# Planning and Optimization

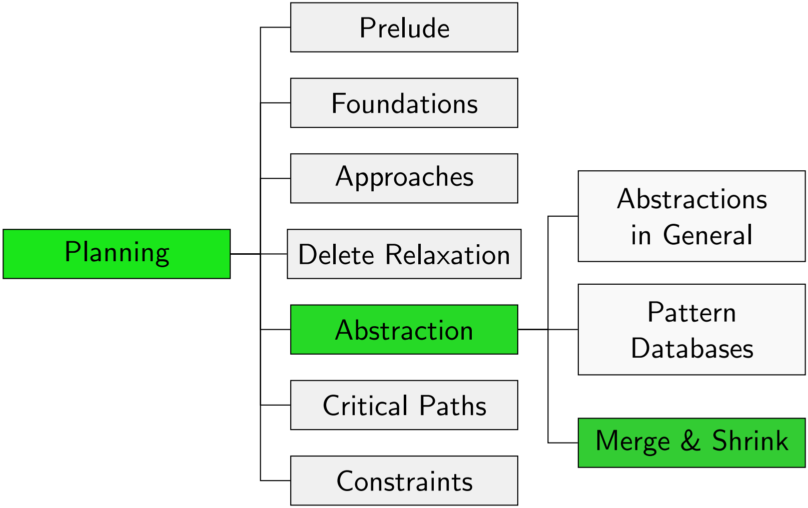
## E10. Merge-and-Shrink: Algorithm

Malte Helmert and Gabriele Röger

Universität Basel

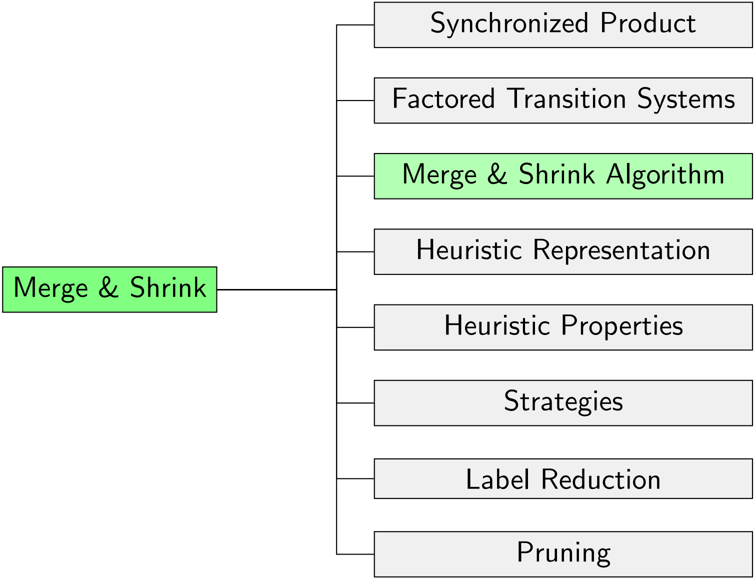
November 15, 2023

# Content of this Course



# Generic Algorithm

# Merge-and-Shrink



# Generic Merge-and-shrink Abstractions: Outline

Using the results of the previous chapter, we can develop a generic abstraction computation procedure that takes all state variables into account.

- Initialization: Compute the FTS consisting of all atomic projections.
- Loop: Repeatedly apply a transformation to the FTS.
  - Merging: Combine two factors by replacing them with their synchronized product.
  - Shrinking: If the factors are too large, make one of them smaller by abstracting it further (applying an arbitrary abstraction to it).
- Termination: Stop when only one factor is left.

The final factor is then used for an abstraction heuristic.

# Generic Algorithm Template

**Generic Merge & Shrink Algorithm for planning task Π**

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

In Ch. E12 and E13, we will include more transformation types (label reduction and pruning)

# Merge-and-Shrink Strategies

Choices to resolve to instantiate the template:

- When to merge, when to shrink?
  $\rightsquigarrow$ general strategy
- Which abstractions to merge?
  $\rightsquigarrow$ merge strategy
- Which abstraction to shrink, and how to shrink it (which $\beta$)?
  $\rightsquigarrow$ shrink strategy

# Merge-and-Shrink Strategies

Choices to resolve to instantiate the template:

- When to merge, when to shrink?
  $\rightsquigarrow$ general strategy
- Which abstractions to merge?
  $\rightsquigarrow$ merge strategy
- Which abstraction to shrink, and how to shrink it (which $\beta$)?
  $\rightsquigarrow$ shrink strategy

merge and shrink strategies $\rightsquigarrow$ Ch. E11/E12

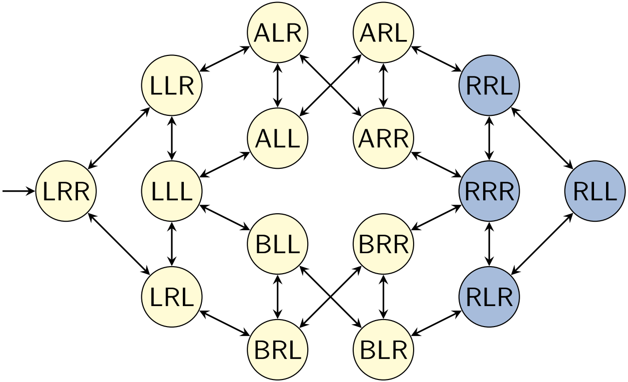
# General Strategy

A typical general strategy:

- define a limit $N$ on the number of states allowed in each factor
- in each iteration, select two factors we would like to merge
- merge them if this does not exhaust the state number limit
- otherwise shrink one or both factors just enough
  to make a subsequent merge possible

# Example

# Back to the Running Example

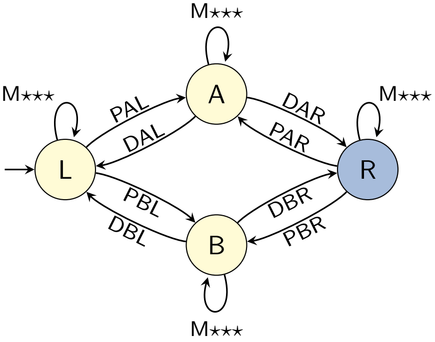



Logistics problem with one package, two trucks, two locations:

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

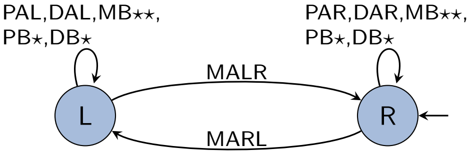
# Initialization Step: Atomic Projection for Package

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



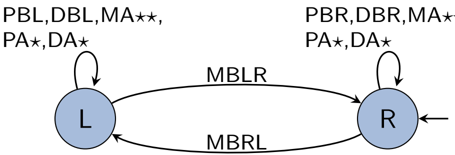
# Initialization Step: Atomic Projection for Truck A

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



# Initialization Step: Atomic Projection for Truck B

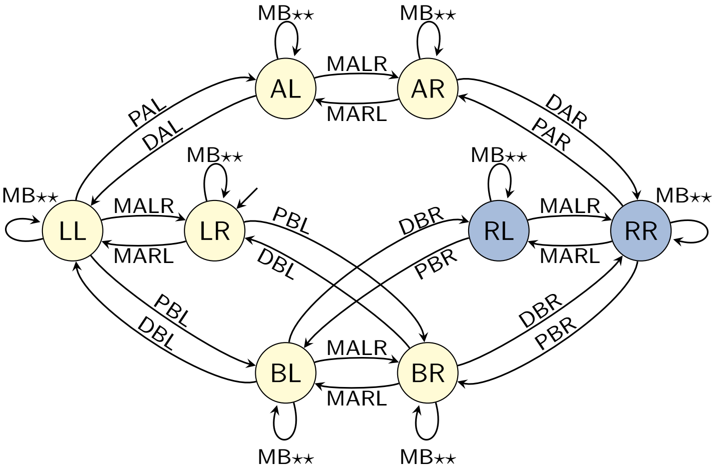
$\mathcal{T}^{\pi_{\{\text{truck B}\}}}$:




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

# First Merge Step

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



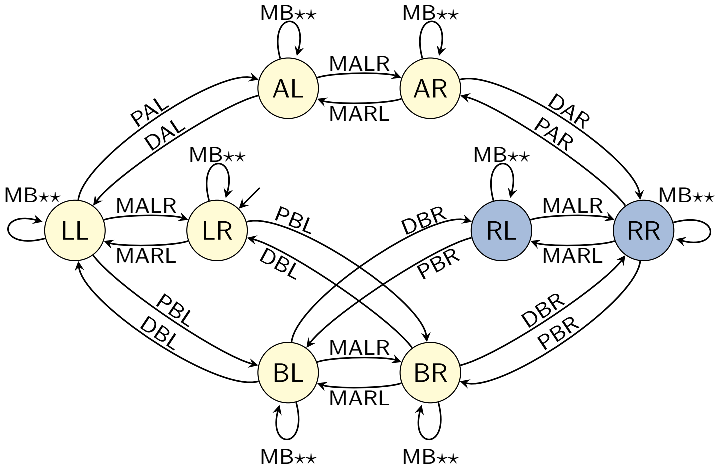

current FTS: $\{\mathcal{T}_1, \mathcal{T}^{\pi\{\text{truck B}\}}\}$

# Need to Shrink?

- With sufficient memory, we could now compute $\mathcal{T}_1 \otimes \mathcal{T}^{\pi_{\{\text{truck B}\}}}$ and recover the full transition system of the task.
- However, to illustrate the general idea,
  we assume that memory is too restricted:
  we may never create a factor with more than 8 states.
- To make the product fit the bound, we shrink $\mathcal{T}_1$ to 4 states.
  We can decide freely how exactly to abstract $\mathcal{T}_1$.
- In this example, we manually choose an abstraction that leads to a good result in the end. Making good shrinking decisions algorithmically is the job of the shrink strategy.

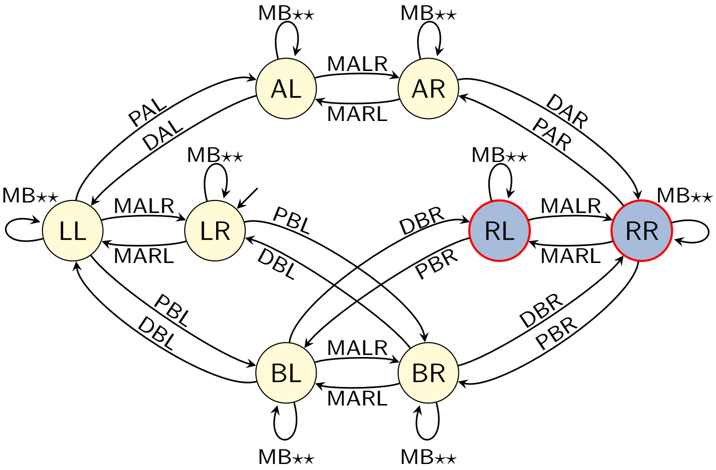
# First Shrink Step

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



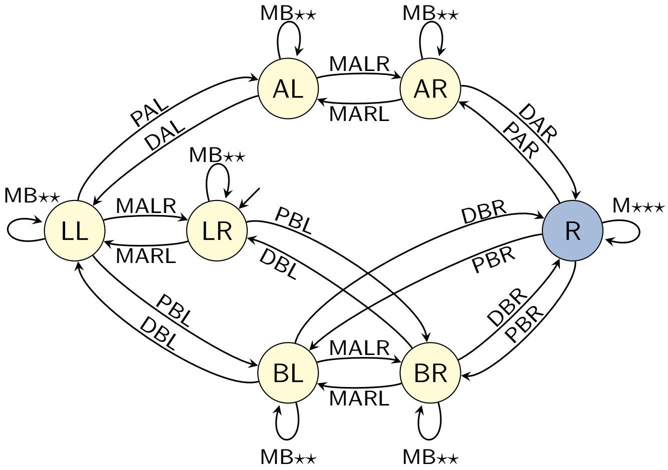
# First Shrink Step

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



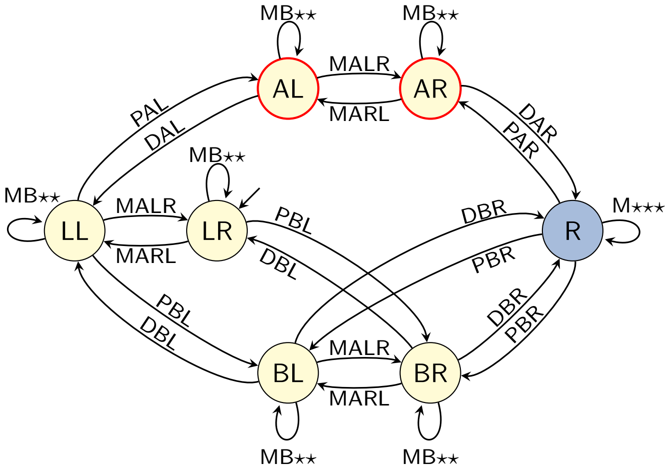
# First Shrink Step

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



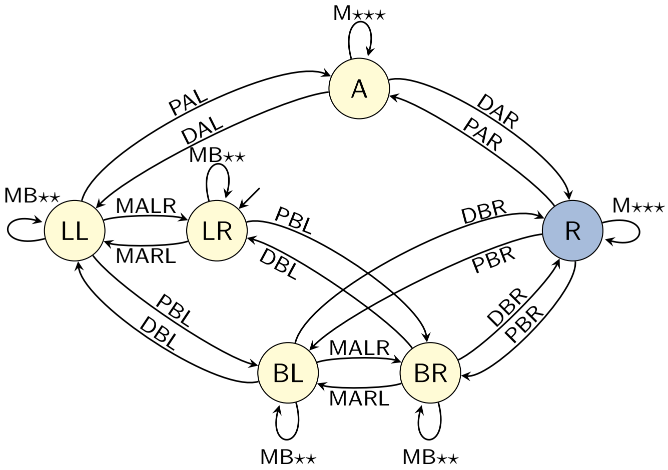
# First Shrink Step

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



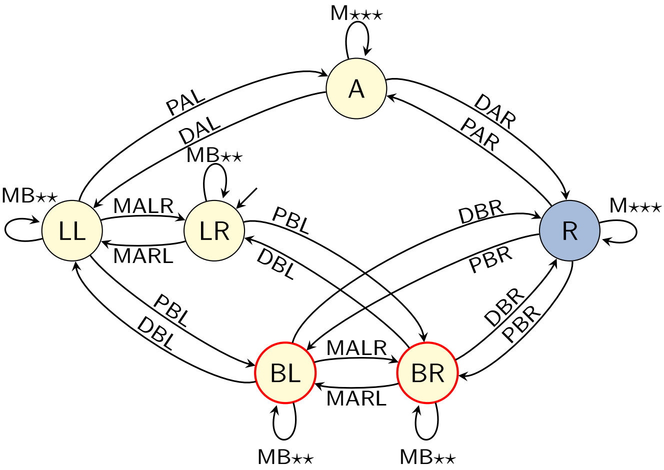
# First Shrink Step

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



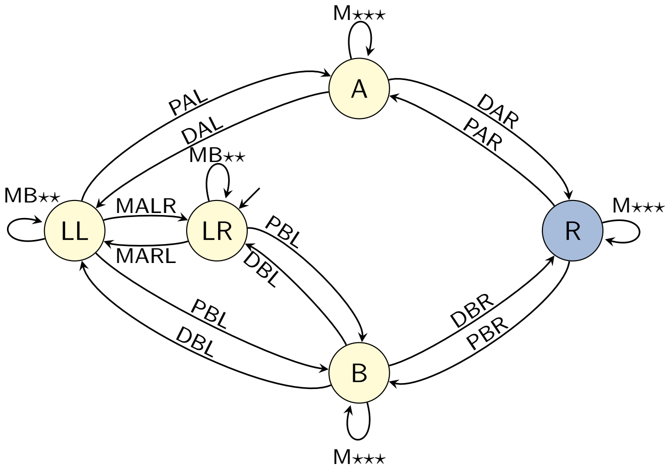
# First Shrink Step

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



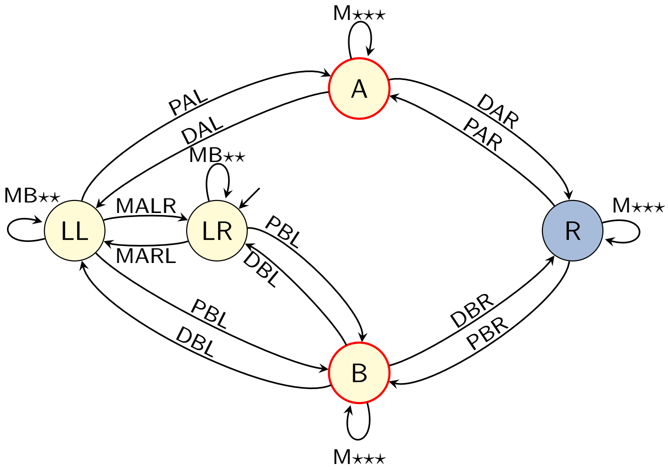
# First Shrink Step

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



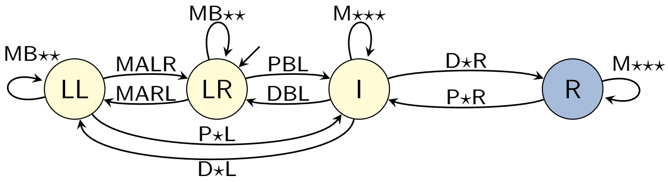
# First Shrink Step

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



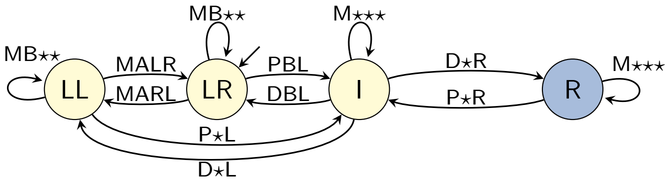
# First Shrink Step

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



# First Shrink Step

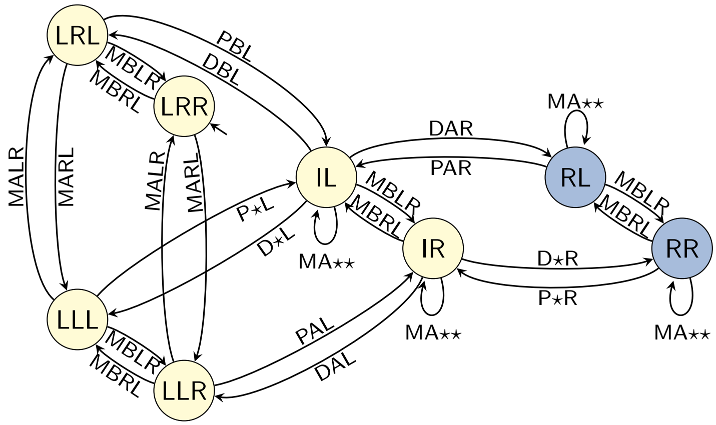
$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$




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

# Second Merge Step

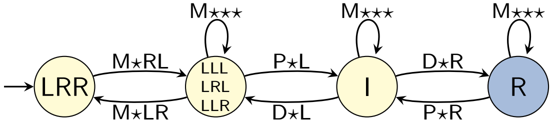
$\mathcal{T}_3 := \mathcal{T}_2 \otimes \mathcal{T}^{\pi_{\{\text{truck B}\}}}$:




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

# Another Shrink Step?

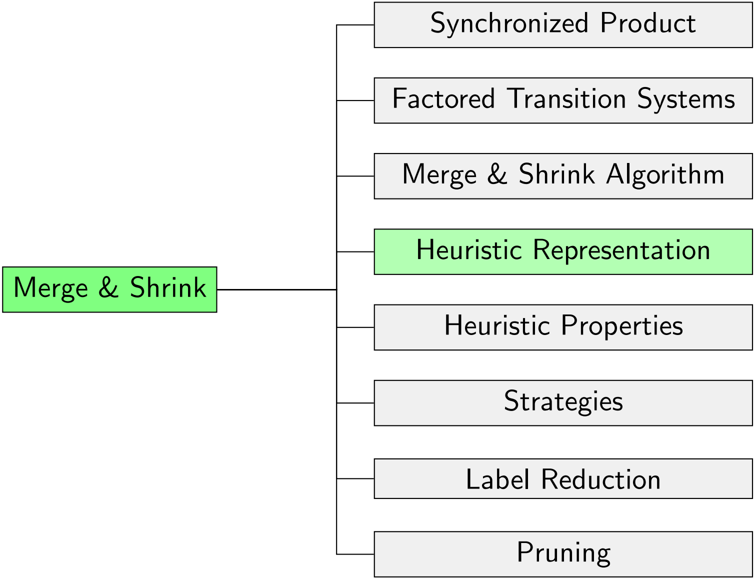
- At this point, merge-and-shrink construction stops.
  The distances in the final factor define the heuristic function.
- If there were further state variables to integrate,
  we would shrink again, e.g., leading to the following
  abstraction (again with four states):




- 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 arbitrarily many trucks,
  even if we stick to the fixed size limit of 8.

# Maintaining the Abstraction

# Merge-and-Shrink



# Generic Algorithm Template

**Generic Merge & Shrink Algorithm for planning task $\Pi$**

$F := F(\Pi)$
**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 $\beta$ on $\mathcal{T}$
        $F := (F \setminus \{\mathcal{T}\}) \cup \{\mathcal{T}^\beta\}$
**return** the remaining factor $\mathcal{T}^\alpha$ in $F$

- The algorithm computes an abstract transition system.
- For the heuristic evaluation, we need an abstraction.
- How to maintain and represent the corresponding abstraction?

# The Need for Succinct Abstractions

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

# How to Represent the Abstraction? (1)

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

- For the atomic abstractions $\pi_{\{v\}}$, we generate a one-dimensional table that denotes which value in $\mathrm{dom}(v)$ corresponds to which abstract state in $\mathcal{T}^{\pi_{\{v\}}}$.
- 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 to keep the table in sync with the abstraction choices.

# How to Represent the Abstraction? (2)

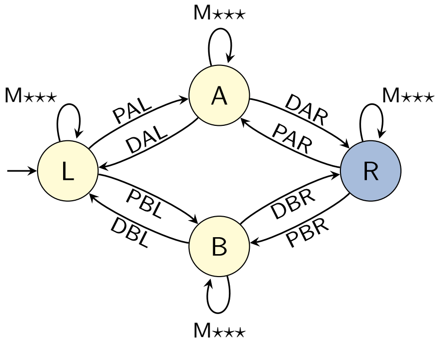
Idea: the computation of the abstraction mapping follows the sequence of product computations

- Once we have computed the final abstract transition system, we compute all abstract goal distances and store them in a one-dimensional table.
- At this point, we can throw away all the abstract transition systems – 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 abstract state and heuristic value
  $\rightsquigarrow 2|V|$ lookups, $O(|V|)$ time

Again, we illustrate the process with our running example.

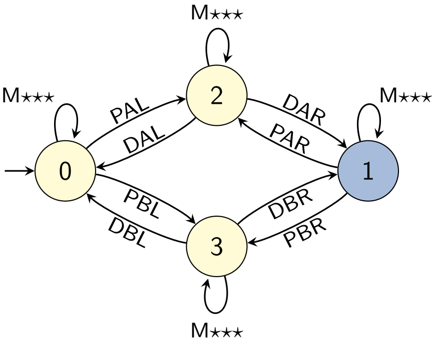
# Abstraction Example: Atomic Abstractions

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



# Abstraction Example: Atomic Abstractions

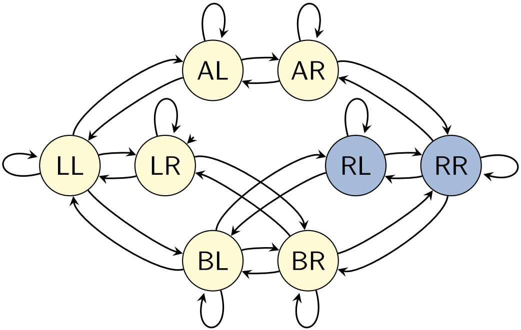
Computing abstractions for the transition systems of atomic abstractions is simple. Just number the states (domain values) consecutively and generate a table of references to the states:




| $L$ | $R$ | $A$ | $B$ |
|---|---|---|---|
| 0 | 1 | 2 | 3 |

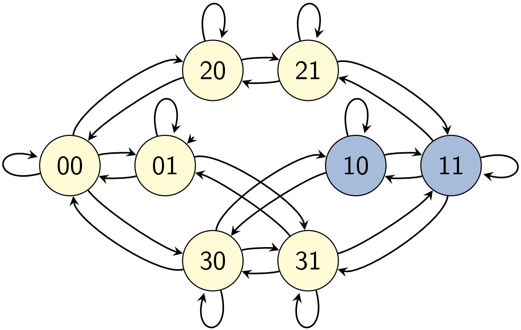
# Abstraction Example: Merge Step

For product transition systems $\mathcal{A}_1 \otimes \mathcal{A}_2$, 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 $\mathcal{A}$:



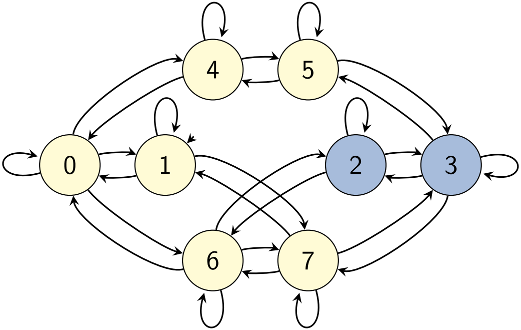
# Abstraction Example: Merge Step

For product transition systems $\mathcal{A}_1 \otimes \mathcal{A}_2$, 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 Example: Merge Step

For product transition systems $\mathcal{A}_1 \otimes \mathcal{A}_2$, 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 Abstraction when Shrinking

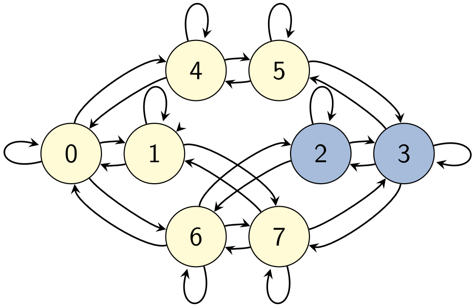
- The hard part in representing the abstraction is to keep it consistent when shrinking.
- In theory, this is easy to do:
  - When combining states $i$ and $j$, arbitrarily use one of them (say $i$) as the number of the new state.
  - Find all table entries in the table for this abstraction which map 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.

# Maintaining the Abstraction Efficiently

- Associate each abstract state with a linked list, representing all table entries that map to this state.
- Before starting the shrink operation, initialize the lists by scanning through the table, then discard the table.
- While shrinking, when combining $i$ and $j$, splice the list elements of $j$ into the list elements of $i$.
  - For linked lists, this is a constant-time operation.
- Once shrinking is completed, renumber all abstract states so that there are no gaps in the numbering.
- Finally, regenerate the mapping table from the linked list information.

# Abstraction Example: Shrink Step

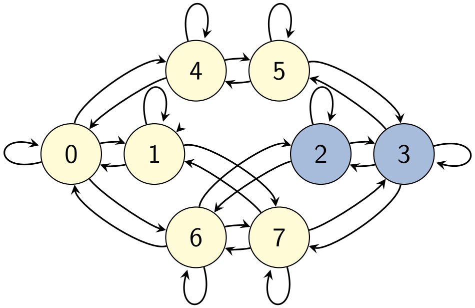
Representation before shrinking:




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

# Abstraction Example: Shrink Step

1. Convert table to linked lists and discard it.

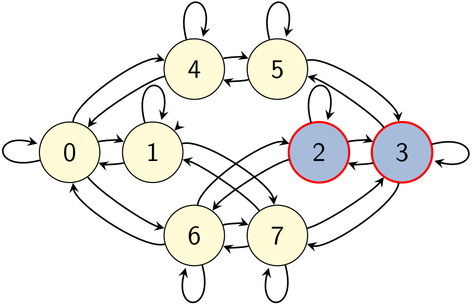



$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0)\}$
$list_3 = \{(1,1)\}$
$list_4 = \{(2,0)\}$
$list_5 = \{(2,1)\}$
$list_6 = \{(3,0)\}$
$list_7 = \{(3,1)\}$

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

# Abstraction Example: Shrink Step

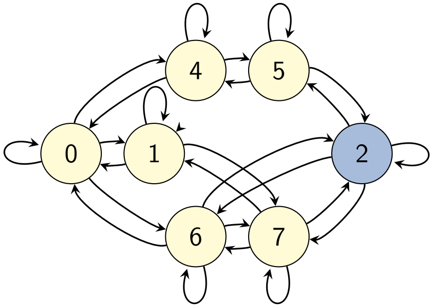
2. When combining $i$ and $j$, splice $list_j$ into $list_i$.




$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0)\}$
$list_3 = \{(1,1)\}$
$list_4 = \{(2,0)\}$
$list_5 = \{(2,1)\}$
$list_6 = \{(3,0)\}$
$list_7 = \{(3,1)\}$

# Abstraction Example: Shrink Step

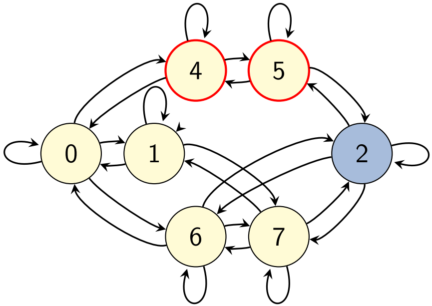
2. When combining $i$ and $j$, splice $list_j$ into $list_i$.




$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0),(1,1)\}$
$list_3 = \emptyset$
$list_4 = \{(2,0)\}$
$list_5 = \{(2,1)\}$
$list_6 = \{(3,0)\}$
$list_7 = \{(3,1)\}$

# Abstraction Example: Shrink Step

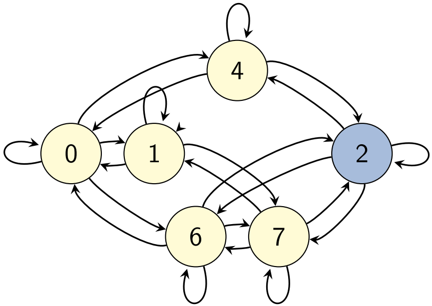
2. When combining $i$ and $j$, splice $list_j$ into $list_i$.




$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0),(1,1)\}$
$list_3 = \emptyset$
$list_4 = \{(2,0)\}$
$list_5 = \{(2,1)\}$
$list_6 = \{(3,0)\}$
$list_7 = \{(3,1)\}$

# Abstraction Example: Shrink Step

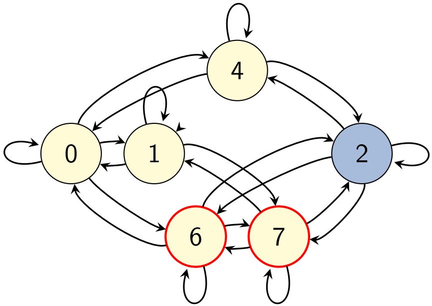
2. When combining $i$ and $j$, splice $list_j$ into $list_i$.




$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0),(1,1)\}$
$list_3 = \emptyset$
$list_4 = \{(2,0),(2,1)\}$
$list_5 = \emptyset$
$list_6 = \{(3,0)\}$
$list_7 = \{(3,1)\}$

# Abstraction Example: Shrink Step

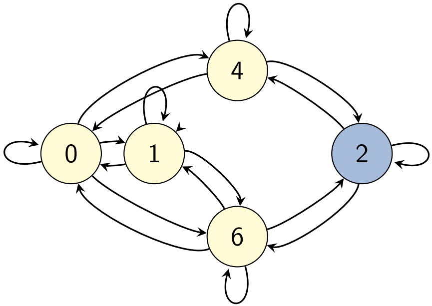
2. When combining $i$ and $j$, splice $list_j$ into $list_i$.




$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0),(1,1)\}$
$list_3 = \emptyset$
$list_4 = \{(2,0),(2,1)\}$
$list_5 = \emptyset$
$list_6 = \{(3,0)\}$
$list_7 = \{(3,1)\}$

# Abstraction Example: Shrink Step

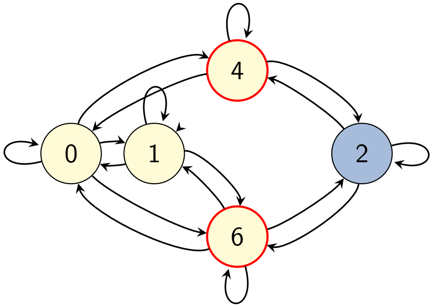
2. When combining $i$ and $j$, splice $list_j$ into $list_i$.




$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0),(1,1)\}$
$list_3 = \emptyset$
$list_4 = \{(2,0),(2,1)\}$
$list_5 = \emptyset$
$list_6 = \{(3,0),(3,1)\}$
$list_7 = \emptyset$

# Abstraction Example: Shrink Step

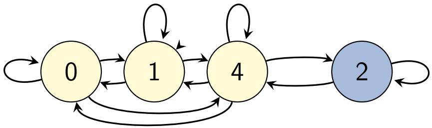
2. When combining $i$ and $j$, splice $list_j$ into $list_i$.




$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0),(1,1)\}$
$list_3 = \emptyset$
$list_4 = \{(2,0),(2,1)\}$
$list_5 = \emptyset$
$list_6 = \{(3,0),(3,1)\}$
$list_7 = \emptyset$

# Abstraction Example: Shrink Step

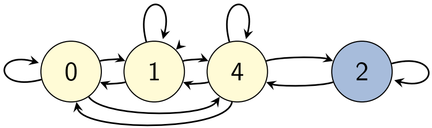
2. When combining $i$ and $j$, splice $list_j$ into $list_i$.




$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0),(1,1)\}$
$list_3 = \emptyset$
$list_4 = \{(2,0),(2,1),(3,0),(3,1)\}$
$list_5 = \emptyset$
$list_6 = \emptyset$
$list_7 = \emptyset$

# Abstraction Example: Shrink Step

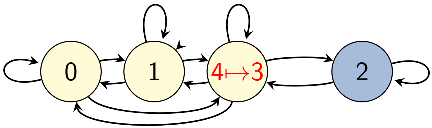
2. When combining $i$ and $j$, splice $list_j$ into $list_i$.




$$
\begin{aligned}
list_0 &= \{(0,0)\} \\
list_1 &= \{(0,1)\} \\
list_2 &= \{(1,0),(1,1)\} \\
list_3 &= \emptyset \\
list_4 &= \{(2,0),(2,1), \\
&\quad (3,0),(3,1)\} \\
list_5 &= \emptyset \\
list_6 &= \emptyset \\
list_7 &= \emptyset
\end{aligned}
$$

# Abstraction Example: Shrink Step

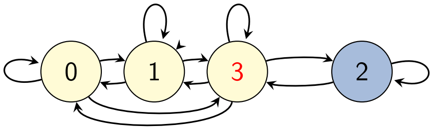
3. Renumber abstract states consecutively.




$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0),(1,1)\}$
$list_3 = \emptyset$
$list_4 = \{(2,0),(2,1),$
$(3,0),(3,1)\}$
$list_5 = \emptyset$
$list_6 = \emptyset$
$list_7 = \emptyset$

# Abstraction Example: Shrink Step

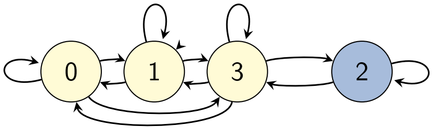
3. Renumber abstract states consecutively.




$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0),(1,1)\}$
$list_3 = \{(2,0),(2,1),(3,0),(3,1)\}$
$list_4 = \emptyset$
$list_5 = \emptyset$
$list_6 = \emptyset$
$list_7 = \emptyset$

# Abstraction Example: Shrink Step

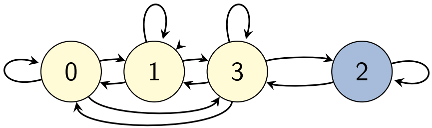
4. Regenerate the mapping table from the linked lists.




$$
\begin{aligned}
list_0 &= \{(0,0)\} \\
list_1 &= \{(0,1)\} \\
list_2 &= \{(1,0),(1,1)\} \\
list_3 &= \{(2,0),(2,1), \\
&\quad (3,0),(3,1)\} \\
list_4 &= \emptyset \\
list_5 &= \emptyset \\
list_6 &= \emptyset \\
list_7 &= \emptyset
\end{aligned}
$$

# Abstraction Example: Shrink Step

4. Regenerate the mapping table from the linked lists.




$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0),(1,1)\}$
$list_3 = \{(2,0),(2,1),$
$(3,0),(3,1)\}$
$list_4 = \emptyset$
$list_5 = \emptyset$
$list_6 = \emptyset$
$list_7 = \emptyset$

| | $s_2 = 0$ | $s_2 = 1$ |
|---|---|---|
| $s_1 = 0$ | 0 | 1 |
| $s_1 = 1$ | 2 | 2 |
| $s_1 = 2$ | 3 | 3 |
| $s_1 = 3$ | 3 | 3 |

# The Final Heuristic Representation

At the end, our heuristic is represented by six tables:

- three one-dimensional tables for the atomic abstractions:

| $T_{\text{package}}$ | $L$ | $R$ | $A$ | $B$ |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |

| $T_{\text{truck A}}$ | $L$ | $R$ |
|---|---|---|
| | 0 | 1 |

| $T_{\text{truck B}}$ | $L$ | $R$ |
|---|---|---|
| | 0 | 1 |

- two tables for the two merge and subsequent shrink steps:

| $T^1_{\text{m\&s}}$ | $s_2 = 0$ | $s_2 = 1$ |
|---|---|---|
| $s_1 = 0$ | 0 | 1 |
| $s_1 = 1$ | 2 | 2 |
| $s_1 = 2$ | 3 | 3 |
| $s_1 = 3$ | 3 | 3 |

| $T^2_{\text{m\&s}}$ | $s_2 = 0$ | $s_2 = 1$ |
|---|---|---|
| $s_1 = 0$ | 1 | 1 |
| $s_1 = 1$ | 1 | 0 |
| $s_1 = 2$ | 2 | 2 |
| $s_1 = 3$ | 3 | 3 |

- one table with goal distances for the final transition system:

| $T_h$ | $s = 0$ | $s = 1$ | $s = 2$ | $s = 3$ |
|---|---|---|---|---|
| $h(s)$ | 3 | 2 | 0 | 1 |

Given a state $s = \{\text{package} \mapsto L, \text{truck A} \mapsto L, \text{truck B} \mapsto R\}$, its heuristic value is then looked up as:

- $h(s) = T_h[T^2_{\text{m\&s}}[T^1_{\text{m\&s}}[T_{\text{package}}[L], T_{\text{truck A}}[L]], T_{\text{truck B}}[R]]]$

# Summary

# Summary (1)

- Merge-and-shrink abstractions are constructed by iteratively transforming the factored transition system of a planning task.
- Merge transformations combine two factors into their synchronized product.
- Shrink transformations reduce the size of a factor by abstracting it.

# Summary (1)

- Merge-and-shrink abstractions are constructed by iteratively transforming the factored transition system of a planning task.
- Merge transformations combine two factors into their synchronized product.
- Shrink transformations reduce the size of a factor by abstracting it.
- Merge-and-shrink abstractions are represented by a set of reference tables, one for each atomic abstraction and one for each merge-and-shrink step.
- The heuristic representation uses an additional table for the goal distances in the final abstract transition system.

# Summary (2)

- Projections of $\text{SAS}^+$ tasks correspond to merges of atomic factors.
- By also including shrinking, merge-and-shrink abstractions generalize projections: they can reflect all state variables, but in a potentially lossy way.