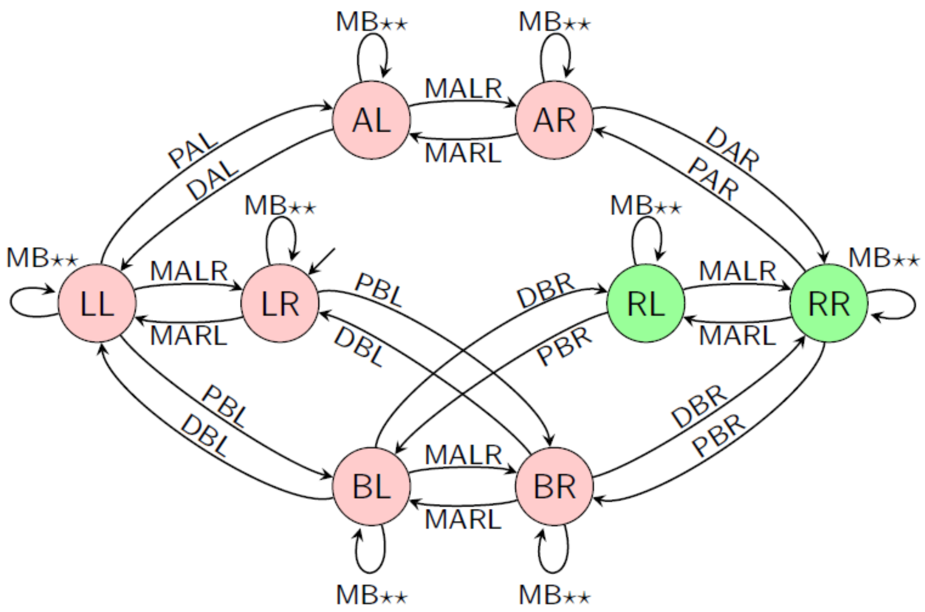
$\mathcal{T}^{\pi\{\text{truck B}\}}$:



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: $\{\mathcal{T}_1, \mathcal{T}^{\pi\{\text{truck B}\}}\}$

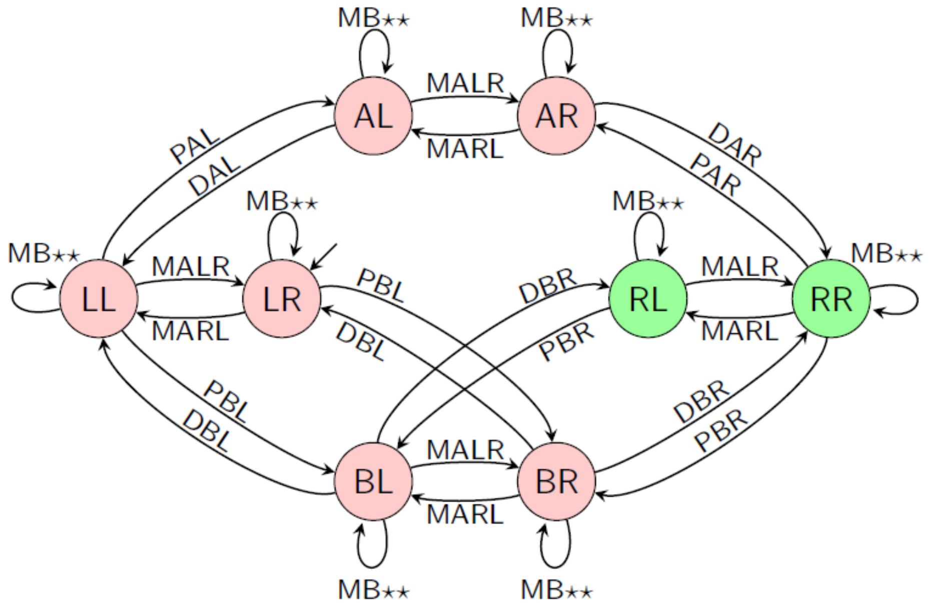
AI CENTER FEE CTU

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.

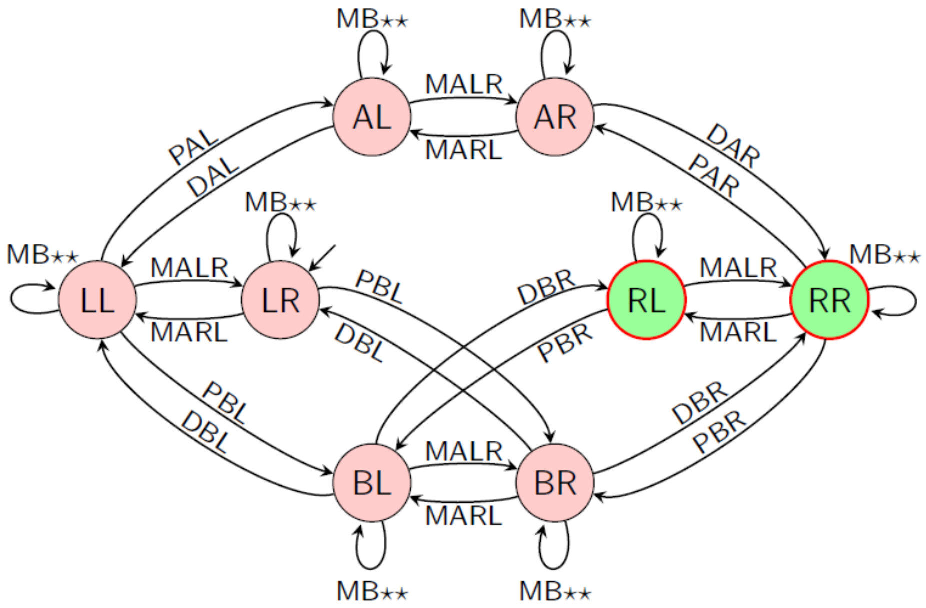
First shrink step

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



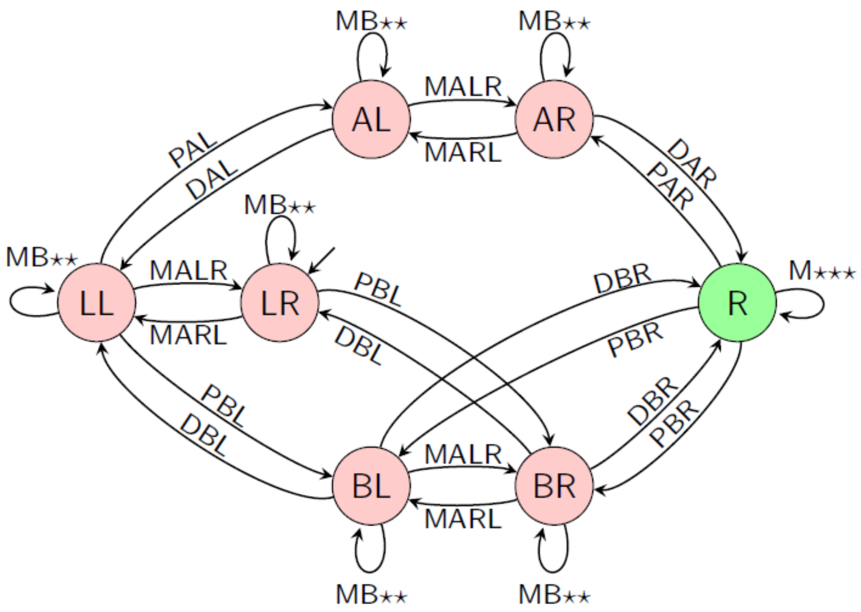
First shrink step

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



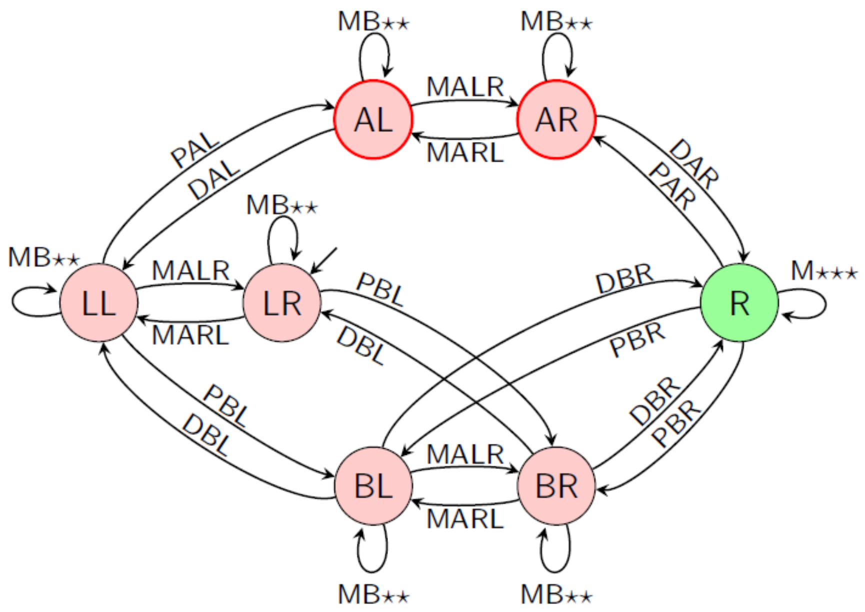
First shrink step

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



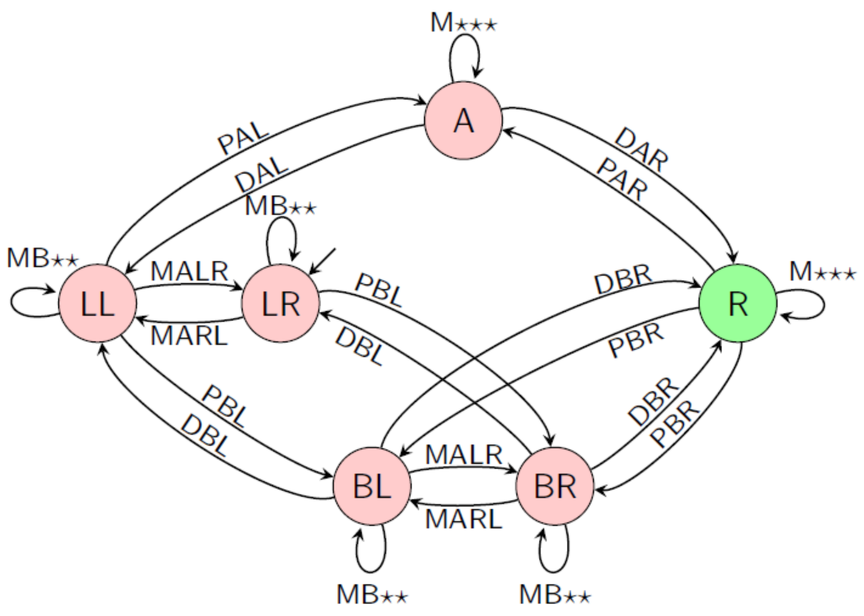
First shrink step

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



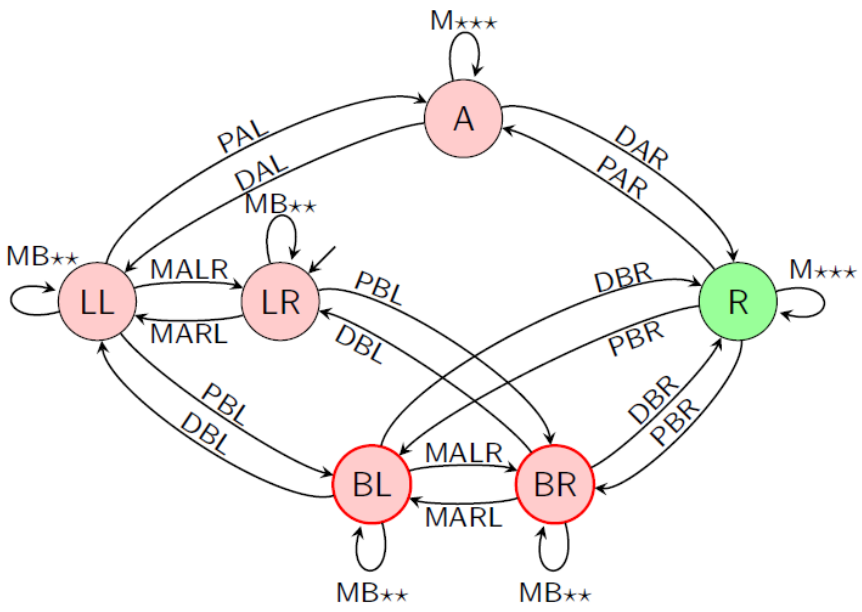
First shrink step

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



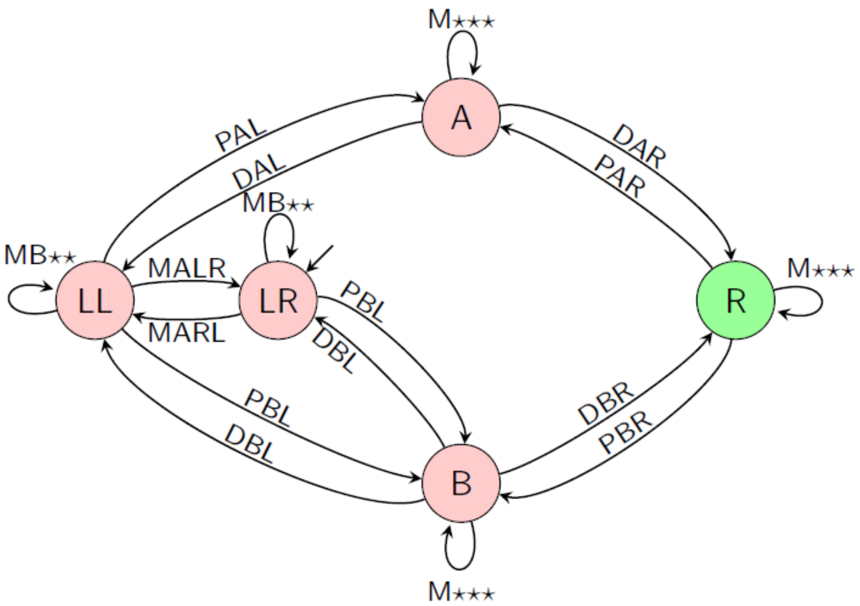
First shrink step

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



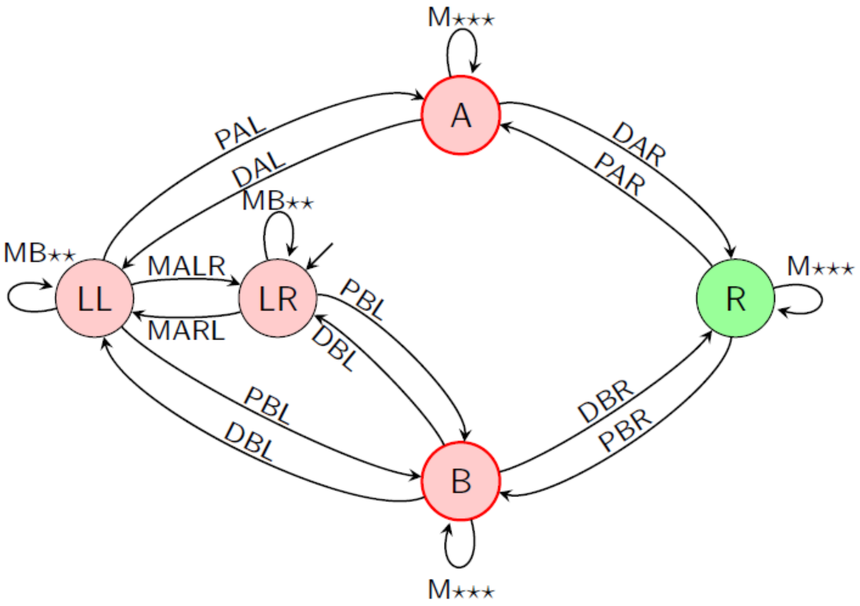
First shrink step

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



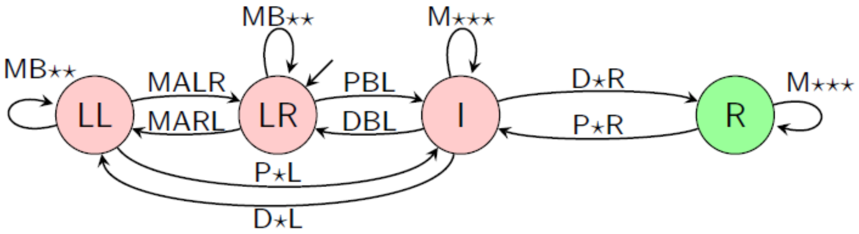
First shrink step

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



First shrink step

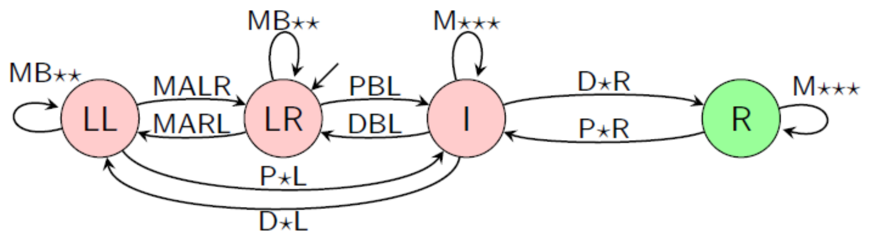
$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



AI CENTER
FEE CTU

First shrink step

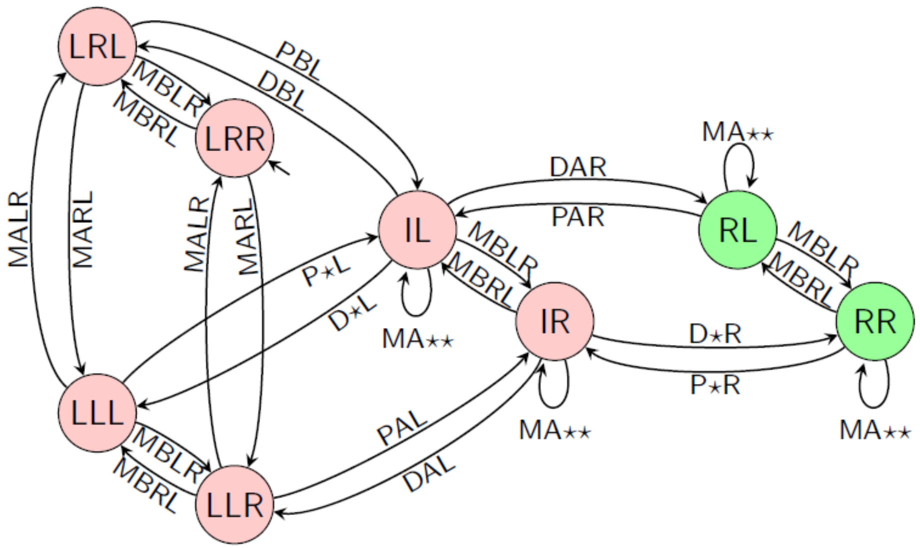
$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



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

Second shrink step

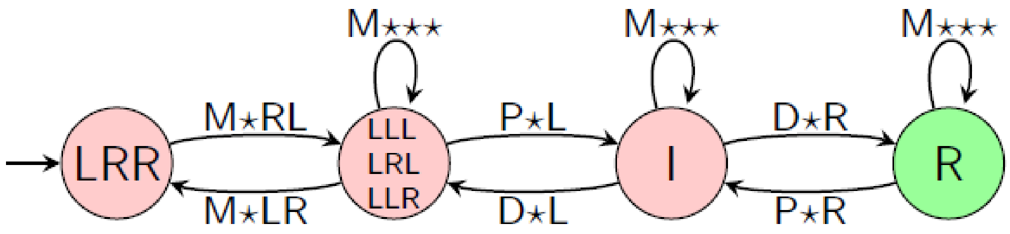
$\mathcal{T}_3 := \mathcal{T}_2 \otimes \mathcal{T}^{\pi\{\text{truck B}\}}$:



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).

Properties of Merge-and-Shrink heuristic

To understand merge-and-shrink abstractions better,
we are interested in the properties of the resulting heuristic:

- Is it admissible ($h^\alpha(s) \leq h^*(s)$ for all states s)?
- Is it consistent ($h^\alpha(s) \leq c(o) + h^\alpha(t)$ for all trans. $s \xrightarrow{o} t$)?
- Is it perfect ($h^\alpha(s) = h^*(s)$ for all states s)?

Because merge-and-shrink is a generic procedure,
the answers may depend on how exactly we instantiate it:

- size limits
- merge strategy
- shrink strategy

Merge-and-Shrink as a sequence of transformations

- Consider a run of the merge-and-shrink construction algorithm with n iterations of the main loop.
- Let F_i $(0 \leq i \leq n)$ be the FTS F after i loop iterations.
- Let $\mathcal{T}_i$ $(0 \leq i \leq n)$ be the transition system represented by F_i, i.e., $\mathcal{T}_i = \bigotimes F_i$.
- In particular, $F_0 = F(\Pi)$ and $F_n = \{\mathcal{T}_n\}$.
- For SAS$^+$ tasks Π, we also know $\mathcal{T}_0 = \mathcal{T}(\Pi)$.

For a formal study, it is useful to view merge-and-shrink construction as a sequence of transformations from $\mathcal{T}_i$ to $\mathcal{T}_{i+1}$.

Transformation

Definition (Transformation)

Let $\mathcal{T} = \langle S, L, c, T, s_0, S_\star \rangle$ and $\mathcal{T}' = \langle S', L, c, T', s_0', S_\star' \rangle$ be transition systems with the same labels and costs.
Let $\sigma : S \to S'$ map the states of $\mathcal{T}$ to the states of $\mathcal{T}'$.

The triple $\tau = \langle \mathcal{T}, \sigma, \mathcal{T}' \rangle$ is called a transformation from $\mathcal{T}$ to $\mathcal{T}'$.
We also write it as $\mathcal{T} \xrightarrow{\sigma} \mathcal{T}'$.

The transformation τ induces the heuristic h^τ for $\mathcal{T}$ defined as $h^\tau(s) = h^*_{\mathcal{T}'}(\sigma(s))$.

Example: If α is an abstraction mapping for transition system $\mathcal{T}$, then $\mathcal{T} \xrightarrow{\alpha} \mathcal{T}^\alpha$ is a transformation.

Special Transformation

- A transformation $\tau = \mathcal{T} \xrightarrow{\sigma} \mathcal{T}'$ is called conservative if it corresponds to an abstraction, i.e., if $\tau = \mathcal{T} \xrightarrow{\alpha} \mathcal{T}^{\alpha}$ for some abstraction mapping α.
- A transformation $\tau = \mathcal{T} \xrightarrow{\sigma} \mathcal{T}'$ is called exact if it induces the perfect heuristic, i.e., if $h^{\tau}(s) = h^{*}(s)$ for all states s of $\mathcal{T}$.

Merge transformations are always conservative and exact.

Shrink transformations are always conservative.

Composition of transformation

Merge-and-shrink performs many transformations in sequence.
We can formalize this with a notion of composition:

- Given $\tau = \mathcal{T} \xrightarrow{\sigma} \mathcal{T}'$ and $\tau' = \mathcal{T}' \xrightarrow{\sigma'} \mathcal{T}''$, their composition $\tau'' = \tau' \circ \tau$ is defined as $\tau'' = \mathcal{T} \xrightarrow{\sigma' \circ \sigma} \mathcal{T}''$.
- If τ and τ' are conservative, then $\tau' \circ \tau$ is conservative.
- If τ and τ' are exact, then $\tau' \circ \tau$ is exact.

Conclusion: Merge-and-Shrink heursitic

We can conclude the following properties of merge-and-shrink heuristics for SAS^+ tasks:

- The heuristic is always admissible and consistent (because it is induced by a a composition of conservative transformations and therefore an abstraction).
- If all shrink transformation used are exact, the heuristic is perfect (because it is induced by a composition of exact transformations).

Further topics

Further topics in merge-and-shrink abstraction:

- how to keep track of the abstraction mapping
- efficient implementation
- concrete merge strategies
  - often focus on goal variables and causal connectivity (similar to hill-climbing for pattern selection)
  - sometimes based on mutexes or symmetries
- concrete shrink strategies
  - especially: h-preserving, f-preserving, bisimulation-based
  - (some) bisimulation-based shrinking strategies are exact
- other transformations besides merging and shrinking
  - especially: pruning and label reduction