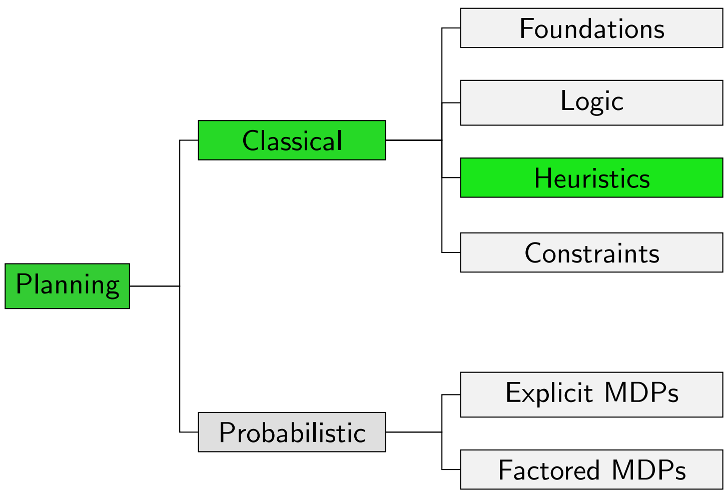
Planning and Optimization

D8. Merge-and-Shrink: Algorithm and Heuristic Properties

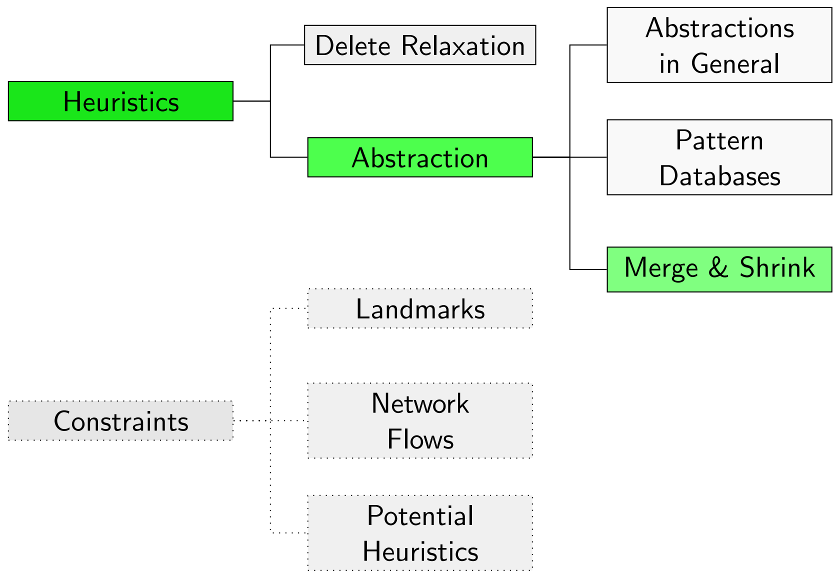
Malte Helmert and Gabriele Röger

Universität Basel

Content of this Course



Content of this Course: Heuristics



D8.1 Generic Algorithm

Generic Merge-and-shrink Abstractions: Outline

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

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

The final factor is then used for an abstraction heuristic.

Generic Algorithm Template

Generic Merge & Shrink Algorithm for planning task Π

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

Merge-and-Shrink Strategies

Choices to resolve to instantiate the template:

- When to merge, when to shrink?
  ⇝ general strategy
- Which abstractions to merge?
  ⇝ merging strategy
- Which abstraction to shrink, and how to shrink it (which β)?
  ⇝ shrinking strategy

Choosing a Strategy

There are many possible ways to resolve these choices, and we do not cover them in detail.

A typical general strategy:

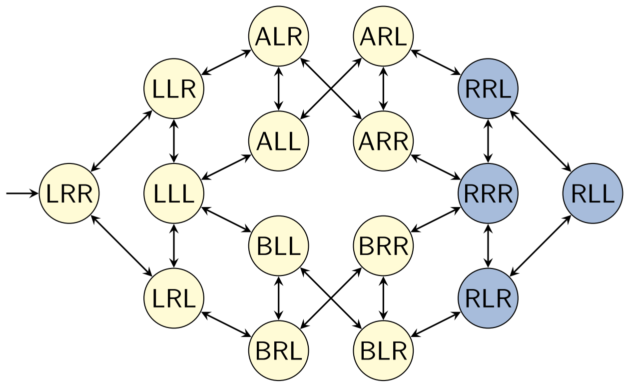
- define a limit N on the number of states allowed in each factor
- in each iteration, select two factors we would like to merge
- merge them if this does not exhaust the state number limit
- otherwise shrink one or both factors just enough to make a subsequent merge possible

Abstraction Mappings

- The pseudo-code as described only returns the final abstract transition system $\mathcal{T}^\alpha$.
- In practice, we also need the abstraction mapping α, so that we can map concrete states to abstract states when we need to evaluate heuristic values.
- We do not describe in detail how this can be done.
  - Key idea: keep track of which factors are merged, which factors are shrunk and how.
  - "Replay" these decisions to map a given concrete state s to the abstract state $\alpha(s)$.
- The run-time for such a heuristic look-up is $O(|V|)$ for a task with state variables V.

D8.2 Example

Back to the Running Example

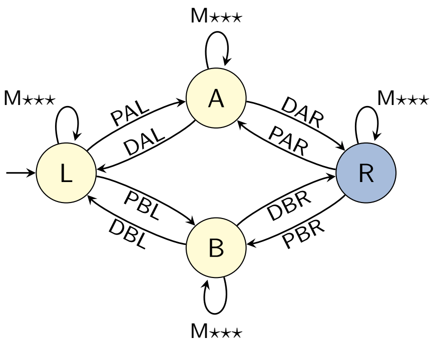


Logistics problem with one package, two trucks, two locations:

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

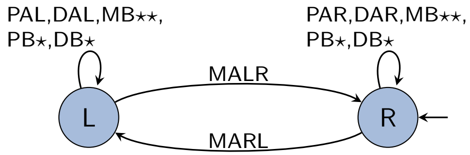
Initialization Step: Atomic Projection for Package

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



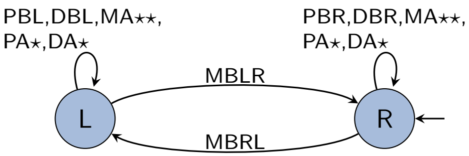
Initialization Step: Atomic Projection for Truck A

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



Initialization Step: Atomic Projection for Truck B

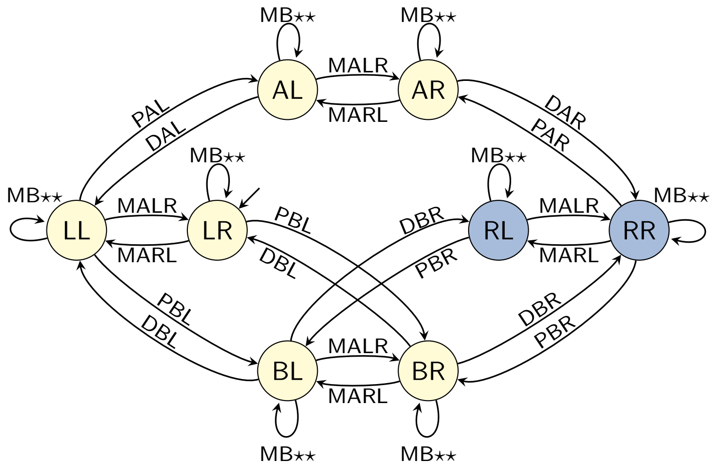
$\mathcal{T}^{\pi_{\{\text{truck B}\}}}$:



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

First Merge Step

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



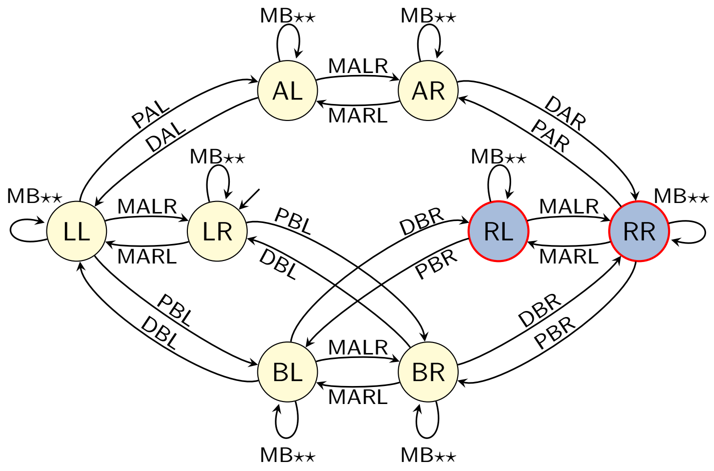
current FTS: $\{\mathcal{T}_1, \mathcal{T}^{\pi_{\{\text{truck B}\}}}\}$

Need to Shrink?

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

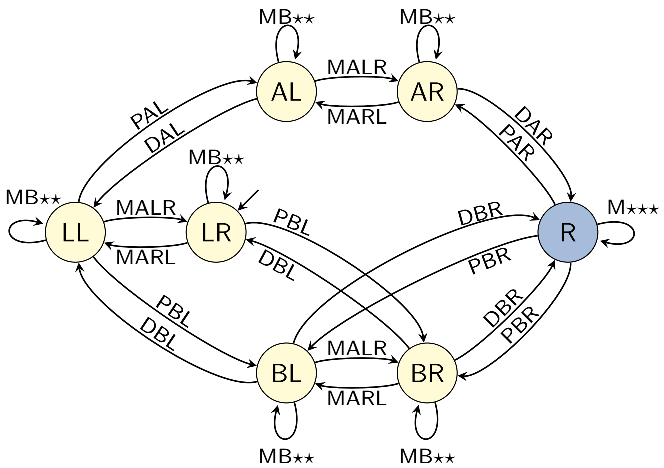
First Shrink Step

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



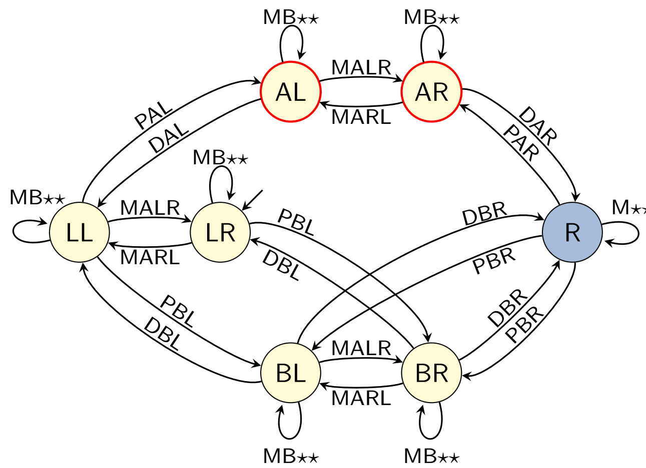
First Shrink Step

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



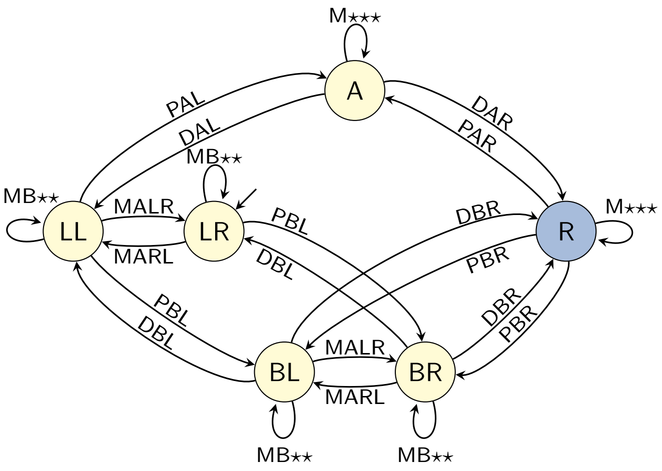
First Shrink Step

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



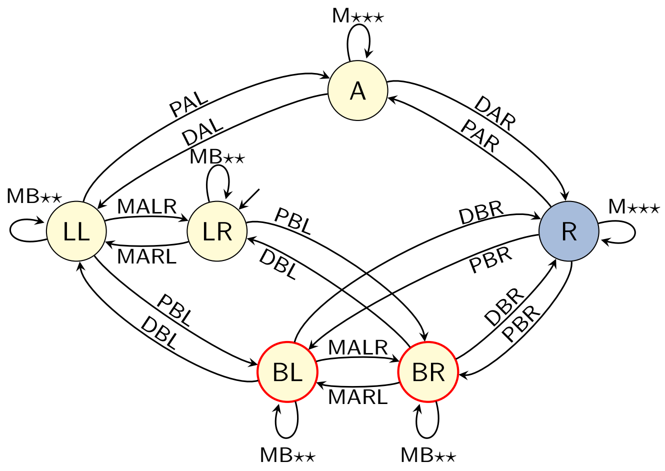
First Shrink Step

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



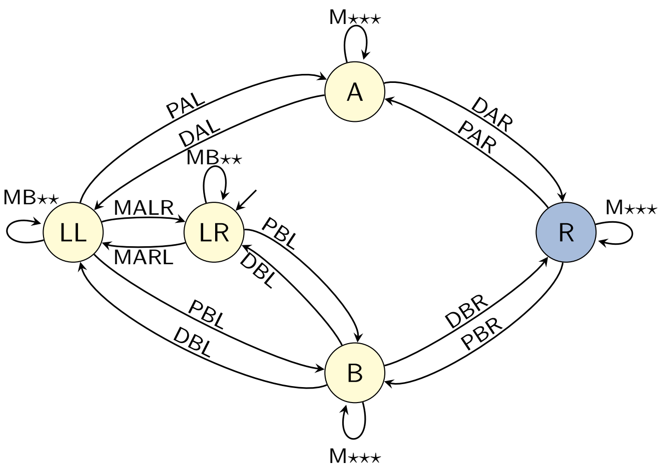
First Shrink Step

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



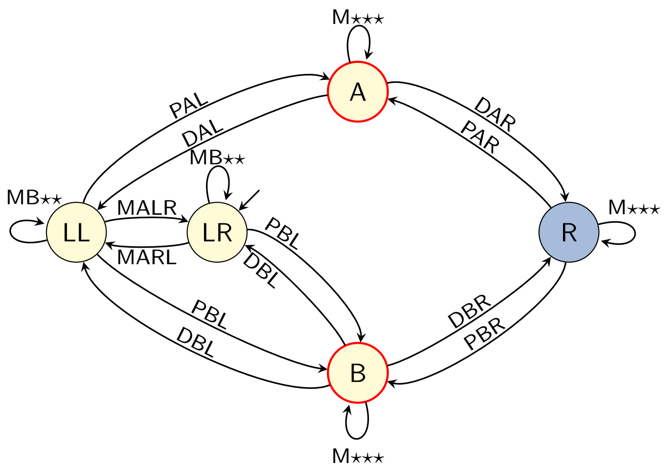
First Shrink Step

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



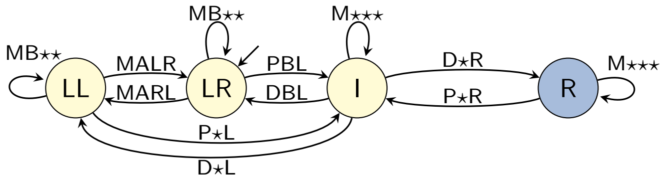
First Shrink Step

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



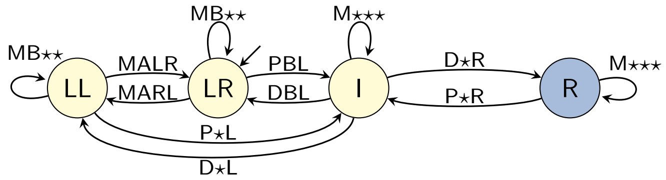
First Shrink Step

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



First Shrink Step

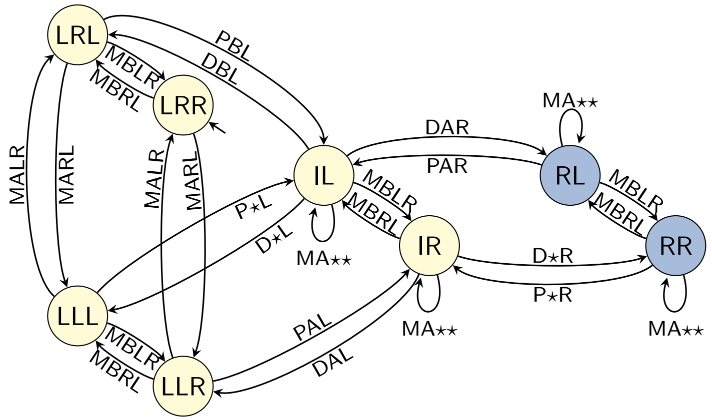
$\mathcal{T}_2 :=$ some abstraction of $\mathcal{T}_1$



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

Second Merge Step

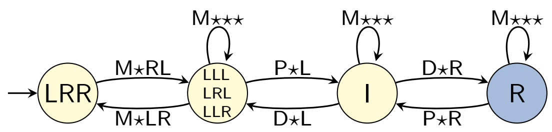
$\mathcal{T}_3 := \mathcal{T}_2 \otimes \mathcal{T}^{\pi_{\{\text{truck B}\}}}$:



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

Another Shrink Step?

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



- We get a heuristic value of 3 for the initial state,
  better than any PDB heuristic that is a proper abstraction.
- The example generalizes to arbitrarily many trucks,
  even if we stick to the fixed size limit of 8.

D8.3 Heuristic Properties

Properties of Merge-and-Shrink Heuristics

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

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

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

- size limits
- merge strategy
- shrink strategy

Merge-and-Shrink as Sequence of Transformations

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

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

Transformations

Definition (Transformation)

Let $\mathcal{T} = \langle S, L, c, T, s_0, S_\star \rangle$ and $\mathcal{T}' = \langle S', L, c, T', s'_0, S'_\star \rangle$ be transition systems with the same labels and costs.
Let $\sigma : S \to S'$ map the states of $\mathcal{T}$ to the states of $\mathcal{T}'$.

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

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

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

Special Transformations

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

Merge transformations are always conservative and exact.

Shrink transformations are always conservative.

Composing Transformations

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

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

Properties of Merge-and-Shrink Heuristics

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

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

D8.4 Further Topics and Literature

Further Topics in Merge and Shrink

Further topics in merge-and-shrink abstraction:

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

Literature (1)

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.

Literature (2)

Raz Nissim, Jörg Hoffmann and Malte Helmert.
Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstractions in Optimal Planning.
Proc. IJCAI 2011, pp. 1983–1990, 2011.
Introduces bisimulation-based shrinking.

Malte Helmert, Patrik Haslum, Jörg Hoffmann and Raz Nissim.
Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces.
Journal of the ACM 61 (3), pp. 16:1–63, 2014.
Detailed journal version of the previous two publications.

Literature (3)

Silvan Sievers, Martin Wehrle and Malte Helmert.
Generalized Label Reduction for Merge-and-Shrink Heuristics.
Proc. AAAI 2014, pp. 2358–2366, 2014.
Introduces modern version of label reduction.
(There was a more complicated version before.)

Gaojian Fan, Martin Müller and Robert Holte.
Non-linear merging strategies for merge-and-shrink based on variable interactions.
Proc. SoCS 2014, pp. 53–61, 2014.
Introduces UMC and MIASM merging strategies

D8.5 Summary

Summary (1)

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

Summary (2)

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

Summary (3)

- Merge-and-shrink abstractions can be analyzed by viewing them as a sequence of transformations.
- We only use conservative transformations, and hence merge-and-shrink heuristics for SAS^+ tasks are admissible and consistent.
- Merge-and-shrink heuristics for SAS^+ tasks that only use exact transformations are perfect.