AI Planning

13. Merge-and-Shrink Heuristics

It's a Long Way to the Goal, But How Long Exactly?
Part III, How-To (B): *Incremental Ignorance*

Álvaro Torralba, Cosmina Croitoru

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Winter Term 2018/2019

Thanks to Prof. Jörg Hoffmann for slide sources

Agenda

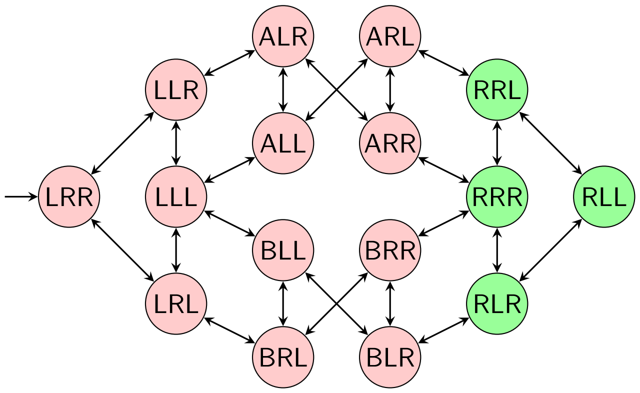
Reminder: Our Program for Abstraction Heuristics

We take a look at abstractions and their use for generating admissible heuristic functions:

- In Chapter 11, we formally introduced abstractions and abstraction heuristics and studied some of their most important properties.
- In Chapter 12, we discussed a particular class of abstraction heuristics and its practical handling in detail, namely pattern database heuristics.
- In This Chapter, we discuss another particular class of abstraction heuristics and its practical handling in detail, namely merge-and-shrink abstractions.

→ We handle all these methods in FDR, where they are most natural. We do not mention STRIPS at all (which is a special case anyway).

PDBs in "Logistics mal anders"

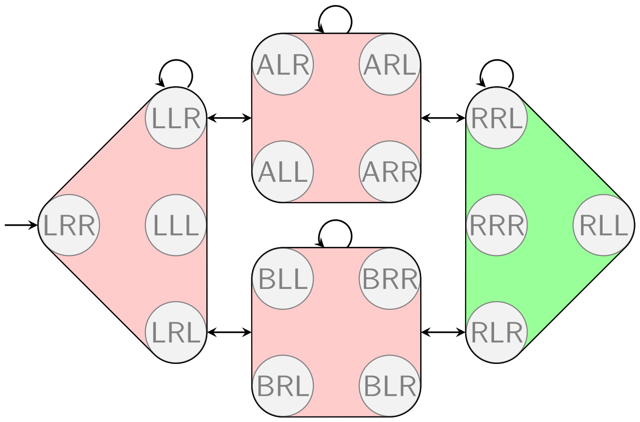


Logistics task with one package, two trucks, two locations:

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

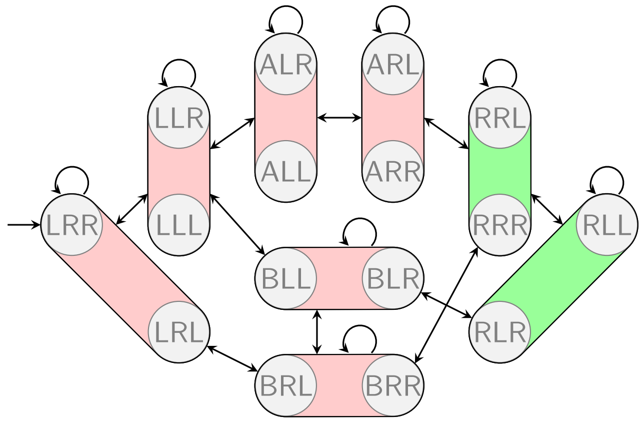
PDBs in "Logistics mal anders": Projection 1

Project to {package}:



PDBs in "Logistics mal anders": Projection 2

Project to {package, truck A}:



On the Limitations of PDBs

How informed is the PDB heuristic?

- Consider generalization of the example: N trucks (still 1 package).
- Consider any pattern P that is proper subset of variable set V.
- Then $h^P(I) \leq$

→ Can we improve this by max'ing over patterns?

→ Can we improve this by sum'ing over orthogonal patterns?

→ By contrast: Merge-and-shrink abstractions can, in time and space polynomial in N, represent the perfect heuristic!

Our Agenda for This Chapter

2. **The Merge-and-Shrink Framework:** Introduces the algorithm and discusses basic properties. Illustration on blackboard.
3. **The Expressive Power of Merge-and-Shrink Abstractions:** Not as scary as it sounds. We prove that merge-and-shrink can, with polynomial overhead, simulate PDBs; but not vice versa."
4. **Concrete Merge-and-Shrink Strategies:** The algorithm has several important choice points. One can rely on the well-known concept of bisimulation, and approximations thereof, to intelligently take some of these choices.
5. **A Full Example: "Logistics mal anders":** Fully specified illustration of that example.
6. **M&S Abstraction Mappings:** We disregard abstraction mappings in the above, and explain their treatment here.

Merge-and-Shrink in a Nutshell

Algorithm Outline:

1. Initialize step: Compute the abstract state space $\Theta^{\{v\}}$ of atomic projections $\pi_{\{v\}}$ to form the initial abstraction collection.
2. Merge step: Combine two abstractions Θ_1 and Θ_2 in the collection by replacing them with their synchronized product $\Theta_1 \otimes \Theta_2$.
3. Shrink step: Make some abstraction in the collection (typically but not necessarily, the synchronized product $\Theta_1 \otimes \Theta_2$ built in the previous step) smaller by abstracting it further, i.e., aggregating subsets of abstract states into block states.

Stop when only a single abstraction is left.

Synchronized Products

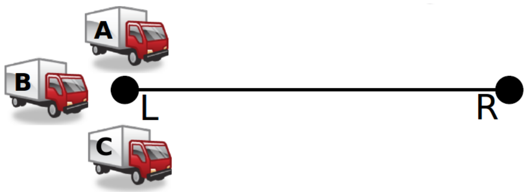
Definition (Synchronized Product of Transition Systems). *For $i \in \{1,2\}$, let $\Theta_i = (S_i, L, c, T_i, I_i, S_i^G)$ be transition systems with identical label set and cost function. The synchronized product of Θ_1 and Θ_2, written $\Theta_1 \otimes \Theta_2$, is the transition system $\Theta_\otimes = (S_\otimes, L, c, T_\otimes, I_\otimes, S_\otimes^G)$ where:*

1. $S_\otimes := S_1 \times S_2$.
2. $T_\otimes := \{((s_1, s_2), l, (t_1, t_2)) \mid (s_1, l, t_1) \in T_1$ *and* $(s_2, l, t_2) \in T_2\}$.
3. $I_\otimes := (I_1, I_2)$.
4. $S_\otimes^G := S_1^G \times S_2^G$.

→ The synchronized system can take a combined transition labeled with l if and only if both component systems can.

Synchronized Products: 3 Trucks

Atomic Projections and a Synchronized Product of: (Blackboard)



Isomorphic Transition Systems

Definition (Isomorphism). *Let $\Theta = (S, L, c, T, I, S^G)$ and $\Theta' = (S', L', c', T', I', S'^G)$ be transition systems. We say that Θ is isomorphic to Θ', written $\Theta \sim \Theta'$, if there exist bijective functions $\varphi : S \mapsto S'$ and $\psi : L \mapsto L'$ such that:*

1. $\varphi(I) = I'$.
2. $s \in S^G$ *iff* $\varphi(s) \in S'^G$.
3. $(s, l, t) \in T$ *iff* $(\varphi(s), \psi(l), \varphi(t)) \in T'$.
4. *For all* $l \in L$, $c(l) = c'(\psi(l))$.

→ Isomorphic transition systems are identical modulo renaming states and labels.

Synchronized Products of Disjoint Projections

Terminology: Two projections Θ^{V_1} and Θ^{V_2} are said to be disjoint if $V_1 \cap V_2 = \emptyset$.

Theorem (Synchronized Products of Disjoint Projections). *Let Π be an FDR planning task. If Θ^{V_1} and Θ^{V_2} are disjoint projections, then* $\Theta^{V_1} \otimes \Theta^{V_2} \sim \Theta^{V_1 \cup V_2}$. (Proof omitted.)

Corollary (Recovering Θ_Π from Atomic Projections). *Let Π be an FDR planning task with variable set V. Then* $\Theta_\Pi \sim \bigotimes_{v \in V} \Theta^{\{v\}}$.

$\rightarrow$ We can construct the concrete state space of a planning task Π through the synchronized product of its atomic projections.

Merge-and-Shrink Algorithm

Merge-and-Shrink for FDR Task Π With Variables V

(I) Initialize: $\mathcal{A} := \{\Theta^{\{v\}} \mid v \in P\}$ where $P \subseteq V$ /* Usually, $P = V$ */
while $\mathcal{A}$ contains more than one element:
  (II) Merge: select $\Theta^{\alpha_1}, \Theta^{\alpha_2}$ from $\mathcal{A}$
    $\mathcal{A} := \mathcal{A} \setminus \{\Theta^{\alpha_1}, \Theta^{\alpha_2}\} \cup \{\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}\}$
  (III) Shrink: select Θ^{α} from $\mathcal{A}$
    $\mathcal{A} := \mathcal{A} \setminus \{\Theta^{\alpha}\} \cup \{\Theta^{\alpha'}\}$ where α' is an abstraction of Θ^{α}
    [Optionally: so that *size*$(\Theta^{\alpha'}) \leq N$] /* N: Parameter */
return the remaining element Θ^{α} in $\mathcal{A}$

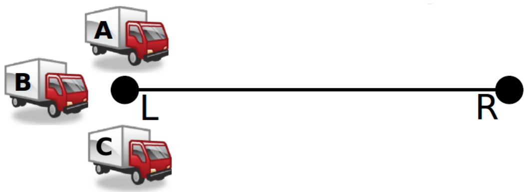
Definition. *Abstractions α constructed by this algorithm are called merge-and-shrink abstractions. The construction size of α is the maximum size of any abstraction contained in $\mathcal{A}$ sometime during α's construction.*

Note: If we apply a shrinking step to $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$ after every merge step, then construction size is bounded polynomially in N and $\|\Pi\|$.

[$\rightarrow$ There is a mathematical definition of "merge-and-shrink abstraction" [Helmert *et al.* (2014)], that I'll spare you here.]

Merge-and-Shrink Algorithm: 3 Trucks

Merge-and-Shrink Algorithm in: (Blackboard)



Questionnaire

Any Abstraction is a Merge-and-Shrink Abstraction!

Merge-and-Shrink steps

(I) Initialize $\mathcal{A} := \{\Theta^{\{v\}} \mid v \in P\}$ where $P \subseteq V$
(II) Merge Θ^{α_1} and Θ^{α_2} to $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$
(III) Shrink Θ^{α} to $\Theta^{\alpha'}$, where α' is an abstraction of Θ^{α}

Any abstraction a merge-and-shrink abstraction. Why?

→ We can construct any abstraction with merge-and-shrink. The important part is, *with which construction size*.

As Good as PDBs

Theorem (Merge-and-Shrink Can Simulate PDBs). *Let Π be an FDR planning task and P a pattern. There exists a merge-and-shrink abstraction α with $h^\alpha = h^P$ and construction size polynomial in $\|\Pi\|$ and the size of Θ^P.*

Proof.

Terminology: By the expressive power of a technique γ for computing heuristic functions in planning, we refer to the class of planning tasks Π for which γ can compute h^* in time polynomial in $\|\Pi\|$.

Corollary. *The expressive power of merge-and-shrink is at least as large as that of pattern databases.*

Word of caution:

- Specialized PDB algorithms are much faster than "merge-and-shrink without shrinking steps".
- This performance difference is "only polynomial", but that doesn't mean it's not important in practice!

Better than PDBs!

Theorem. *The expressive power of merge-and-shrink is strictly greater than that of PDBs, even for orthogonal pattern collections.*

Proof Sketch. There exist families of planning tasks in which merge-and-shrink can compute h^* in polynomial time, but PDBs, even additive ones, cannot. One example is "Logistics mal anders" when scaling the number of trucks (cf. slides 8 and 45).

→ Some competition benchmarks are such families, too! (Gripper, Schedule, Dining-Philosopers, Optical-Telegraph.)

Word of caution:

- Note the *"can"* (compute h^* in time polynomial in $\|\Pi\|$)!

  → The above assumes we are taking the right decisions: *Which abstractions to merge next, and how to shrink their product?*

  → One of the main issues in practice, see next section.

Better than *Additive* PDBs!

Theorem. *Let Π be a unit-cost FDR planning task and $\{P_1, \dots, P_k\}$ be an orthogonal pattern collection. There exists a merge-and-shrink abstraction α whose construction size is polynomial in $\|\Pi\|$ and the sizes of Θ^{P_i}, and where $\sum_{i=1}^{k} h^{P_i} \leq h^\alpha$.* (Proof omitted.)

Remarks:

- One could, in principle, derive a similar *theory of orthogonality/additivity* for merge-and-shrink abstraction as we did for pattern databases in **Chapter 12**.
- However, the above theorem shows that this is "not necessary": Orthogonality can be exploited by single merge-and-shrink abstractions.
- BUT that does not mean that summing over multiple merge-and-shrink abstractions couldn't be useful in practice! (Hasn't been tried yet → FAI BSc/MSc/HiWi)

Questionnaire

How To *Instantiate* the Merge-and-Shrink Framework?

Merge-and-Shrink Algorithm: Choice Points

(I) Initialize $\mathcal{A} := \{\Theta^{\{v\}} \mid v \in P\}$ where $P \subseteq V$
while $\mathcal{A}$ contains more than one element:
  (II) Merge: select Θ^{α_1}, Θ^{α_2} from $\mathcal{A}$
    $\mathcal{A} := \mathcal{A} \setminus \{\Theta^{\alpha_1}, \Theta^{\alpha_2}\} \cup \{\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}\}$
  (III) Shrink: select Θ^{α} from $\mathcal{A}$
    $\mathcal{A} := \mathcal{A} \setminus \{\Theta^{\alpha}\} \cup \{\Theta^{\alpha'}\}$ where α' is an abstraction of Θ^{α}
    [Optionally: so that *size*$(\Theta^{\alpha'}) \leq N$]
return the remaining element Θ^{α} in $\mathcal{A}$

- Which abstractions to select? $\rightsquigarrow$ Merging strategy.
  $\rightarrow$ Not covered in what follows. (For shrinking, all current implementations select the last synchronized product built i.e. $\Theta^{\alpha} = \Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$.)
- How to shrink an abstraction? $\rightsquigarrow$ Shrinking strategy.
  $\rightarrow$ Overview up next.
- How to choose N, if used? (We don't use it in what follows)
  $\rightarrow$ Nobody knows ... currently, if N is used, fix some value by hand.

Shrinking Strategies, Take 1: Bisimulation

Reminder: → Chapter 11

Let $\Theta = (S, L, c, T, I, S^G)$ be a transition system, and let $\alpha : S \mapsto S'$ be a surjective function. Then by $\sim^{\alpha}$ we denote the induced equivalence relation on Θ, defined by $s \sim^{\alpha} t$ iff $\alpha(s) = \alpha(t)$.
$\rightarrow \Theta/{\sim^{\alpha}}$ is isomorphic to Θ^{α}.

- We can characterize classes of abstractions by properties of their induced equivalence relations.
- Bisimulations are a particular kind of equivalence relation, *that do not incur any information loss*.
- The idea is for each shrinking step to set α' to a bisimulation of $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$.

Shrinking Strategies, Take 1: Bisimulation, ctd.

Definition (Bisimulation). *Let $\Theta = (S, L, c, T, I, S^G)$ be a transition system, and let $\alpha : S \mapsto S'$ be a surjective function. We say that α is a bisimulation of Θ if, for all $s, t \in S$, $s \sim^{\alpha} t$ implies that:*

1. *either $s, t \in S^G$ or $s, t \notin S^G$;*
2. *for all $l \in L$, $\{[s'] \mid (s, l, s') \in T\} = \{[t'] \mid (t, l, t') \in T\}$.*

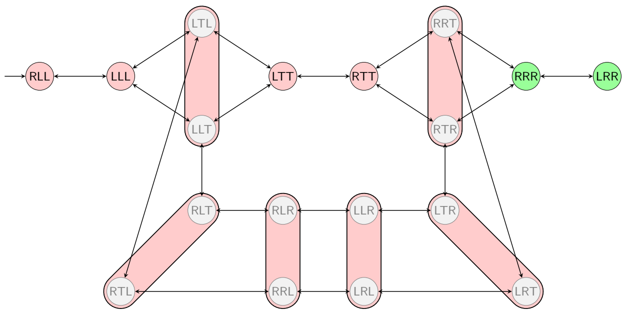
$\rightarrow$ States s, t are bisimilar iff (i) they agree on whether or not the goal is true, and (ii) every action leads into equivalent outcome states.

Theorem (Bisimulation Heuristics are Perfect). *Let Θ be a transition system, and let α be a bisimulation of Θ. Then h^{α} is perfect.*

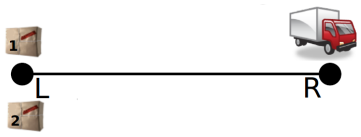
(Proof omitted.)

Bisimulation: Example Logistics

A bisimulation[1] in a Logistics variant:



State variables:
truck, pack1, pack2.



[1] For the pedantic reader: This is a bisimulation only under conservative label reduction, that I won't describe here; for the purpose of understanding the picture, just ignore the distinctions between labels.

Shrinking Strategies, Take 1: Bisimulation, ctd.

Merge-and-Shrink with Shrinking by Bisimulation

(I) Initialize $\mathcal{A} := \{\Theta^{\{v\}} \mid v \in V\}$
(II) Merge Θ^{α_1} and Θ^{α_2} to $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$
(III) Shrink Θ^{α} to $\Theta^{\alpha'}$, where α' is a bisimulation of Θ^{α}

Notation: Identify α with Θ^{α}.[2]

Lemma (Invariance over Merging Steps). *For disjoint projections Θ^{V_1} and Θ^{V_2}, if Θ^{α_1} is a bisimulation for Θ^{V_1} and Θ^{α_2} is a bisimulation for Θ^{V_2}, then $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$ is a bisimulation for $\Theta^{V_1 \cup V_2}$.* (Proof omitted.)

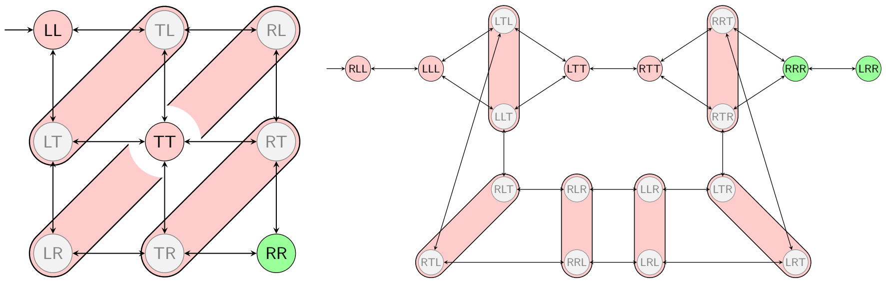
Theorem (Shrinking by Bisimulation is Perfect). *If α is a merge-and-shrink abstraction constructed using shrinking by bisimulation, then $h^{\alpha} = h^*$.*

Proof. Show by induction: Every Θ^{α} in α is a bisimulation for $\Theta^{V'}$ where $V' \subseteq V$ are the variables merged into Θ^{α}. Base case and invariance over shrinking steps are trivial, invariance over merging steps by above Lemma. Thus the returned Θ^{α} is a bisimulation for $\Theta^{V} = \Theta_{\Pi}$. Done with slide 27 Theorem.

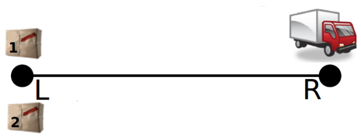
[2] I have so far glossed over the role of the abstraction mapping α. See Section **M&S Abstraction Mappings**.

Bisimulation: Example Logistics, ctd.

(a) Bisimulation[3] of $\Theta^{\{\text{pack1},\text{pack2}\}}$, and (b) its product with $\Theta^{\{\text{truck}\}}$:



State variables:
truck, pack1, pack2.



[3] For the pedantic reader: This is a bisimulation only under conservative label reduction, that I won't describe here; for the purpose of understanding the picture, just ignore the distinctions between labels.

Computing Perfect Heuristics??

The good news: There are significant families of planning tasks where *merge-and-shrink with shrinking by bisimulation has polynomial runtime.*[4]

→ Some competition benchmarks are such families! In Gripper and Schedule, Merge-and-shrink with shrinking by bisimulation computes the perfect heuristic in polynomial time.

→ This provides a partial solution to the issue posed on slide 22 (how to realize the expressive power in practice, efficiently computing h^* where merge-and-shrink in principle has the ability to do that).

The bad news: Typically, merge-and-shrink with shrinking by bisimulation has exponential runtime. Practical performance is poor except in very few domains.

[4] Provided we use conservative label reduction, that I won't describe here.

How to Relax a Bisimulation?

Bisimulation is too ambitious: Of course we can't typically compute h^* effectively. So we need to "approximate more":

(a) Impose a size bound N and "stop splitting non-bisimilar states when that size is reached".

(b) Define a less restrictive (relaxed) class of equivalence relations.

→ Both (a) [Dräger *et al.* (2006); Nissim *et al.* (2011)] and (b) [Katz *et al.* (2012)] have been done. We briefly consider (b).

Definition (K-Catching Bisimulation). *Let $\Theta = (S, L, c, T, I, S^G)$ be a transition system, let $K \subseteq L$, and let $\alpha : S \mapsto S'$ be a surjective function. We say that α is a K-catching bisimulation of Θ if, for all $s, t \in S$, $s \sim^\alpha t$ implies that:*

(i) *either $s, t \in S^G$ or $s, t \notin S^G$;*

(ii) *for all $l \in K$, $\{[s'] \mid (s, l, s') \in T\} = \{[t'] \mid (t, l, t') \in T\}$.*

→ Instead of accurately reflecting *all* actions, we reflect only a subset K.

Shrinking Strategies, Take 2: K-Catching Bisimulation

Merge-and-Shrink with Shrinking by K-Catching Bisimulation

Select a subset of actions K

(I) Initialize $\mathcal{A} := \{\Theta^{\{v\}} \mid v \in V\}$

(II) Merge Θ^{α_1} and Θ^{α_2} to $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$

(III) Shrink Θ^{α} to $\Theta^{\alpha'}$, where α' is a K-catching bisimulation of Θ^{α}

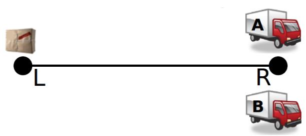
Lemma (K-Catching Bisimulation is Invariant over Merging Steps). *For disjoint projections Θ^{V_1} and Θ^{V_2}, if Θ^{α_1} is a K-catching bisimulation for Θ^{V_1} and Θ^{α_2} is a K-catching bisimulation for Θ^{V_2}, then $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$ is a K-catching bisimulation for $\Theta^{V_1 \cup V_2}$.* (Proof omitted.)

$\Longrightarrow$ (*) The abstraction returned is a K-catching bisimulation of Θ_Π.

- Selection of K controls accuracy/overhead trade-off.
  $K = L \Longrightarrow$
  $K = \emptyset \Longrightarrow$
- With (*), *guarantees* on quality of h^α can be ensured by selecting appropriate K. Two significant concepts exist [Katz *et al.* (2012)].

Our Example: "Logistics mal anders"

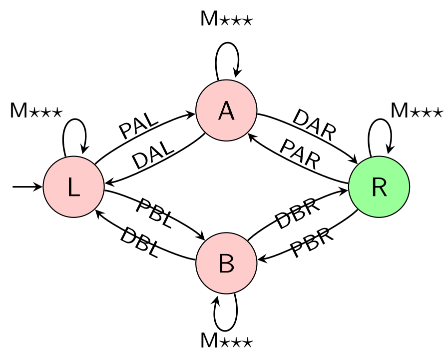
We zoom in on synchronized products in "Logistics mal anders":



- Hence we now look at the transition systems for atomic projections of this example, including the transition labels as they are important for synchronized products.
- We abbreviate action 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 transitions (same start, end vertices) 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

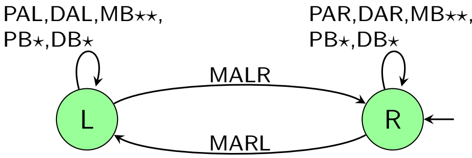
Atomic Projections

$\Theta^{\{\text{package}\}}$:



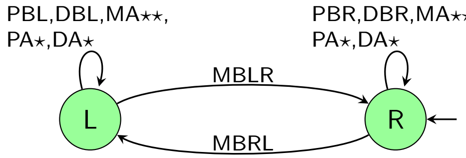
Atomic Projections

$\Theta^{\{\text{truck A}\}}$:



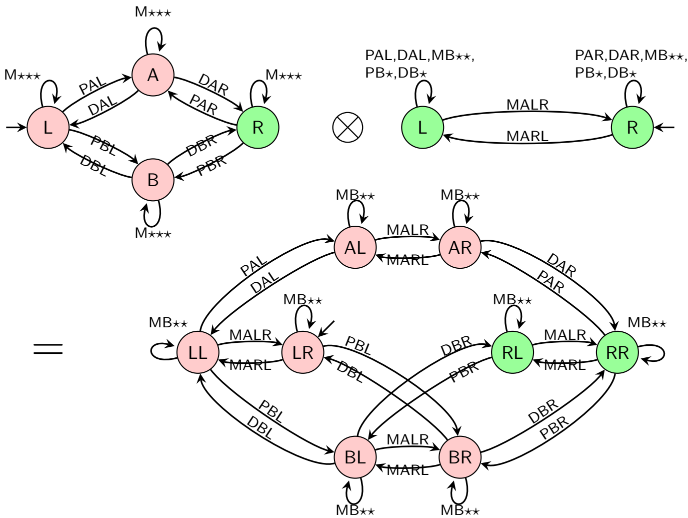
Atomic Projections

$\Theta^{\{\text{truck B}\}}$:



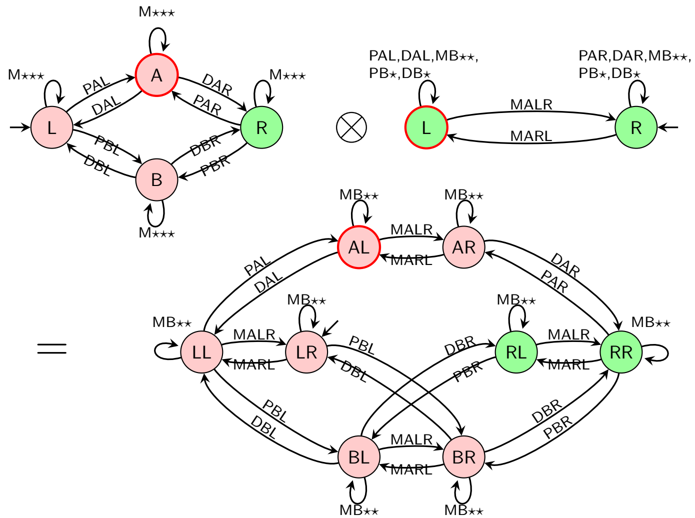
Synchronized Product Computation

$\Theta^{\{\text{package}\}} \otimes \Theta^{\{\text{truck A}\}}$:



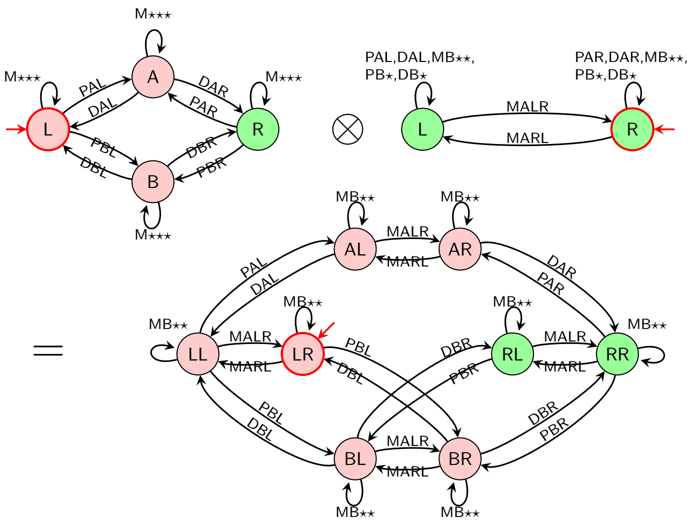
Synchronized Product Computation

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



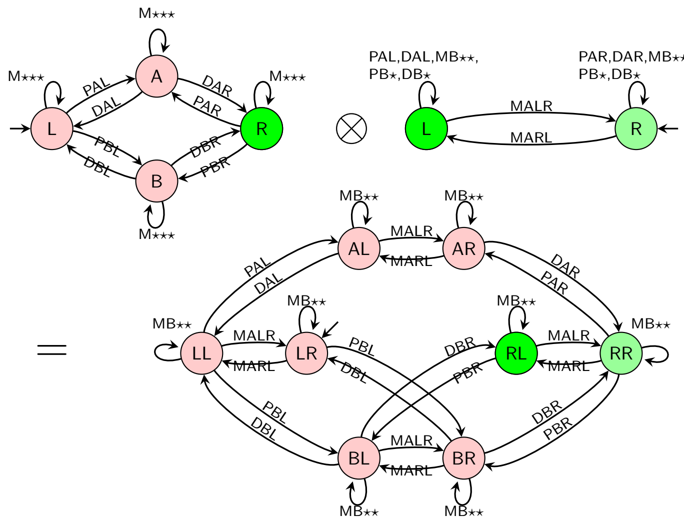
Synchronized Product Computation

$\Theta^{\{\text{package}\}} \otimes \Theta^{\{\text{truck A}\}}$: $s_{0\otimes} = (s_{01}, s_{02})$



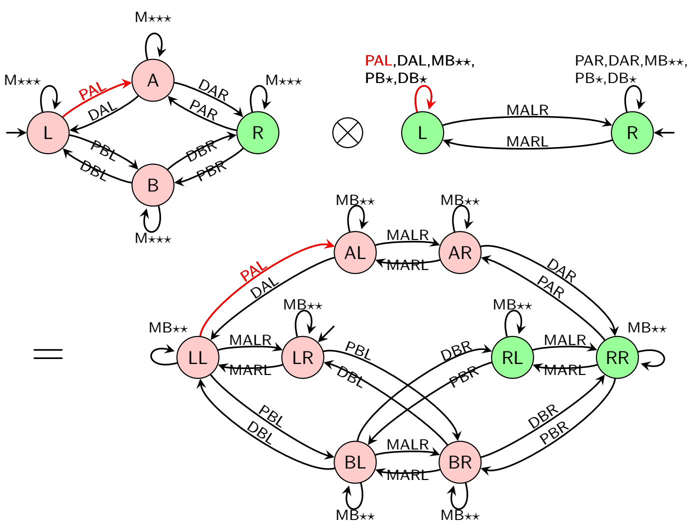
Synchronized Product Computation

$\Theta^{\{\text{package}\}} \otimes \Theta^{\{\text{truck A}\}}$: $S^G{}_\otimes = S^G{}_1 \times S^G{}_2$



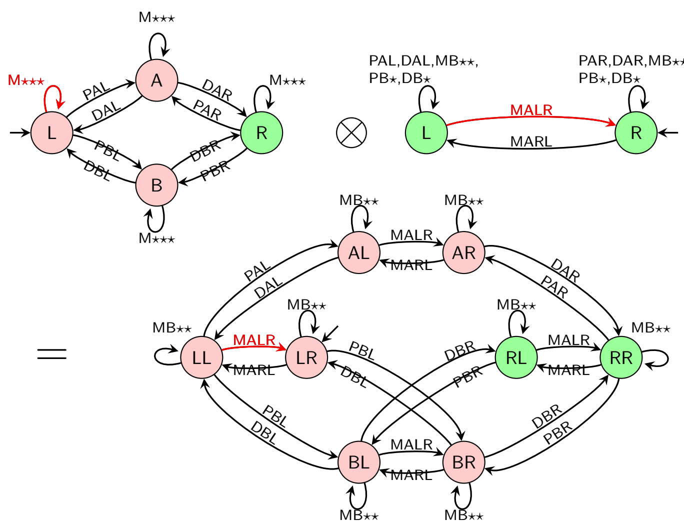
Synchronized Product Computation

$\Theta^{\{\text{package}\}} \otimes \Theta^{\{\text{truck A}\}}$: $T_\otimes := \{((s_1, s_2), l, (t_1, t_2)) \mid \dots\}$



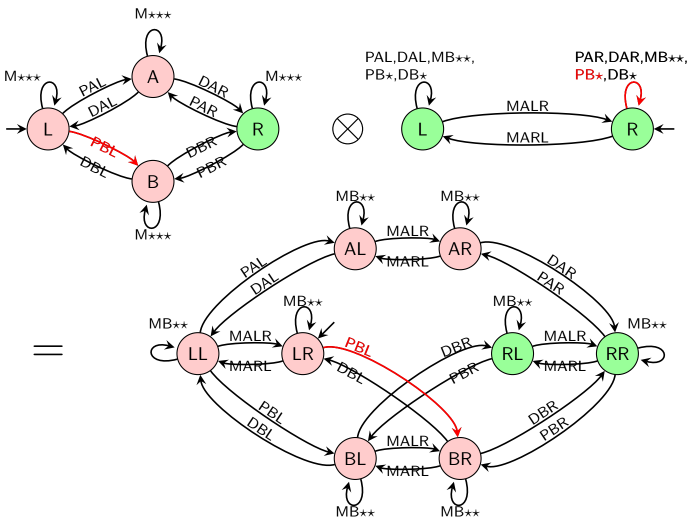
Synchronized Product Computation

$\Theta^{\{\text{package}\}} \otimes \Theta^{\{\text{truck A}\}}$: $T_\otimes := \{((s_1, s_2), l, (t_1, t_2)) \mid \dots\}$



Synchronized Product Computation

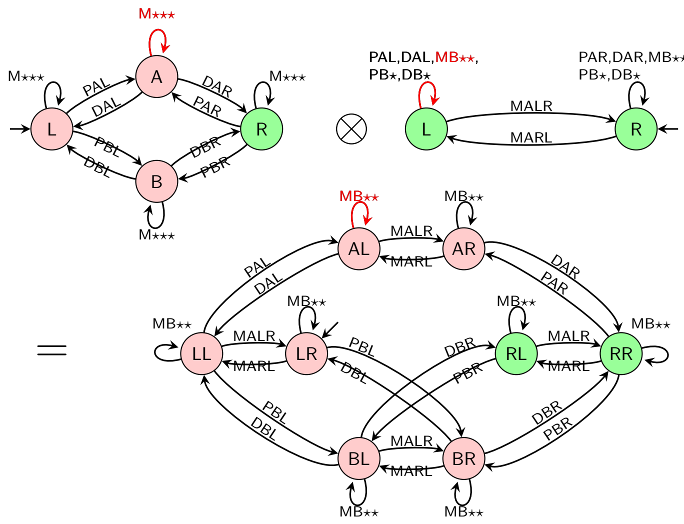
$\Theta^{\{\text{package}\}} \otimes \Theta^{\{\text{truck A}\}}$: $T_\otimes := \{((s_1, s_2), l, (t_1, t_2)) \mid \dots\}$



Introduction M&S Framework Power M&S Strategies A Full Example Mappings Conclusion References

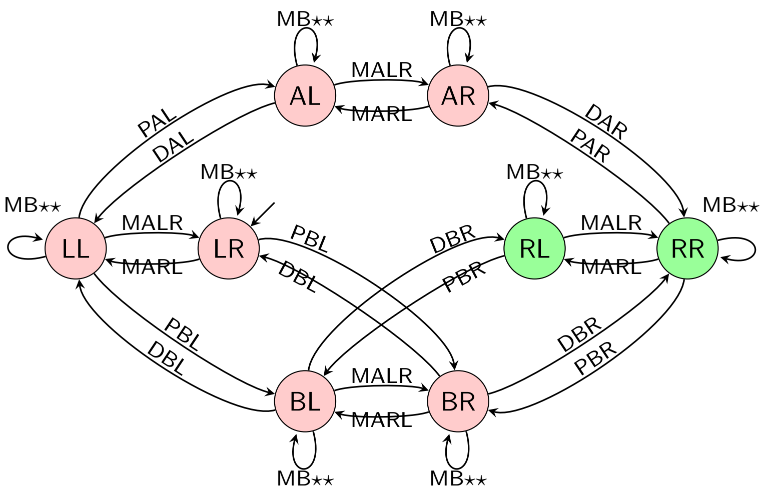
Synchronized Product Computation

$\Theta^{\{\text{package}\}} \otimes \Theta^{\{\text{truck A}\}}$: $T_\otimes := \{((s_1, s_2), l, (t_1, t_2)) \mid \dots\}$



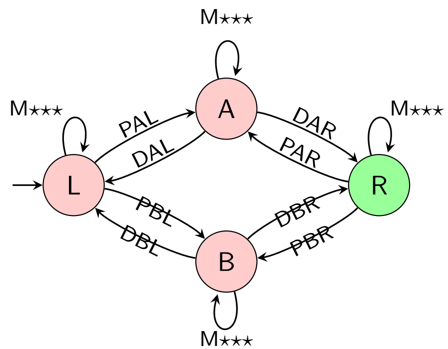
Disjoint Synchronized Products

$\Theta^{\{\text{package}\}} \otimes \Theta^{\{\text{truck A}\}} \sim \Theta^{\{\text{package,truck A}\}}$:



Step (I) Initialize

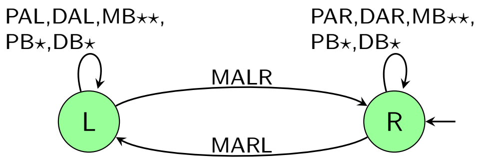
$\Theta^{\{\text{package}\}}$:



Current collection $\mathcal{A}$: $\{\Theta^{\{\text{package}\}}, \Theta^{\{\text{truck A}\}}, \Theta^{\{\text{truck B}\}}\}$

Step (I) Initialize

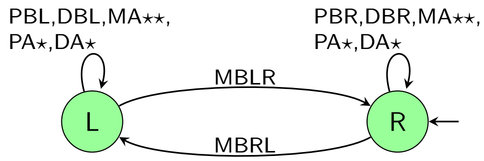
$\Theta^{\{\text{truck A}\}}$:



Current collection $\mathcal{A}$: $\{\Theta^{\{\text{package}\}}, \Theta^{\{\text{truck A}\}}, \Theta^{\{\text{truck B}\}}\}$

Step (I) Initialize

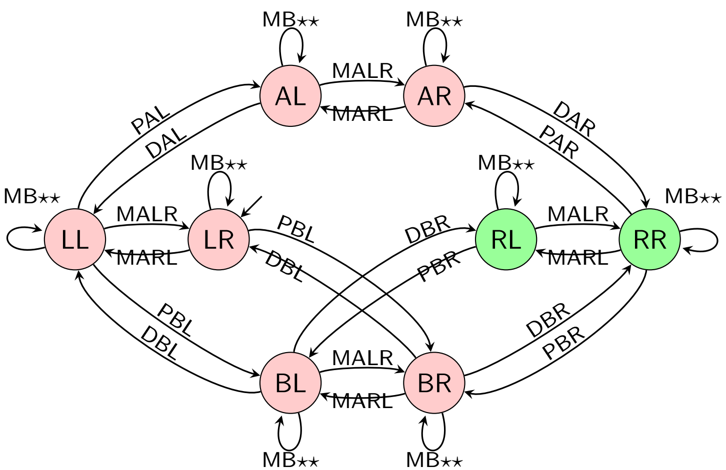
$\Theta^{\{\text{truck B}\}}$:



Current collection $\mathcal{A}$: $\{\Theta^{\{\text{package}\}}, \Theta^{\{\text{truck A}\}}, \Theta^{\{\text{truck B}\}}\}$

Step (II) Merge

$\Theta_1 := \Theta^{\{\text{package}\}} \otimes \Theta^{\{\text{truck A}\}}$:



Current collection $\mathcal{A}$: $\{\Theta_1, \Theta^{\{\text{truck B}\}}\}$

To Shrink Or Not to Shrink?

Shrinking is needed only to keep abstraction size at bay:

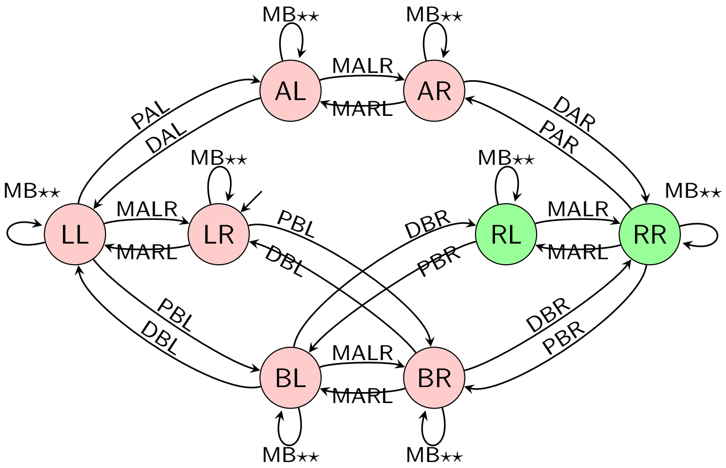
- If we have sufficient memory available, we can now compute $\Theta_1 \otimes \Theta^{\{\text{truck B}\}}$, which would recover the complete transition system of the task, and thus yield the perfect heuristic $h^\alpha = h^*$.
- However, to illustrate the general idea, let us assume that we do not have sufficient memory for this product.
- Specifically, we assume that $N = 4$, i.e., after each merge step we need to replace the product abstraction with a coarsening that has only four abstract states.

$\rightarrow$ So, in the present example, we need to reduce Θ_1 to four states.

$\rightarrow$ Lots of freedom in *how exactly* to abstract Θ_1. Here, we use a simple abstraction whose outcome heuristic is Ok, and that's easy to illustrate.

Step (III) Shrink

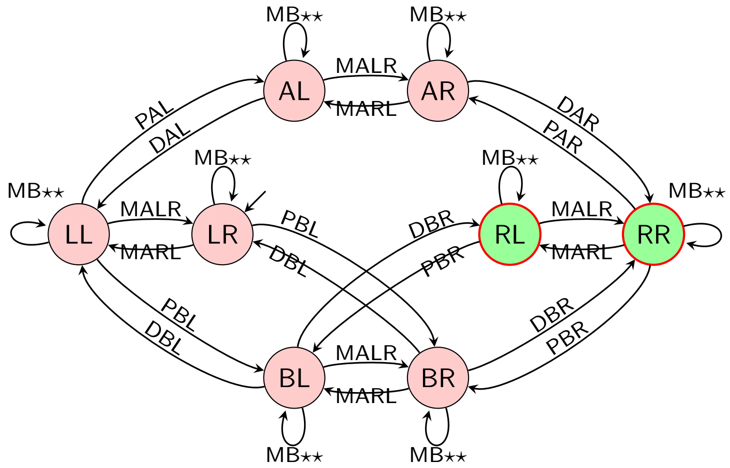
$\Theta_2 :=$ some abstraction of Θ_1, obtained by aggregating states:



Introduction M&S Framework Power M&S Strategies A Full Example Mappings Conclusion References

Step (III) Shrink

$\Theta_2 :=$ some abstraction of Θ_1, obtained by aggregating states:

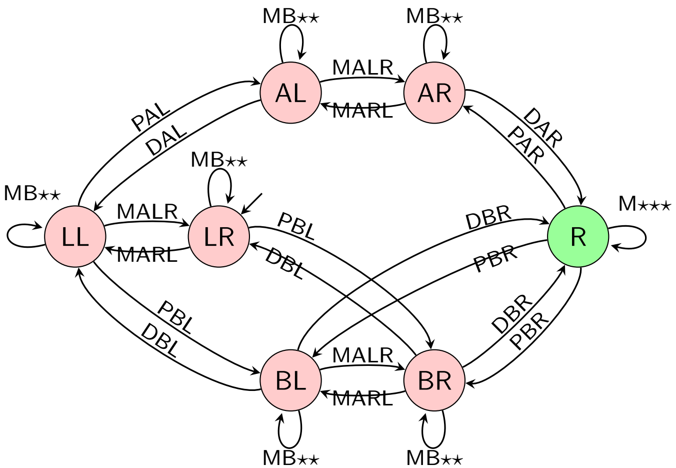


Álvaro Torralba, Cosmina Croitoru AI Planning Chapter 13: Merge-and-Shrink Heuristics 43/63

Introduction M&S Framework Power M&S Strategies A Full Example Mappings Conclusion References

Step (III) Shrink

$\Theta_2 :=$ some abstraction of Θ_1, obtained by aggregating states:

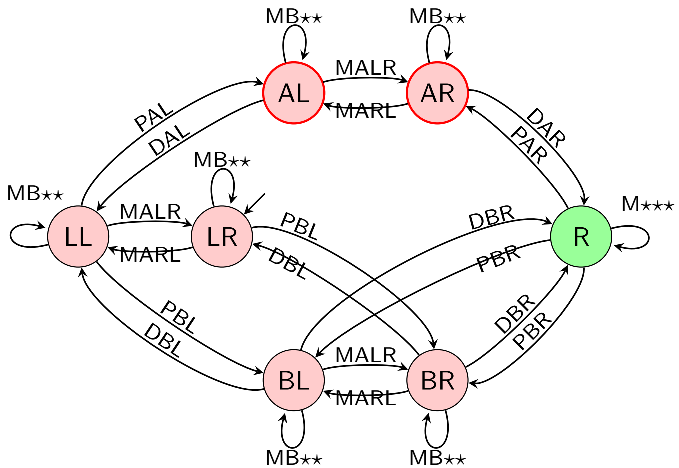


Álvaro Torralba, Cosmina Croitoru AI Planning Chapter 13: Merge-and-Shrink Heuristics 43/63

Introduction M&S Framework Power M&S Strategies A Full Example Mappings Conclusion References

Step (III) Shrink

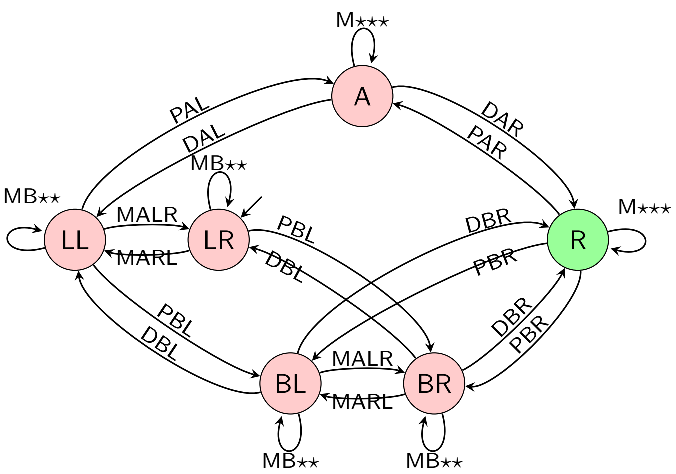
$\Theta_2 :=$ some abstraction of Θ_1, obtained by aggregating states:



Álvaro Torralba, Cosmina Croitoru AI Planning Chapter 13: Merge-and-Shrink Heuristics 43/63

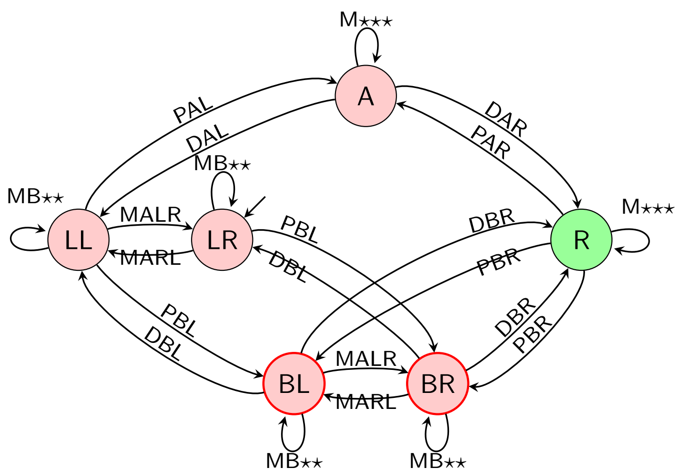
Step (III) Shrink

Θ_2 := some abstraction of Θ_1, obtained by aggregating states:



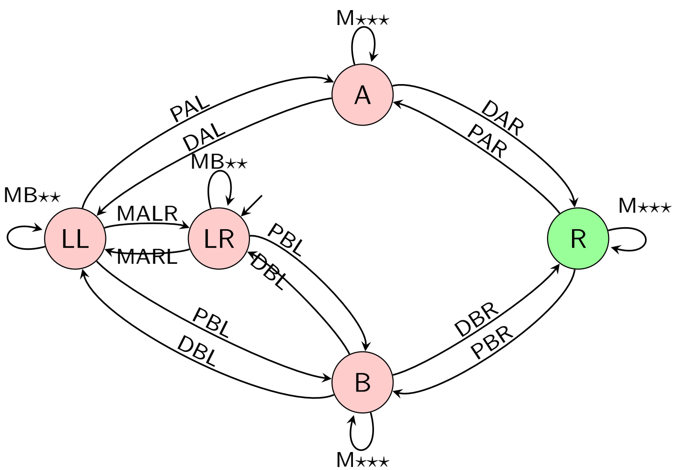
Step (III) Shrink

Θ_2 := some abstraction of Θ_1, obtained by aggregating states:



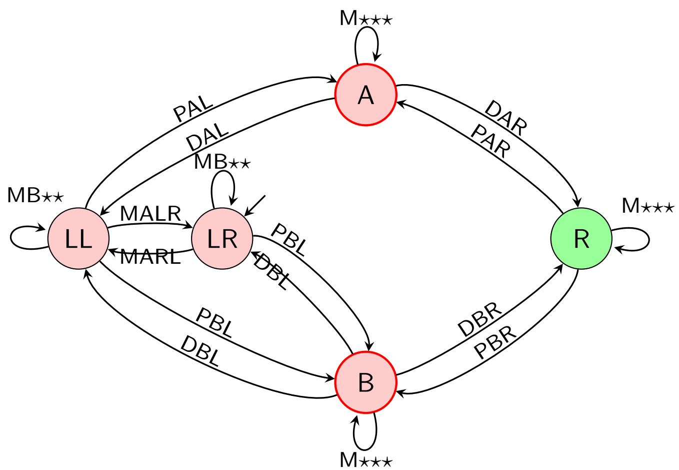
Step (III) Shrink

Θ_2 := some abstraction of Θ_1, obtained by aggregating states:



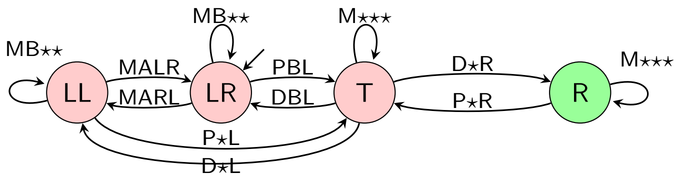
Step (III) Shrink

Θ_2 := some abstraction of Θ_1, obtained by aggregating states:



Step (III) Shrink

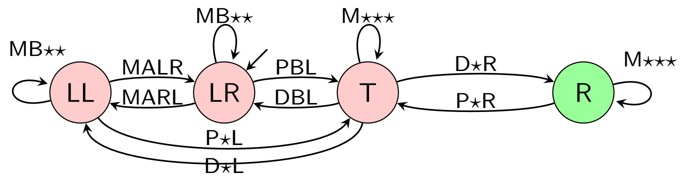
Θ_2 := some abstraction of Θ_1, obtained by aggregating states:



Introduction M&S Framework Power M&S Strategies A Full Example Mappings Conclusion References

Step (III) Shrink

Θ_2 := some abstraction of Θ_1, obtained by aggregating states:



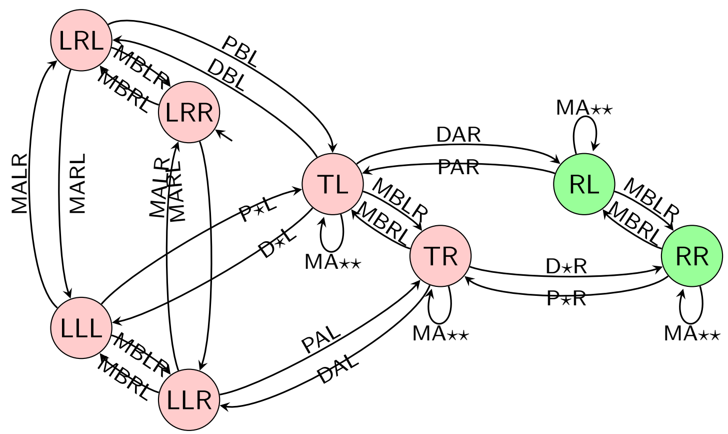
Current collection $\mathcal{A}$: $\{\Theta_2, \Theta^{\{\text{truck B}\}}\}$

Álvaro Torralba, Cosmina Croitoru AI Planning Chapter 13: Merge-and-Shrink Heuristics 43/63

Introduction M&S Framework Power M&S Strategies A Full Example Mappings Conclusion References

Step (II) Merge

$\Theta_3 := \Theta_2 \otimes \Theta^{\{\text{truck B}\}}$:



Current collection $\mathcal{A}$: $\{\Theta_3\}$

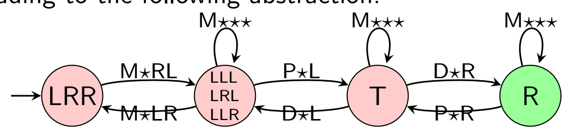
Introduction M&S Framework Power M&S Strategies A Full Example Mappings Conclusion References

Shrink Some More?

Assume we skip the next shrinking step, and use remaining cost in the final abstraction Θ_3 as our heuristic function.

$\rightarrow$ What is the value of $h^\alpha(I)$?

Assume we do not skip the shrinking step, and instead shrink Θ_3 down to size $N = 4$, e.g. leading to the following abstraction:



$\rightarrow$ Here we get $h^\alpha(I) =$

Remark: We can get $h^\alpha(I) = h^*$ with polynomial construction size (but $N > 4$). Intuition: In each shrinking step, we can aggregate all states that have the same number of trucks on either side; and that agree on *which* truck (if any) the package is in, and on which side that truck is.

Abstraction Mappings vs. Abstract State Spaces

Abstraction mappings? Recall: α vs. Θ^α

- I have so far completely glossed over everything relating to abstraction mappings.
- We need to maintain the mappings α along with the abstract state spaces Θ^α.
- We need to prove that the Θ^α we compute using the synchronized product is indeed the abstract state space of the mapping α it is associated with.

  $\rightarrow$ Without this property, we could not "identify α with Θ^α" (compare slide 28).
- We need to efficiently represent and maintain α.

$\rightarrow$ See [Helmert *et al.* (2014)] for full details. In what follows, we detail the last point, how to efficiently represent and maintain α.

How do we actually represent α?

By cascading tables. **During merge-and-shrink:**

(i) For atomic abstraction $\pi_{\{v\}}$, generate a one-dimensional table mapping values in D_v to states in $\Theta^{\pi_{\{v\}}}$.

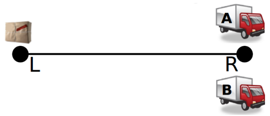
(ii) For merge step $\mathcal{A} := \mathcal{A}_1 \otimes \mathcal{A}_2$, generate a two-dimensional table mapping pairs of states in $\mathcal{A}_1$ and $\mathcal{A}_2$ to states in $\mathcal{A}$.

(iii) For shrink steps, make sure to keep the table in sync with the abstraction choices.

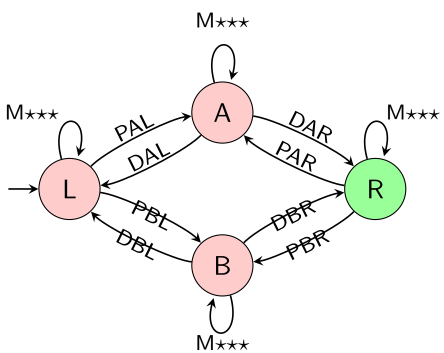
After merge-and-shrink has stopped with (Θ^α, α):

- Compute all remaining costs in Θ^α and store them in a one-dimensional table.
- Throw away all the abstractions – just keep the tables.
- To compute $h^\alpha(s)$ for any state s during search, do a sequence of table look-ups from the atomic abstractions down to the final abstraction state and heuristic value.

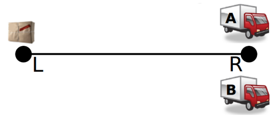
"Logistics mal anders": Atomic Abstractions



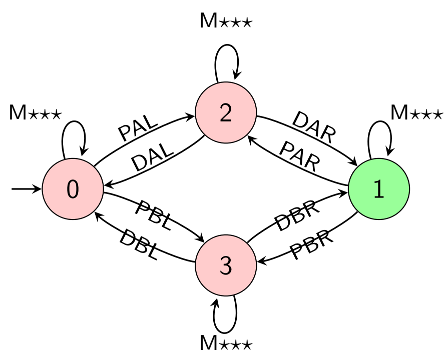
For $\Theta^{\pi_{\{v\}}}$, number the states (domain values) consecutively, and generate a table of references to the states:



"Logistics mal anders": Atomic Abstractions

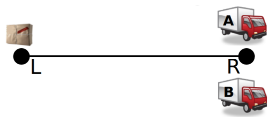


For $\Theta^{\pi_{\{v\}}}$, number the states (domain values) consecutively, and generate a table of references to the states:

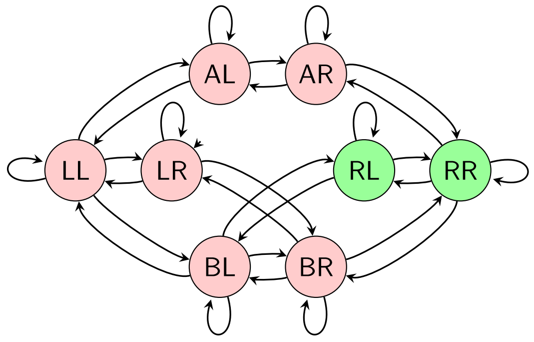


L	R	A	B
0	1	2	3

"Logistics mal anders": Merge Step

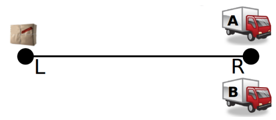


For $\mathcal{A}_1 \otimes \mathcal{A}_2$, number the product states consecutively and generate a table mapping pairs of states in $\mathcal{A}_1$ and $\mathcal{A}_2$ to states in $\mathcal{A}$:

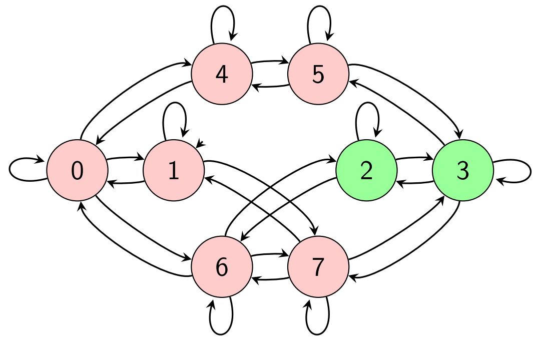


Introduction M&S Framework Power M&S Strategies A Full Example Mappings Conclusion References

"Logistics mal anders": Merge Step



For $\mathcal{A}_1 \otimes \mathcal{A}_2$, number the product states consecutively and generate a table mapping pairs of states in $\mathcal{A}_1$ and $\mathcal{A}_2$ to states in $\mathcal{A}$:



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

Introduction M&S Framework Power M&S Strategies A Full Example Mappings Conclusion References

Maintenance over Shrinking Steps

→ *How to "make sure to keep the table in sync with the abstraction choices"?*

Naïve solution:

- When aggregating 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, that map to j; change these to i.

→ Every time we aggregate two states, we must make a pass over the entire table, i.e., over the whole abstract state space :-(

→ We can do the same operation in constant time per state aggregation, using *splicing*.

Introduction M&S Framework Power M&S Strategies A Full Example Mappings Conclusion References

Effective Maintenance over Shrinking Steps: Splicing

Before the shrink operation:

- Associate each abstract state with a linked list, representing all table entries that map to this state.
- Initialize the lists by one pass over the table. Then discard the table.

During the shrink operation:

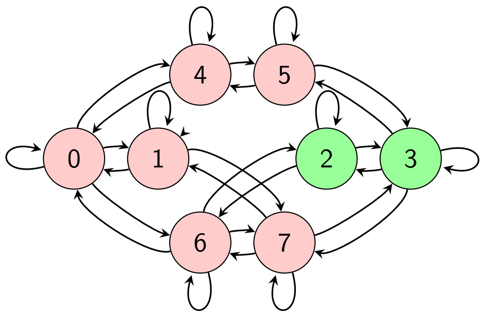
- When aggregating i and j, splice the list elements of j into the list elements of i.
  → That is, move the elements of list j to the end of list i.
  → For linked lists, this takes how much runtime?

After the shrink operation:

- Renumber the abstract states so there are no gaps in the numbering.
- Regenerate the mapping table from the linked list information.

"Logistics mal anders": Splicing

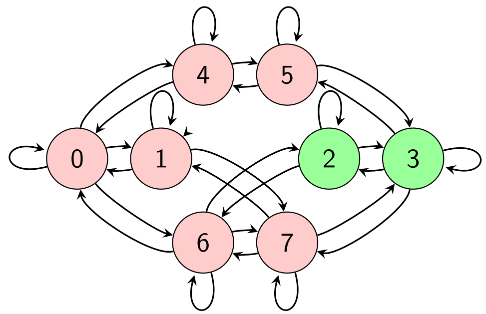
Representation before shrinking:



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

"Logistics mal anders": Splicing

1. Convert table to linked lists and discard it.

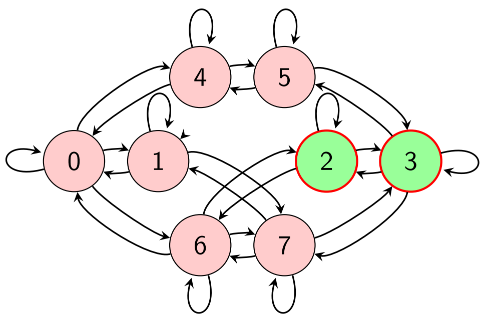


$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0)\}$
$list_3 = \{(1,1)\}$
$list_4 = \{(2,0)\}$
$list_5 = \{(2,1)\}$
$list_6 = \{(3,0)\}$
$list_7 = \{(3,1)\}$

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

"Logistics mal anders": Splicing

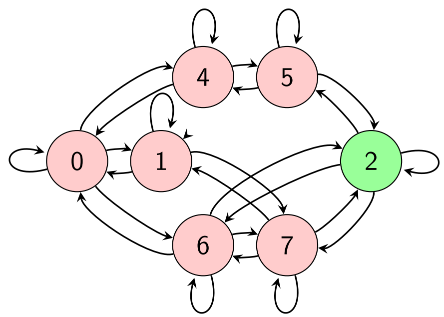
2. When combining i and j, splice list$_j$ into list$_i$.



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

"Logistics mal anders": Splicing

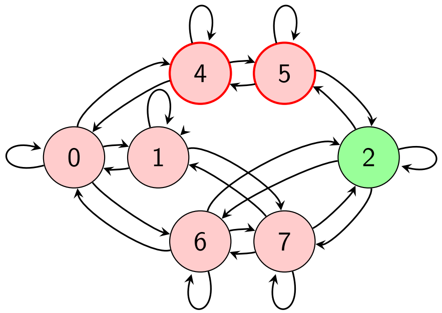
2. When combining i and j, splice list$_j$ into list$_i$.



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

"Logistics mal anders": Splicing

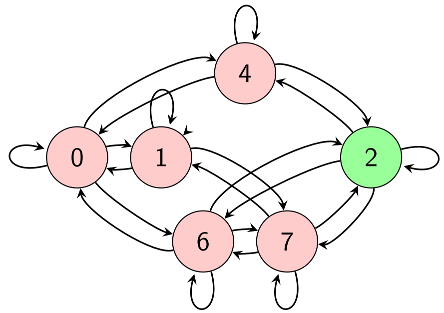
2. When combining i and j, splice list$_j$ into list$_i$.



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

"Logistics mal anders": Splicing

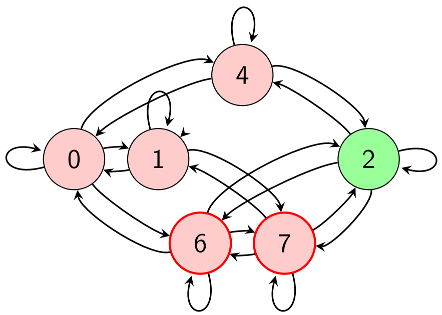
2. When combining i and j, splice list$_j$ into list$_i$.



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

"Logistics mal anders": Splicing

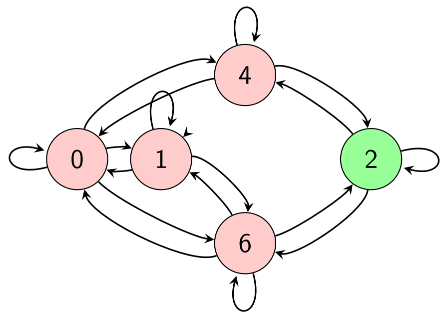
2. When combining i and j, splice list$_j$ into list$_i$.



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

"Logistics mal anders": Splicing

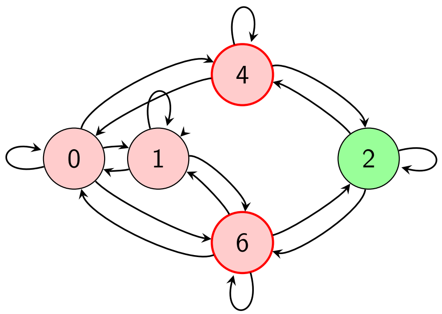
2. When combining i and j, splice list$_j$ into list$_i$.



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

“Logistics mal anders”: Splicing

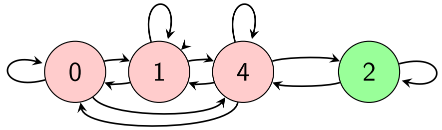
2. When combining i and j, splice list$_j$ into list$_i$.



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

“Logistics mal anders”: Splicing

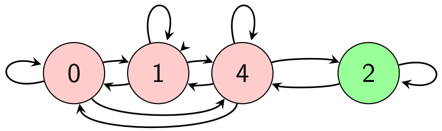
2. When combining i and j, splice list$_j$ into list$_i$.



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

“Logistics mal anders”: Splicing

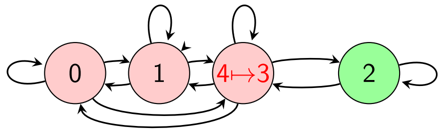
2. When combining i and j, splice list$_j$ into list$_i$.



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

“Logistics mal anders”: Splicing

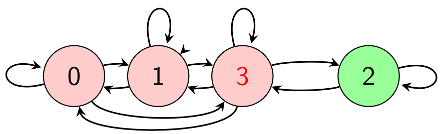
3. Renumber abstract states consecutively.



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

“Logistics mal anders”: Splicing

3. Renumber abstract states consecutively.

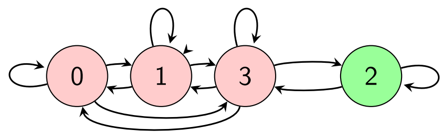


$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0),(1,1)\}$
$list_3 = \{(2,0),(2,1),(3,0),(3,1)\}$
$list_4 = \emptyset$
$list_5 = \emptyset$
$list_6 = \emptyset$
$list_7 = \emptyset$

Introduction M&S Framework Power M&S Strategies A Full Example Mappings Conclusion References

“Logistics mal anders”: Splicing

4. Regenerate the mapping table from the linked lists.



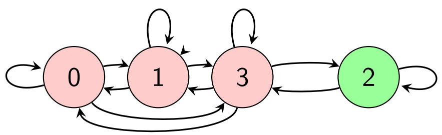
$list_0 = \{(0,0)\}$
$list_1 = \{(0,1)\}$
$list_2 = \{(1,0),(1,1)\}$
$list_3 = \{(2,0),(2,1),(3,0),(3,1)\}$
$list_4 = \emptyset$
$list_5 = \emptyset$
$list_6 = \emptyset$
$list_7 = \emptyset$

Álvaro Torralba, Cosmina Croitoru AI Planning Chapter 13: Merge-and-Shrink Heuristics 53/63

Introduction M&S Framework Power M&S Strategies A Full Example Mappings Conclusion References

“Logistics mal anders”: Splicing

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

Álvaro Torralba, Cosmina Croitoru AI Planning Chapter 13: Merge-and-Shrink Heuristics 53/63

Introduction M&S Framework Power M&S Strategies A Full Example Mappings Conclusion References

“Logistics mal anders”: The Cascading Tables

For the example from the A Full Example Section:

- Three one-dimensional tables for the atomic abstractions:

T_{package}	L	R	A	B
	0	1	2	3

T_{truckA}	L	R
	0	1

T_{truckB}	L	R
	0	1

- Two tables for the intermediate abstractions Θ_1 and Θ_2:

T_1	$s_2 = 0$	$s_2 = 1$
$s_1 = 0$	0	1
$s_1 = 1$	2	2
$s_1 = 2$	3	3
$s_1 = 3$	3	3

T_2	$s_2 = 0$	$s_2 = 1$
$s_1 = 0$	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 Θ_3:

T_3	$s = 0$	$s = 1$	$s = 2$	$s = 3$
$h(s)$	3	2	1	0

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

- $h(s) = T_3[T_2[T_1[T_{\text{package}}[p], T_{\text{truckA}}[a]], T_{\text{truckB}}[b]]]$

Summary

- PDBs are limited because they are based on projections, a very restrictive class of abstractions.
- Merge-and-shrink can construct *any* abstraction, following the construction of the state space from atomic projections using their synchronized product, $\Theta_\Pi \sim \bigotimes_{v \in V} \Theta^{\{v\}}$: Starting from the atomic projections, merge steps build the synchronized product of two abstractions, and shrink steps replace an abstraction with a coarsening of itself.
- Merge-and-shrink *can*, in principle, polynomially simulate PDBs, and can efficiently compute h^* in some cases where PDBs cannot.
- To run merge-and-shrink in practice, we need a merging strategy and a shrinking strategy.
- Using bisimulation in the shrinking strategy, merge-and-shrink computes h^*. This is polynomial-time in some IPC benchmarks, but infeasible in most. More approximate shrinking strategies are an active area of research.

Remarks

- Merge-and-shrink abstraction has its origin in work applying heuristic search to the falsification of safety properties (i.e., finding error states) in model checking [Dräger *et al.* (2006)]. In difference to planning, the state space there is *defined* as the synchronized product of atomic state machines.
- The technique was adapted to planning one year later [Helmert *et al.* (2007)], also pointing out its relation to PDBs.
- Recent work [Nissim *et al.* (2011); Katz *et al.* (2012)] concentrated on meaningful ways to define shrinking strategies (see also next slides).
- The 2011 version of merge-and-shrink participated in IPC'11. It won a 2nd prize in the track for optimal planners, and was part of the portfolio winning the 1st prize.
- Merge-and-shrink heuristics are generally very competitive with other admissible heuristics for planning; in some domains, they are the best ones we currently have.

Reading

- *Flexible Abstraction Heuristics for Optimal Sequential Planning* [Helmert *et al.* (2007)].

  Available at:

  `http://fai.cs.uni-saarland.de/hoffmann/papers/icaps07.pdf`

  Content: The first paper on merge-and-shrink in planning. Introduces the basic framework, in particular observing that $\Theta_\Pi \sim \bigotimes_{v \in V} \Theta^{\{v\}}$. Briefly states the theoretical results compared to pattern databases. Devises first simple merging and shrinking strategies (not covered here), and runs experiments in some IPC benchmarks of the time, concluding that PDBs are often outperformed.

Reading, ctd.

- *Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstraction in Optimal Planning* [Nissim *et al.* (2011)].

  Available at:

  `http://fai.cs.uni-saarland.de/hoffmann/papers/ijcai11.pdf`

  Content: Introduces the bisimulation-based shrinking strategy (bisimulation itself is a well-known concept and has for a very long time been used in other areas of in computer science). Observes its theoretical properties, devises some simple approximations, and runs experiments showing that it outperforms state-of-the-art optimal planners (in particular heuristic search with LM-cut, cf. → **Chapter 17**) in some domains.

Reading, ctd.

- *Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces* [Helmert *et al.* (2014)].

  Available at:

  http://fai.cs.uni-saarland.de/hoffmann/papers/jacm14.pdf

  Content: Subsumes the two previous papers. Treats the subject in full detail, giving lots of additional theoretical analysis, and comprehensive experiments.

Reading, ctd.

- *How to Relax a Bisimulation?* [Katz *et al.* (2012)].

  Available at:

  http://fai.cs.uni-saarland.de/hoffmann/papers/icaps12b.pdf

  Content: Introduces K-catching bisimulation and examines its properties. Devises two methods for selecting K in a way that guarantees A^* will not have to perform any search. Those label sets K cannot actually be found efficiently, so approximations are devised and tested in benchmarks. Experiments outcome basically shows that not much is gained. It is an open question whether that can be fixed by better approximations of K.

References I

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

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

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

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

References II

Raz Nissim, Jörg Hoffmann, and Malte Helmert. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 1983–1990. AAAI Press/IJCAI, 2011.