Principles of AI Planning

12. Planning as search: merge-and-shrink abstractions

Bernhard Nebel and Robert Mattmüller

Albert-Ludwigs-Universität Freiburg

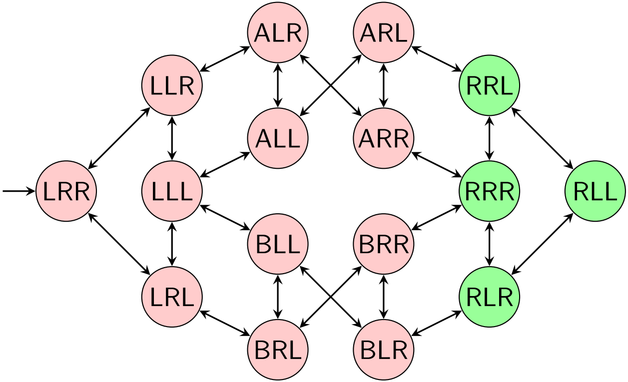
January 13th, 2012

Motivation

Beyond pattern databases

- Despite their popularity, pattern databases have some fundamental limitations ($\rightsquigarrow$ example on next slides).
- In this chapter, we study a 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

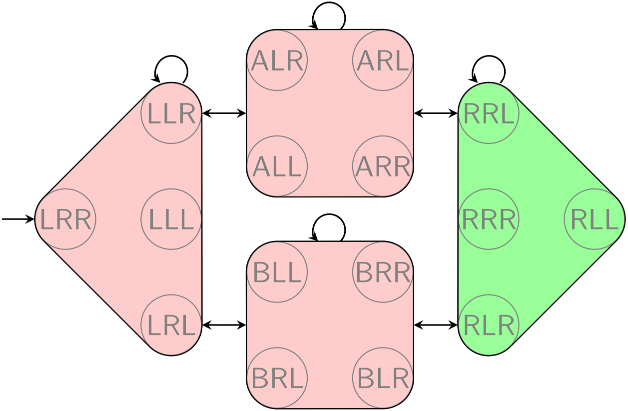


Logistics problem with one package, two trucks, two locations:

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

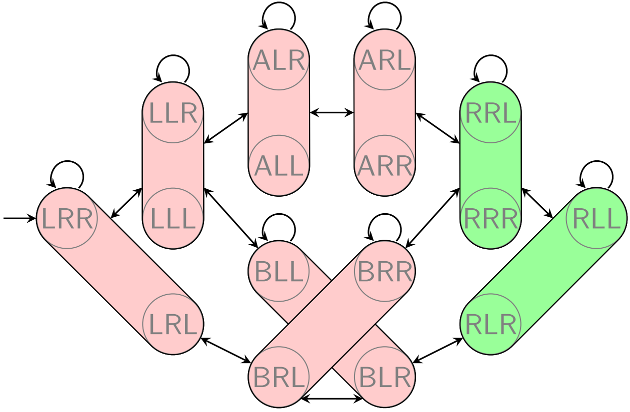
Example: projection

Project to {package}:



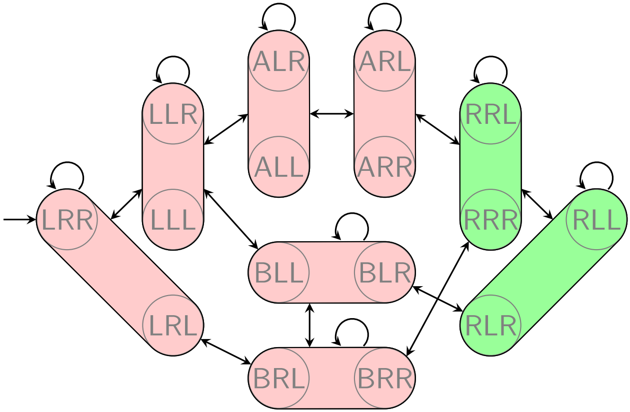
Example: projection (2)

Project to $\{\text{package}, \text{truck A}\}$:



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

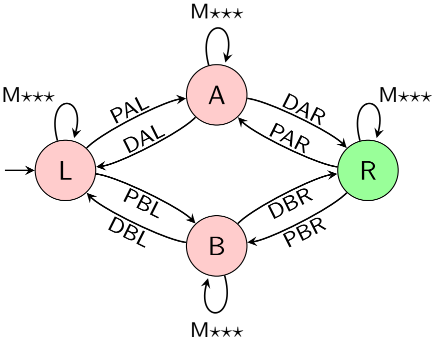
- The main idea underlying merge-and-shrink abstractions is that given two abstractions $\mathcal{A}$ and $\mathcal{A}'$, we can merge them into a new product abstraction.
- The product abstraction captures all information of both abstractions and can be better informed than either.
- It can even be 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

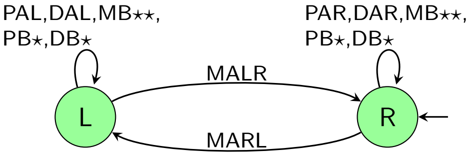
Running example: atomic projection for package

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



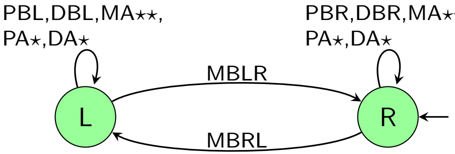
Running example: atomic projection for truck A

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



Running example: atomic projection for truck B

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



Synchronized products

Synchronized product of transition systems

Definition (synchronized product of transition systems)

For $i \in \{1, 2\}$, let $\mathcal{T}_i = \langle S_i, L, T_i, s_{0i}, S_{\star i}\rangle$ 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 $\mathcal{T}_\otimes = \langle S_\otimes, L, T_\otimes, s_{0\otimes}, S_{\star\otimes}\rangle$ 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 $\langle s_2, l, t_2\rangle \in T_2\}$
- $s_{0\otimes} := \langle s_{01}, s_{02}\rangle$
- $S_{\star\otimes} := S_{\star 1} \times S_{\star 2}$

Synchronized product of functions

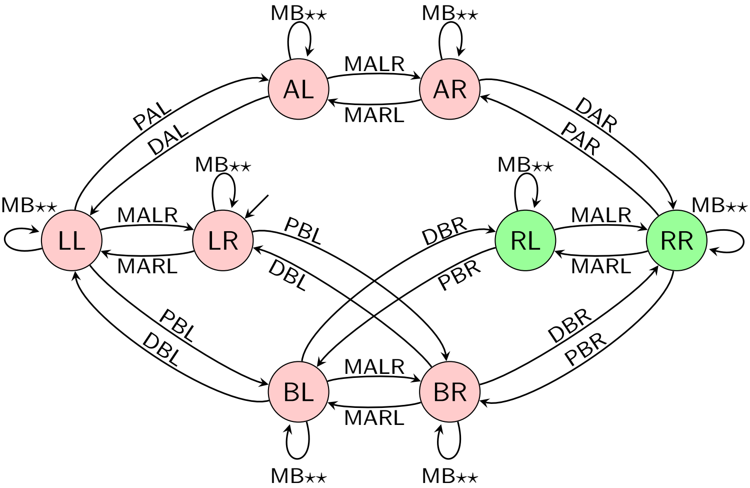
Definition (synchronized product of functions)

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

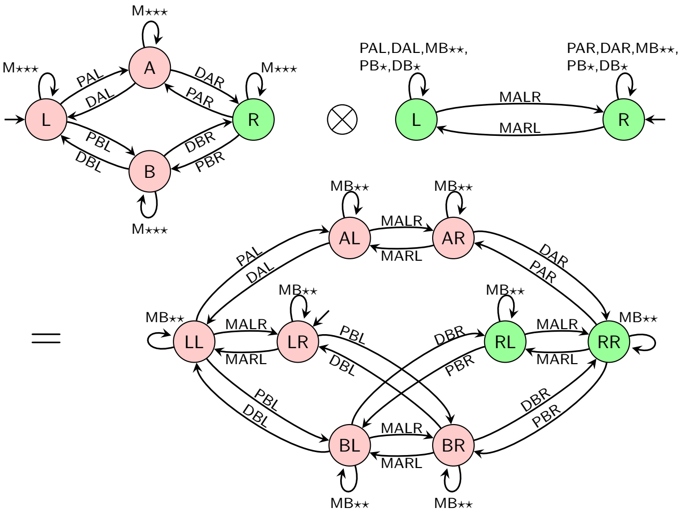
Example: synchronized product

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



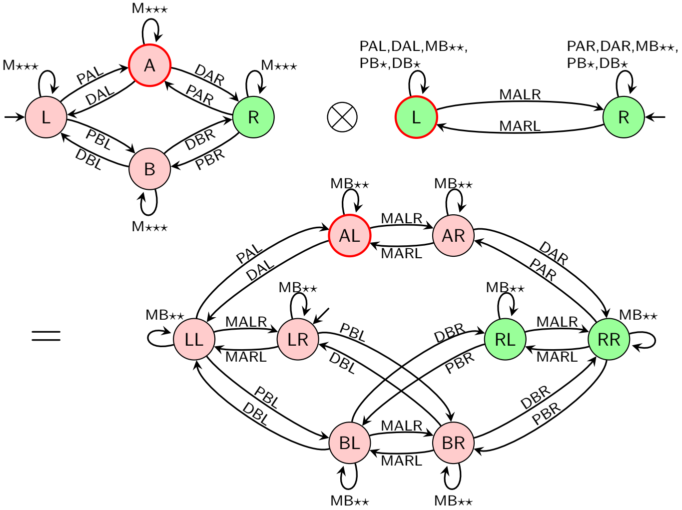
Example: computation of synchronized product

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



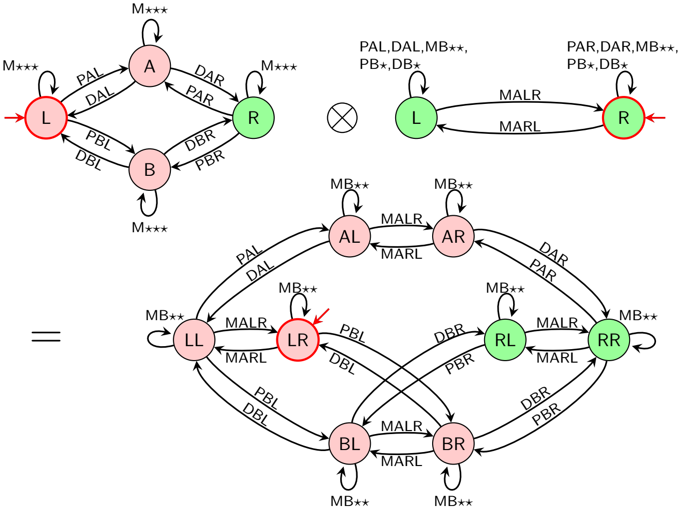
Example: computation of synchronized product

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



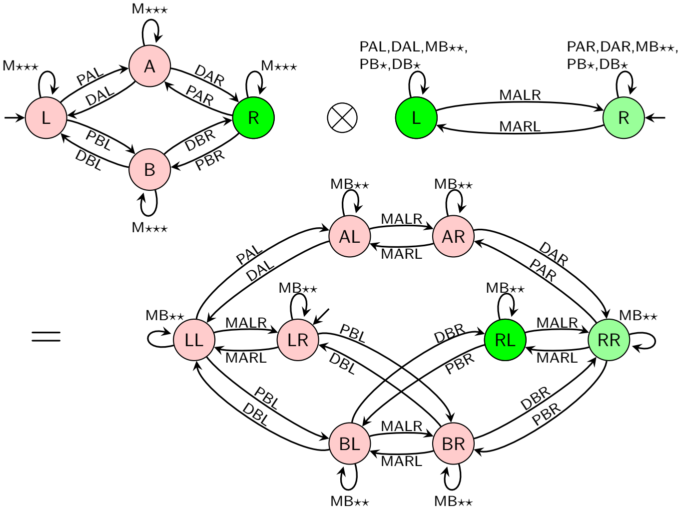
Example: computation of synchronized product

$\mathcal{T}^{\pi\{\text{package}\}} \otimes \mathcal{T}^{\pi\{\text{truck A}\}}$: $s_{0\otimes} = \langle s_{01}, s_{02}\rangle$



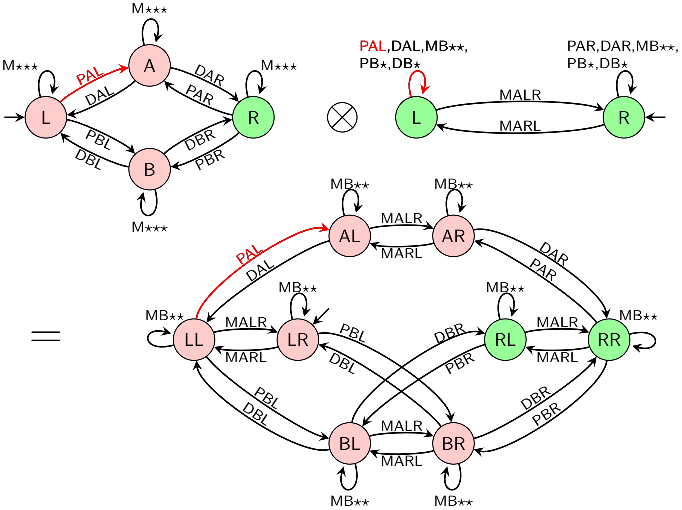
Example: computation of synchronized product

$\mathcal{T}^{\pi\{\text{package}\}} \otimes \mathcal{T}^{\pi\{\text{truck A}\}}$: $S_{\star\otimes} = S_{\star 1} \times S_{\star 2}$



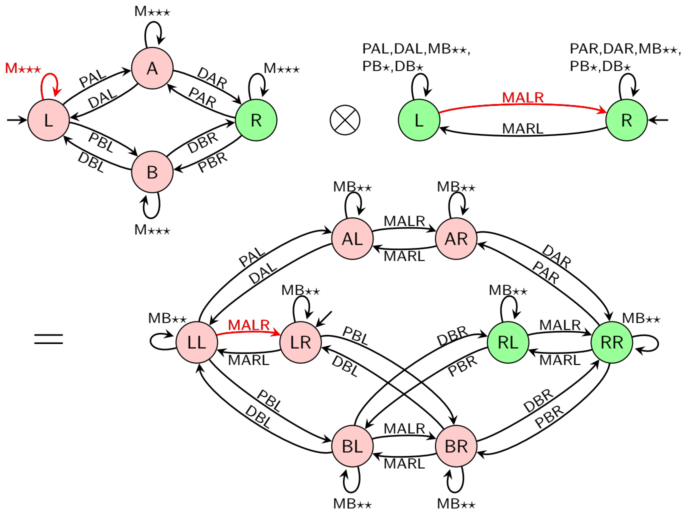
Example: computation of synchronized product

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



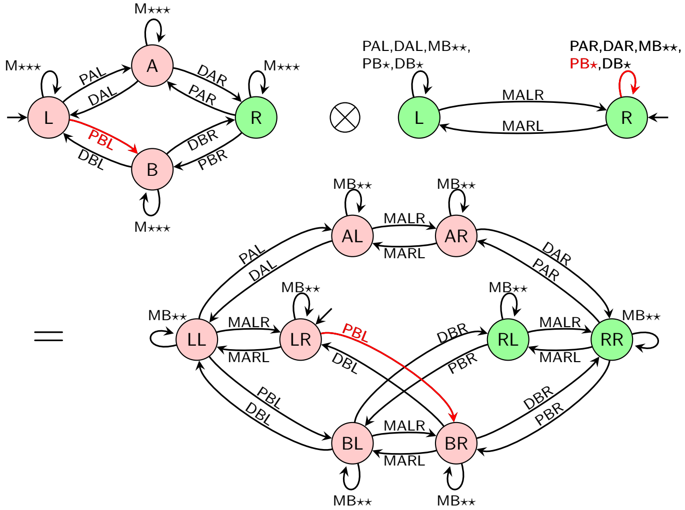
Example: computation of synchronized product

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



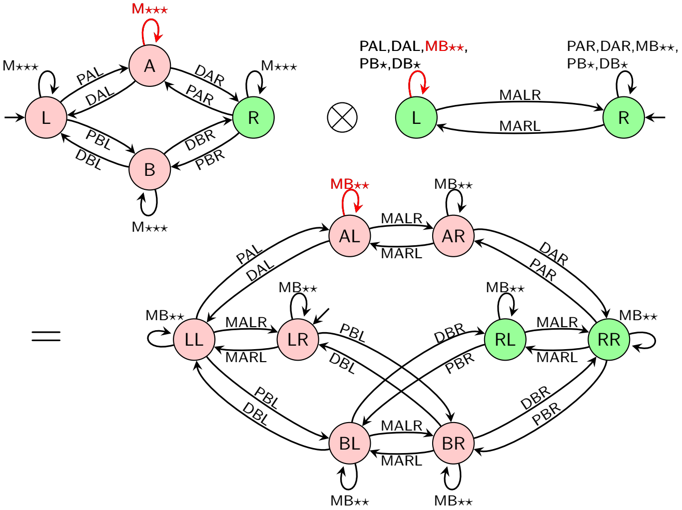
Example: computation of synchronized product

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



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 such that $\alpha_1 \otimes \alpha_2$ is surjective.

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 $\langle \mathcal{T}_2, \alpha_2 \rangle$.

Remark: If $\alpha_1 \otimes \alpha_2$ is not surjective, then the proof also holds if we restrict $\mathcal{T}_\otimes$ to the states in the image of $\alpha_1 \otimes \alpha_2$.

Synchronized products are abstractions (ctd.)

Proof.

We prove the first part. The refinement property is then easy to see: the mapping $\langle s_1, s_2 \rangle \mapsto s_i$ is a strict homomorphism from $\mathcal{T}_\otimes$ to $\mathcal{T}_i$ for $i \in \{1, 2\}$.

To show that $\mathcal{T}_\otimes$ is an abstraction of $\mathcal{T}$ with mapping $\alpha_\otimes$, we need to show that $\alpha_\otimes$ is surjective and preserves initial states, goal states and transitions.

Synchronized products are abstractions (ctd.)

Proof.

We prove the first part. The refinement property is then easy to see: the mapping $\langle s_1, s_2 \rangle \mapsto s_i$ is a strict homomorphism from $\mathcal{T}_\otimes$ to $\mathcal{T}_i$ for $i \in \{1, 2\}$.

To show that $\mathcal{T}_\otimes$ is an abstraction of $\mathcal{T}$ with mapping $\alpha_\otimes$, we need to show that $\alpha_\otimes$ is surjective and preserves initial states, goal states and transitions.

Synchronized products are abstractions (ctd.)

Proof (ctd.)

Let $\mathcal{T} = \langle S, L, T, s_0, S_\star \rangle$, and let $\mathcal{T}_i = \langle S_i, L, T_i, s_{0i}, S_{\star i} \rangle$ for $i \in \{1, 2, \otimes\}$.

- $\alpha_\otimes$ surjective: This is given in the premise.
- $\alpha_\otimes$ preserves the initial state:
  $\alpha_1(s_0) = s_{01}$, $\alpha_2(s_0) = s_{02}$ (abstraction property for $\mathcal{T}_1$, $\mathcal{T}_2$)
  $\rightsquigarrow \langle \alpha_1(s_0), \alpha_2(s_0) \rangle = s_{0\otimes}$ (definition of $s_{0\otimes}$)
  $\rightsquigarrow \alpha_\otimes(s_0) = s_{0\otimes}$ (definition of $\alpha_\otimes$)

Synchronized products are abstractions (ctd.)

Proof (ctd.)

Let $\mathcal{T} = \langle S, L, T, s_0, S_\star \rangle$, and let $\mathcal{T}_i = \langle S_i, L, T_i, s_{0i}, S_{\star i} \rangle$ for $i \in \{1, 2, \otimes\}$.

- $\alpha_\otimes$ surjective: This is given in the premise.
- $\alpha_\otimes$ preserves the initial state:
  $\alpha_1(s_0) = s_{01}$, $\alpha_2(s_0) = s_{02}$ (abstraction property for $\mathcal{T}_1$, $\mathcal{T}_2$)
  $\rightsquigarrow \langle \alpha_1(s_0), \alpha_2(s_0) \rangle = s_{0\otimes}$ (definition of $s_{0\otimes}$)
  $\rightsquigarrow \alpha_\otimes(s_0) = s_{0\otimes}$ (definition of $\alpha_\otimes$)

Synchronized products are abstractions (ctd.)

Proof (ctd.)

- $\alpha_\otimes$ preserves goal states:
  Let $s \in S_\star$.
  $\rightsquigarrow \alpha_1(s) \in S_{\star 1}$, $\alpha_2(s) \in S_{\star 2}$ (abstraction property for $\mathcal{T}_1$, $\mathcal{T}_2$)
  $\rightsquigarrow \langle \alpha_1(s), \alpha_2(s) \rangle \in S_{\star \otimes}$ (definition of $\mathcal{S}_{\star \otimes}$)
  $\rightsquigarrow \alpha_\otimes(s) \in S_{\star \otimes}$ (definition of $\alpha_\otimes$)
- $\alpha_\otimes$ preserves transitions:
  Let $\langle s, l, t \rangle \in T$.
  $\rightsquigarrow \langle \alpha_1(s), l, \alpha_1(t) \rangle \in T_1$, $\langle \alpha_2(s), l, \alpha_2(t) \rangle \in T_2$
  $\rightsquigarrow \langle \langle \alpha_1(s), \alpha_2(s) \rangle, l, \langle \alpha_1(t), \alpha_2(t) \rangle \rangle \in T_\otimes$
  $\rightsquigarrow \langle \alpha_\otimes(s), l, \alpha_\otimes(t) \rangle \in T_\otimes$

$\square$

Synchronized products are abstractions (ctd.)

Proof (ctd.)

- $\alpha_\otimes$ preserves goal states:
  Let $s \in S_\star$.
  $\rightsquigarrow \alpha_1(s) \in S_{\star 1}$, $\alpha_2(s) \in S_{\star 2}$ (abstraction property for $\mathcal{T}_1$, $\mathcal{T}_2$)
  $\rightsquigarrow \langle \alpha_1(s), \alpha_2(s) \rangle \in S_{\star\otimes}$ (definition of $\mathcal{S}_{\star\otimes}$)
  $\rightsquigarrow \alpha_\otimes(s) \in S_{\star\otimes}$ (definition of $\alpha_\otimes$)
- $\alpha_\otimes$ preserves transitions:
  Let $\langle s, l, t \rangle \in T$.
  $\rightsquigarrow \langle \alpha_1(s), l, \alpha_1(t) \rangle \in T_1$, $\langle \alpha_2(s), l, \alpha_2(t) \rangle \in T_2$
  $\rightsquigarrow \langle \langle \alpha_1(s), \alpha_2(s) \rangle, l, \langle \alpha_1(t), \alpha_2(t) \rangle \rangle \in T_\otimes$
  $\rightsquigarrow \langle \alpha_\otimes(s), l, \alpha_\otimes(t) \rangle \in T_\otimes$

$\square$

Preserving strict homomorphisms

- It would be very nice if we could also prove that if $\mathcal{T}_1$ and $\mathcal{T}_2$ are strictly homomorphic abstractions, then so is $\mathcal{T}_1 \otimes \mathcal{T}_2$.
- However, this is not true in general.
- It is not even true for SAS$^+$ tasks.
- However, there is an important sufficient condition for preserving the strict homomorphism property.

Synchronized products and strict homomorphisms

Theorem (synchronized products and strict homomorphisms)

Let Π be an SAS$^+$ planning task with variable set V, and let V_1 and V_2 be disjoint subsets of V.

For $i \in \{1, 2\}$, let $\mathcal{T}_i$ be a strictly homomorphic abstraction of $\mathcal{T}(\Pi)$ with mapping α_i such that $\langle \mathcal{T}_i, \alpha_i \rangle$ is a coarsening of $\langle \mathcal{T}^{\pi_{V_i}}, \pi_{V_i} \rangle$.

Then $\alpha_\otimes := \alpha_1 \otimes \alpha_2$ is surjective and $\mathcal{T}_\otimes := \mathcal{T}_1 \otimes \mathcal{T}_2$ is a strict homomorphic abstraction of $\mathcal{T}(\Pi)$ with mapping $\alpha_\otimes$.

(Proof omitted.)

Note: In this special case, we do not need to require that $\alpha_\otimes$ is surjective but can conclude it from the other premises.

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 $\mathcal{T}^{\pi_{V_1}} \otimes \mathcal{T}^{\pi_{V_2}} \sim \mathcal{T}^{\pi_{V_1 \cup V_2}}$.

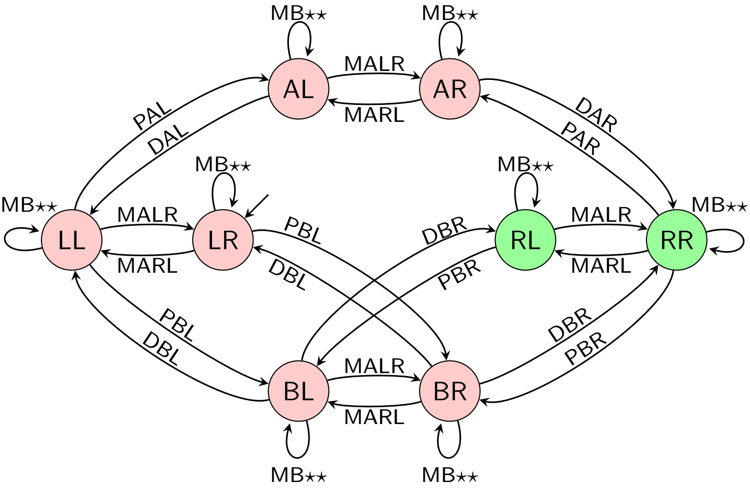
Proof.

- By the theorem, $\mathcal{T}_\otimes := T^{\pi_{V_1}} \otimes \mathcal{T}^{\pi_{V_2}}$ is a strictly homomorphic abstraction of $\mathcal{T}(\Pi)$ with mapping $\pi_{V_1} \otimes \pi_{V_2}$.
- $\mathcal{T}^{\pi_{V_1 \cup V_2}}$ is a strictly homomorphic abstraction of $\mathcal{T}(\Pi)$ with mapping $\pi_{V_1 \cup V_2}$.

$\pi_{V_1} \otimes \pi_{V_2}$ and $\pi_{V_1 \cup V_2}$ are identical functions up to renaming of abstract states, and strictly homomorphic abstractions are uniquely determined by the abstraction function, so the abstractions must be isomorphic. □

Example: product for disjoint projections

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



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 $\mathrm{id} = \pi_V$.

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

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

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

Algorithm

Generic merge-and-shrink abstractions: outline

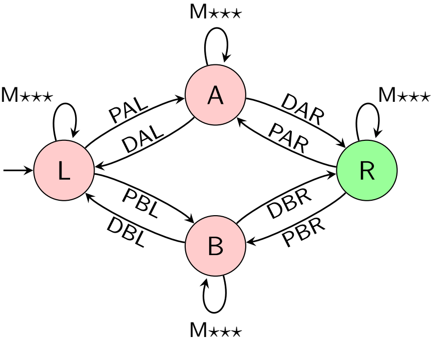
Using the results from the previous section, we can develop the ideas of a generic abstraction computation procedure that takes all state variables into account:

- 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 strict homomorphism to them).

We explain these steps with our running example.

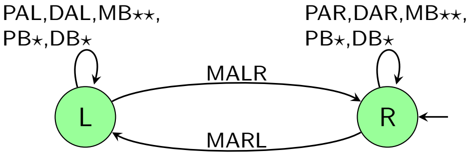
Initialization step: atomic projection for package

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



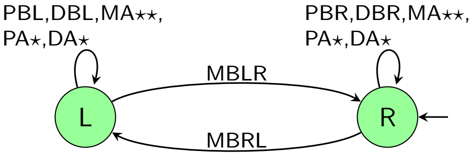
Initialization step: atomic projection for truck A

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



Initialization step: atomic projection for truck B

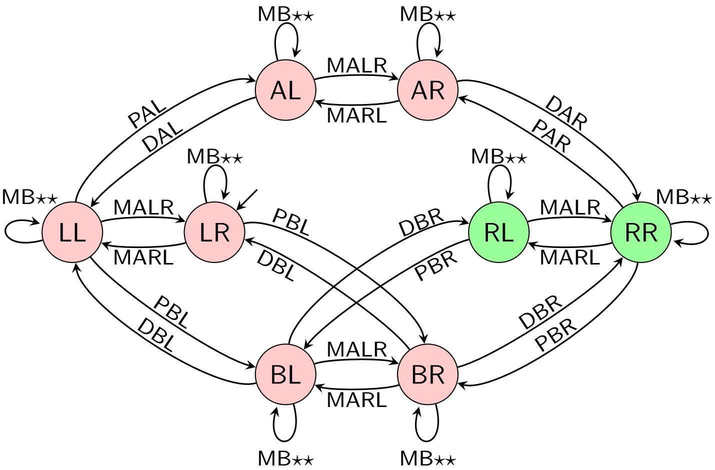
$\mathcal{T}^{\pi_{\{\text{truck B}\}}}$:



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

First merge step

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



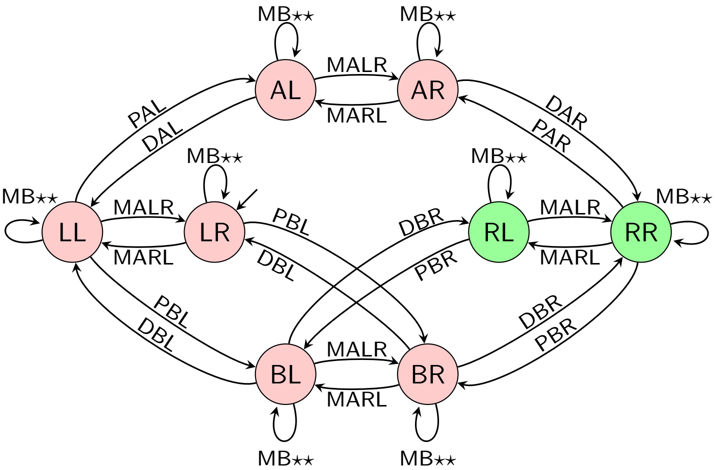
current collection: $\{\mathcal{T}_1, \mathcal{T}^{\pi_{\{\text{truck B}\}}}\}$

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.

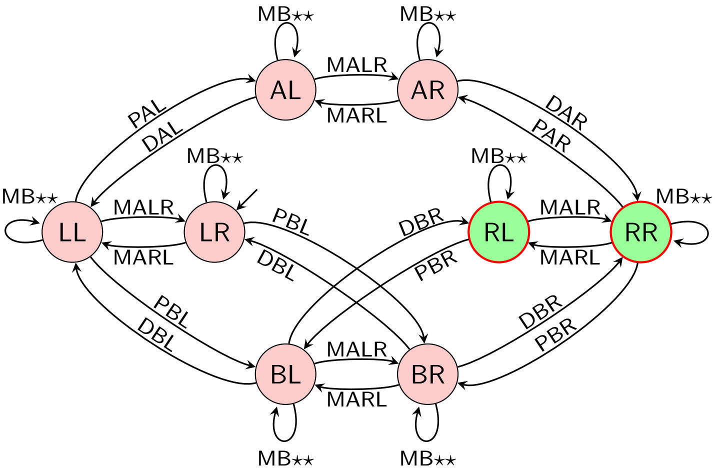
First shrink step

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



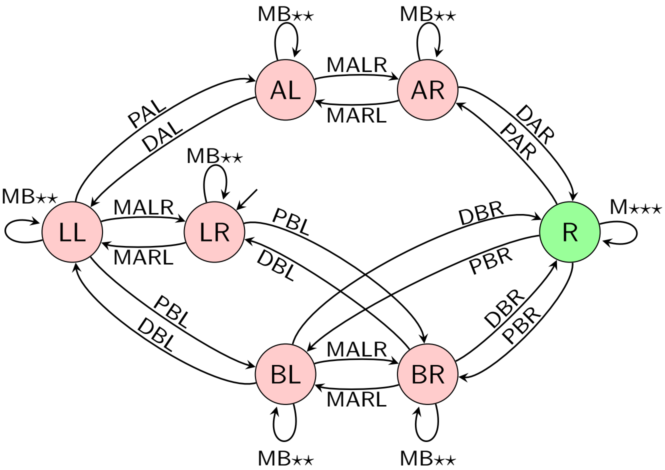
First shrink step

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



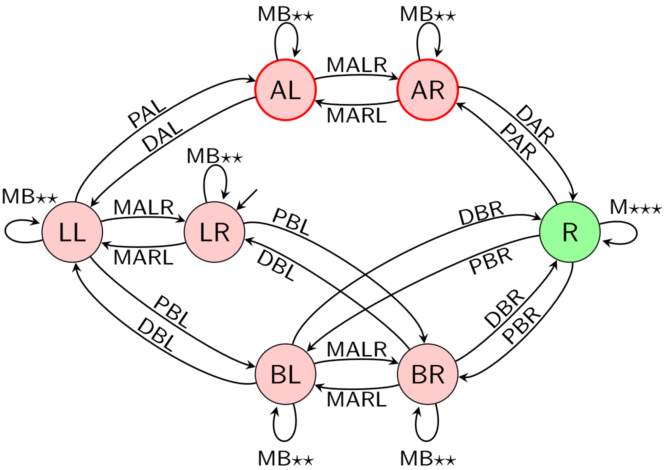
First shrink step

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



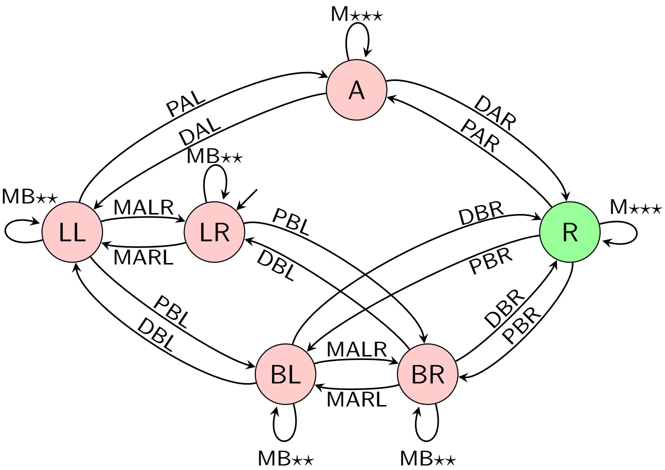
First shrink step

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



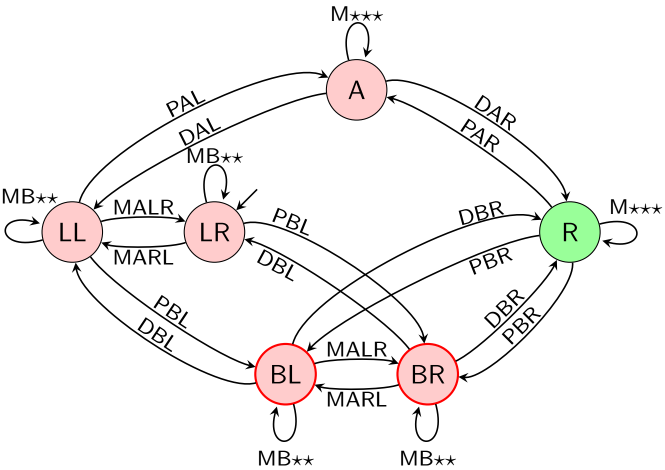
First shrink step

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



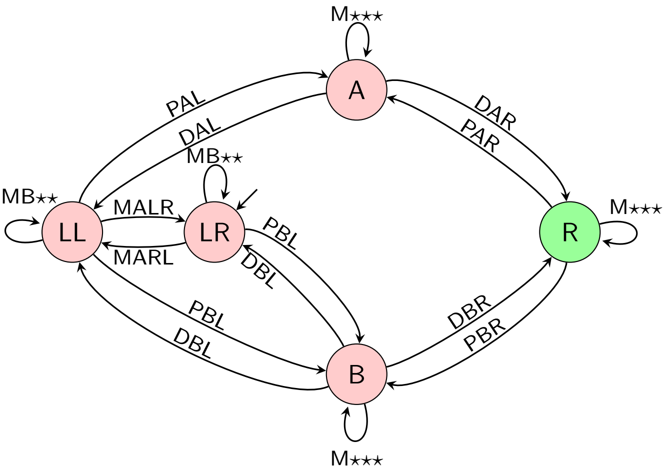
First shrink step

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



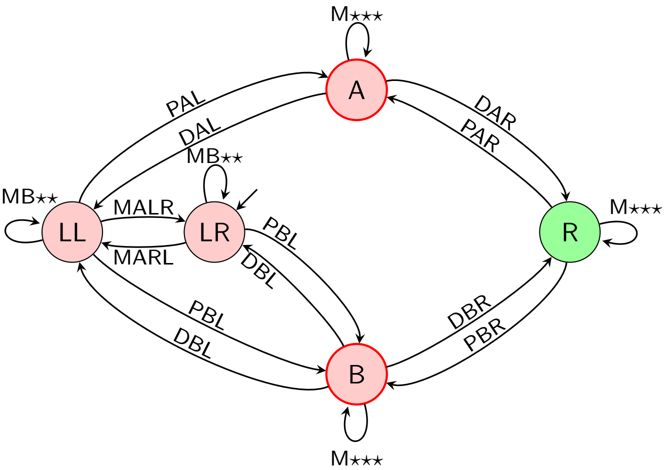
First shrink step

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



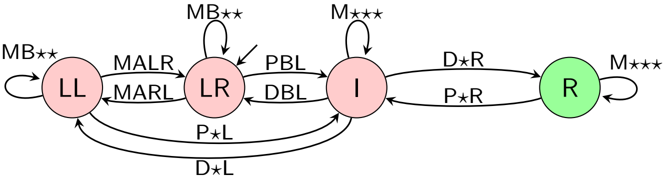
First shrink step

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



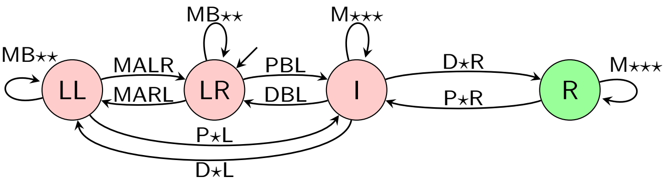
First shrink step

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



First shrink step

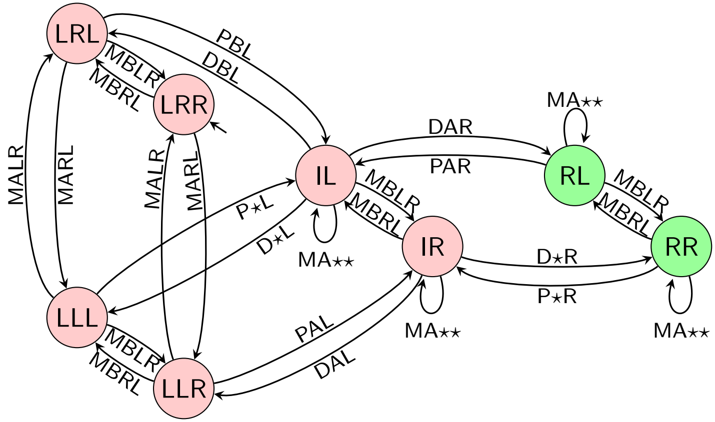
$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



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

Second merge step

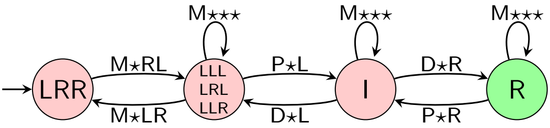
$\mathcal{T}_3 := \mathcal{T}_2 \otimes \mathcal{T}^{\pi_{\{\text{truck B}\}}}$:



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 to keep the table in sync with the abstraction choices.

How to represent the abstraction mapping? (ctd.)

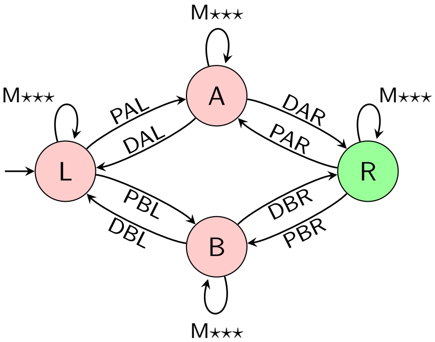
Idea: the computation of the abstraction mapping follows the sequence of product computations

- 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.

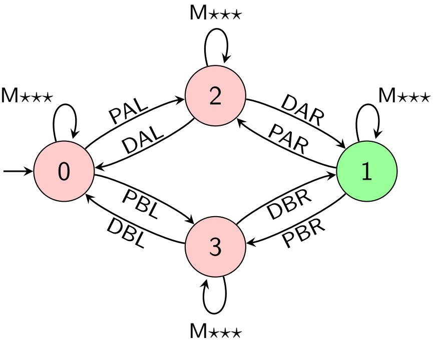
Abstraction mapping example: atomic abstractions

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

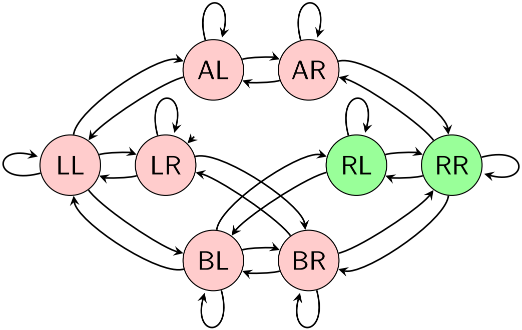
Computing abstraction mappings for the atomic abstractions is simple. Just number the states (domain values) consecutively and generate a table of references to the states:



L	R	A	B
0	1	2	3

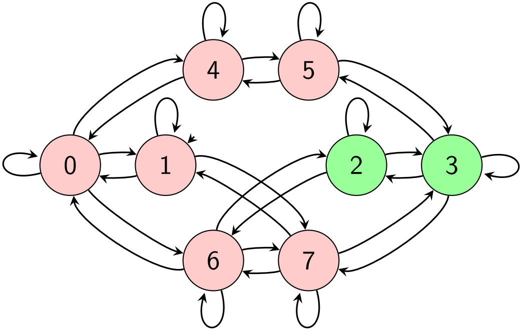
Abstraction mapping example: merge step

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

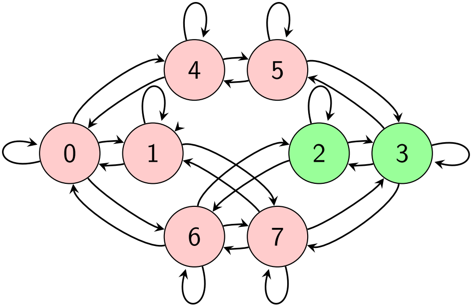
- The hard part in representing the abstraction mapping is to keep it consistent when shrinking.
- In theory, this is easy to do:
  - When combining states i and j, arbitrarily use one of them (say i) as the number of the new state.
  - Find all table entries in the table for this abstraction which map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

Maintaining the mapping 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 mapping example: shrink step

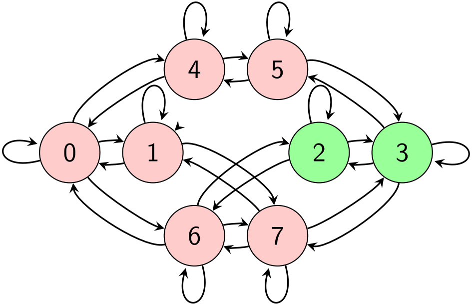
Representation before shrinking:



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

Abstraction mapping example: shrink step

1. Convert table to linked lists and discard it.

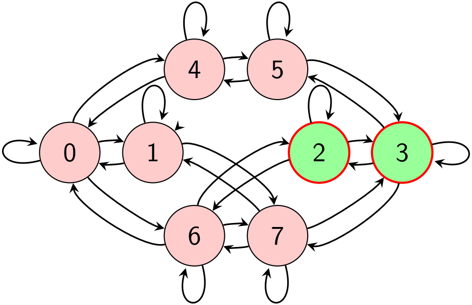


$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0)\}$
$list_3 = \{(1,1)\}$
$list_4 = \{(2,0)\}$
$list_5 = \{(2,1)\}$
$list_6 = \{(3,0)\}$
$list_7 = \{(3,1)\}$

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

Abstraction mapping example: shrink step

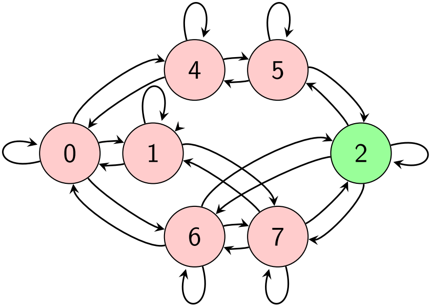
2. When combining i and j, splice list_j into list_i.



$\mathit{list}_0 = \{(0,0)\}$
$\mathit{list}_1 = \{(0,1)\}$
$\mathit{list}_2 = \{(1,0)\}$
$\mathit{list}_3 = \{(1,1)\}$
$\mathit{list}_4 = \{(2,0)\}$
$\mathit{list}_5 = \{(2,1)\}$
$\mathit{list}_6 = \{(3,0)\}$
$\mathit{list}_7 = \{(3,1)\}$

Abstraction mapping example: shrink step

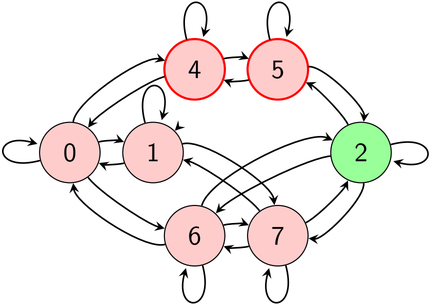
2. When combining i and j, splice $list_j$ into $list_i$.



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

Abstraction mapping example: shrink step

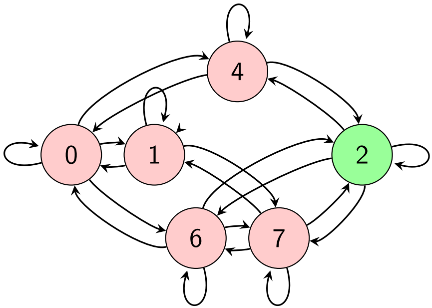
2. When combining i and j, splice list_j into list_i.



$\mathit{list}_0 = \{(0,0)\}$
$\mathit{list}_1 = \{(0,1)\}$
$\mathit{list}_2 = \{(1,0),(1,1)\}$
$\mathit{list}_3 = \emptyset$
$\mathit{list}_4 = \{(2,0)\}$
$\mathit{list}_5 = \{(2,1)\}$
$\mathit{list}_6 = \{(3,0)\}$
$\mathit{list}_7 = \{(3,1)\}$

Abstraction mapping example: shrink step

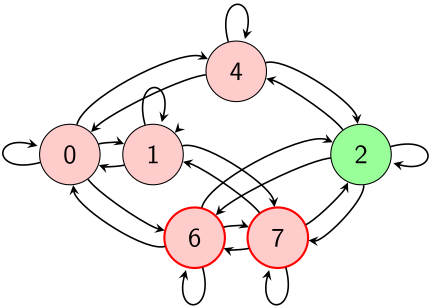
2. When combining i and j, splice list_j into list_i.



$\mathit{list}_0 = \{(0,0)\}$
$\mathit{list}_1 = \{(0,1)\}$
$\mathit{list}_2 = \{(1,0),(1,1)\}$
$\mathit{list}_3 = \emptyset$
$\mathit{list}_4 = \{(2,0),(2,1)\}$
$\mathit{list}_5 = \emptyset$
$\mathit{list}_6 = \{(3,0)\}$
$\mathit{list}_7 = \{(3,1)\}$

Abstraction mapping example: shrink step

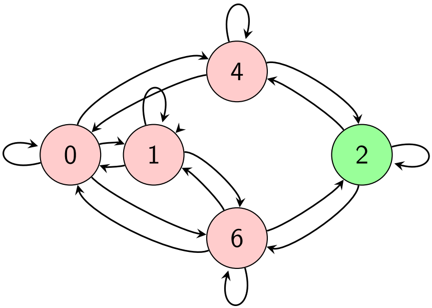
2. When combining i and j, splice $list_j$ into $list_i$.



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

Abstraction mapping example: shrink step

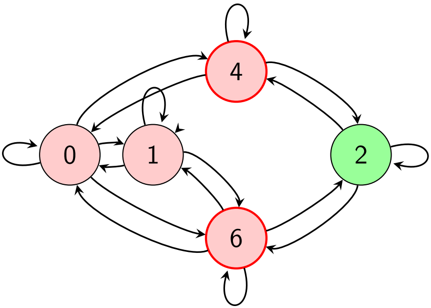
2. When combining i and j, splice list_j into list_i.



$\mathit{list}_0 = \{(0,0)\}$
$\mathit{list}_1 = \{(0,1)\}$
$\mathit{list}_2 = \{(1,0),(1,1)\}$
$\mathit{list}_3 = \emptyset$
$\mathit{list}_4 = \{(2,0),(2,1)\}$
$\mathit{list}_5 = \emptyset$
$\mathit{list}_6 = \{(3,0),(3,1)\}$
$\mathit{list}_7 = \emptyset$

Abstraction mapping example: shrink step

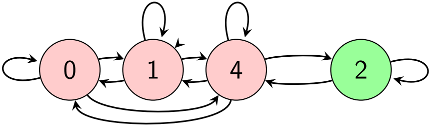
2. When combining i and j, splice $list_j$ into $list_i$.



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

Abstraction mapping example: shrink step

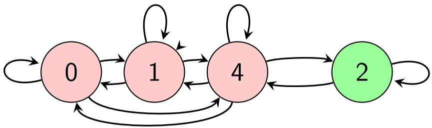
2. When combining i and j, splice list_j into list_i.



$\mathit{list}_0 = \{(0,0)\}$
$\mathit{list}_1 = \{(0,1)\}$
$\mathit{list}_2 = \{(1,0),(1,1)\}$
$\mathit{list}_3 = \emptyset$
$\mathit{list}_4 = \{(2,0),(2,1),(3,0),(3,1)\}$
$\mathit{list}_5 = \emptyset$
$\mathit{list}_6 = \emptyset$
$\mathit{list}_7 = \emptyset$

Abstraction mapping example: shrink step

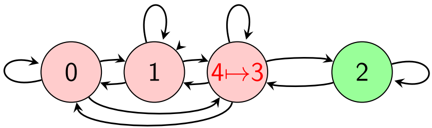
2. When combining i and j, splice $list_j$ into $list_i$.



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

Abstraction mapping example: shrink step

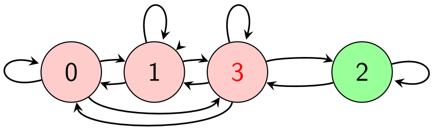
3. Renumber abstract states consecutively.



$\mathit{list}_0 = \{(0,0)\}$
$\mathit{list}_1 = \{(0,1)\}$
$\mathit{list}_2 = \{(1,0),(1,1)\}$
$\mathit{list}_3 = \emptyset$
$\mathit{list}_4 = \{(2,0),(2,1),(3,0),(3,1)\}$
$\mathit{list}_5 = \emptyset$
$\mathit{list}_6 = \emptyset$
$\mathit{list}_7 = \emptyset$

Abstraction mapping example: shrink step

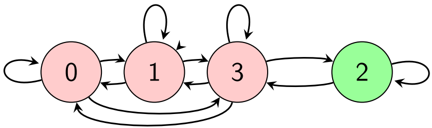
3. Renumber abstract states consecutively.



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

Abstraction mapping example: shrink step

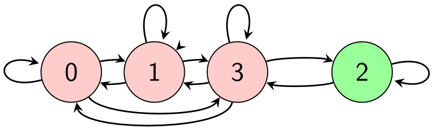
4. Regenerate the mapping table from the linked lists.



$$
\begin{aligned}
\mathit{list}_0 &= \{(0,0)\}\\
\mathit{list}_1 &= \{(0,1)\}\\
\mathit{list}_2 &= \{(1,0),(1,1)\}\\
\mathit{list}_3 &= \{(2,0),(2,1),\\
&\qquad (3,0),(3,1)\}\\
\mathit{list}_4 &= \emptyset\\
\mathit{list}_5 &= \emptyset\\
\mathit{list}_6 &= \emptyset\\
\mathit{list}_7 &= \emptyset
\end{aligned}
$$

Abstraction mapping 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_{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 abstraction:

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

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

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

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 (cf. causal graph criteria for growing patterns)

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?

$h_{\textsf{HHH}}$: 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

$h_{\textsf{HHH}}$: 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

Conclusion

Properties of merge-and-shrink abstractions

- We conclude by briefly mentioning a number of theoretical properties of merge-and-shrink abstractions (without proof).
- While these theoretical results are interesting, heuristics in planning usually need to be justified by good empirical performance.
- Regarding empirical performance, initial results for h_{HHH} are very encouraging, outperforming pattern databases (and all other admissible heuristics) on a number of benchmark domains.
- However, merge-and-shrink abstractions are not nearly as well studied (and understood) as pattern databases, so the jury is still out.

Theoretical properties: as good as PDBs

As powerful as PDBs

Pattern database heuristics are a special case of our abstraction heuristics, and arise naturally as a side product.

- More precisely, PDB heuristics are merge-and-shrink abstractions without shrink steps (terminating heuristic computation as soon as space runs out).
- However, specialized PDB algorithms are faster than the generic merge-and-shrink algorithm.
- This performance difference is only polynomial, but this does not mean that it does not matter in practice!
- Still, this shows that representational power is at least as large as that of PDB heuristics.

Theoretical properties: better than PDBs

Greater representational power

In some planning domains where polynomial-sized pattern database heuristics have unbounded error (Gripper, Schedule, two Promela variants), merge-and-shrink abstractions can obtain perfect heuristics in polynomial time with suitable merging/shrinking strategies.

- This shows that representational power is strictly greater than that of PDB heuristics.
- However, it does not mean that we know good general (domain-independent) merging/shrinking strategies that will generate these perfect heuristics in practice.

Theoretical properties: additivity

Get additivity for free

If P_1 and P_2 are additive patterns of a SAS$^+$ task, then for all h-preserving merge-and-shrink abstractions $\mathcal{A}_1$ of $\mathcal{T}^{\pi_{P_1}}$, $\mathcal{A}_2$ of $\mathcal{T}^{\pi_{P_2}}$ and $\mathcal{A}$ of $\mathcal{A}_1 \otimes \mathcal{A}_2$, the abstraction heuristic for $\mathcal{A}$ dominates $h^{P_1} + h^{P_2}$. (An abstraction is h-preserving if $\alpha(s) = \alpha(s')$ only for s, s' with same abstract goal distance.)

- One can derive a similar theory of additivity for merge-and-shrink abstraction as for pattern databases.
- However, this result shows that this is not as necessary as for pattern databases: additivity is exploited automatically by a single merge-and-shrink abstraction to some extent.
- Still, experimental results show that there is sometimes a benefit of using multiple merge-and-shrink abstractions. (However, so far only maximization has been explored.)

Literature

References on merge-and-shrink abstractions:

- Klaus Dräger, Bernd Finkbeiner and Andreas Podelski.
  Directed Model Checking with Distance-Preserving Abstractions.
  Proc. SPIN 2006, pp. 19–34, 2006.
  Introduces merge-and-shrink abstractions (for model-checking).

- Malte Helmert, Patrik Haslum and Jörg Hoffmann.
  Flexible Abstraction Heuristics for Optimal Sequential Planning.
  Proc. ICAPS 2007, pp. 176–183, 2007.
  Introduces merge-and-shrink abstractions for planning.
  Most ideas of this chapter come from this paper.

Summary

- Projections perfectly reflect a few state variables. Merge-and-shrink abstractions are a generalization that can reflect all state variables, but in a potentially lossy way.
- The merge steps combine two abstractions by replacing them with their synchronized product.
- The shrink steps make an abstraction smaller by abstracting it further.
- The resulting abstraction mapping is represented by a set of reference tables.