Journal of Artificial Intelligence Research 71 (2021) 781-883 Submitted 12/2020; published 08/2021

Merge-and-Shrink: A Compositional Theory of Transformations of Factored Transition Systems

Silvan Sievers SILVAN.SIEVERS@UNIBAS.CH
Malte Helmert MALTE.HELMERT@UNIBAS.CH
University of Basel, Department of Mathematics and Computer Science
Spiegelgasse 1, 4051 Basel, Switzerland

Abstract

The *merge-and-shrink* framework has been introduced as a general approach for defining abstractions of large state spaces arising in domain-independent planning and related areas. The distinguishing characteristic of the merge-and-shrink approach is that it operates directly on the factored representation of state spaces, repeatedly modifying this representation through transformations such as *shrinking* (abstracting a factor of the representation), *merging* (combining two factors), *label reduction* (abstracting the way in which different factors interact), and *pruning* (removing states or transitions of a factor).

We provide a novel view of the merge-and-shrink framework as a "toolbox" or "algebra" of transformations on factored transition systems, with the construction of abstractions as only one possible application. For each transformation, we study desirable properties such as *conservativeness* (overapproximating the original transition system), *inducedness* (absence of spurious states and transitions), and *refinability* (reconstruction of paths in the original transition system from the transformed one). We provide the first complete characterizations of the conditions under which these desirable properties can be achieved. We also provide the first full formal account of *factored mappings*, the mechanism used within the merge-and-shrink framework to establish the relationship between states in the original and transformed factored transition system.

Unlike earlier attempts to develop a theory for merge-and-shrink, our approach is fully compositional: the properties of a sequence of transformations can be entirely understood by the properties of the individual transformations involved. This aspect is key to the use of merge-and-shrink as a general toolbox for transforming factored transition systems. New transformations can easily be added to our theory, with compositionality taking care of the seamless integration with the existing components. Similarly, new properties of transformations can be integrated into the theory by showing their compositionality and studying under which conditions they are satisfied by the building blocks of merge-and-shrink.

1. Introduction

Many problems in artificial intelligence can be cast as state-space search, where the objective is to find a sequence of transitions from an initial state to a state satisfying a goal condition, or to prove the absence of such a transition sequence (Pearl, 1984). Classical planning (Ghallab, Nau, & Traverso, 2004) is such an area where state spaces are specified in a compact representation because the number of states in most practically interesting problem instances is too large for explicit representations. A common approach for dealing with such large state spaces is to *abstract* them, i.e., to map them to smaller, more manageable state spaces that serve as the basis of distance heuristics (e.g., Culberson & Schaeffer, 1998; Katz & Domshlak, 2010; Yang, Culberson, Holte, Zahavi, & Felner, 2008) or refinement methods (e.g., Knoblock, 1994; Bäckström & Jonsson, 2013).

Merge-and-shrink abstractions (Dräger, Finkbeiner, & Podelski, 2009) are a general class of abstractions that subsumes the abstractions underlying the well-known *pattern database* heuristics (e.g., Culberson & Schaeffer, 1998; Edelkamp, 2001) and has been successfully applied in the context of optimal classical planning (e.g., Helmert, Haslum, & Hoffmann, 2007; Nissim, Hoffmann, & Helmert, 2011; Katz, Hoffmann, & Helmert, 2012; Hoffmann, Kissmann, & Torralba, 2014; Helmert, Haslum, Hoffmann, & Nissim, 2014; Sievers, Wehrle, & Helmert, 2014; Fan, Müller, & Holte, 2014). In recent work, merge-and-shrink has increasingly been used not just to provide abstractions in classical planning, but as a general framework for reasoning about factored transition systems and their properties, for example to prove unsolvability (Hoffmann et al., 2014), to find symmetries (Sievers, Wehrle, Helmert, Shleyfman, & Katz, 2015) and state dominance relationships (Torralba & Hoffmann, 2015), for symbolic search (Torralba, Linares López, & Borrajo, 2018), and as an alternative representation for planning tasks (Torralba & Sievers, 2019).

The merge-and-shrink framework operates on a *factored transition system*, i.e., a set of explicitly represented transition systems (called *factors*) which together represent a *joint transition system*. For example, the state space of a SAS$^+$ planning task (Bäckström & Nebel, 1995) can be described by a factored transition system where each factor is an *atomic transition system*, describing how the operators of the planning task affect a single state variable. The given factored transition system is then iteratively modified through transformations such as *merging*, *shrinking*, *pruning* or *label reduction* in order to achieve a given computational objective, such as computing an abstraction of the original factored transition system or identifying symmetries.

There is a substantial body of prior work on merge-and-shrink, including past attempts to develop a comprehensive theory for the framework (Helmert et al., 2014). However, the existing theory does not allow us to fully understand the properties of merge-and-shrink through the properties of its components. A given transformation might be conservative (over-approximating) in isolation, but not in the context of other transformations, leading to a complex interplay between transformations and elaborate restrictions on allowable combinations of transformations that limit the use of merge-and-shrink as a general toolbox.

Our main contribution in this paper is to develop a solid theoretical foundation for merge-and-shrink as a *compositional* theory of transformations of factored transition systems. The framework as a whole can be understood by understanding its components. For example, a combination of conservative transformations is always conservative, and if we introduce a new conservative transformation to the framework, none of the existing theory needs to be revisited.

In more detail, we make the following contributions:

- In Section 3, we provide a theory of *transformations of* (non-factored) *transition systems* and define desirable properties of such transformations. For example, transformations can exactly preserve the behavior of the transition system, overapproximate it, or preserve or overapproximate it on the set of reachable states only. We show that transformations can be composed in a natural way and that composed transformations inherit the common properties of the component transformations. We further analyze the effect that transformations with certain properties have on the structure of the transformed transition systems and show how these properties allow us to derive state-space search heuristics with certain properties such as admissibility or perfection. While our development of the theory is (and needs to be) different, it shares many aspects with a theory of abstractions of state spaces developed by Bäckström and Jonsson (2012a, 2013), from which we draw significant inspiration.

- In Section 4, we discuss how *factored representations* can be used to represent states, transition systems, and mappings between states compactly. Using these representations, we can lift our treatment of transformations to a factored setting. This leads to a view of the merge-and-shrink framework as factored transformations of factored transition systems. We also explain how classical planning is an example application naturally giving rise to the kind of factored representations we consider. While the idea of describing merge-and-shrink in terms of transformations and factored transition systems is not novel, we provide a more comprehensive treatment than earlier works. Our treatment of factored mappings (Section 4.3) is entirely novel in this form.

- The following sections describe the four core transformations of the merge-and-shrink framework: shrinking (Section 5), merging (Section 6), label reduction (Section 7), and pruning (Section 8). Compared to earlier work discussing these transformations in the literature, our treatment is more rigorous because it is rooted in the theory of transformations discussed in the preceding sections and more fine-grained because we consider a wider range of transformation properties than earlier work. We also close some gaps in earlier results, for example by providing a complete characterization of the conditions under which shrinking is an exact transformation.

  Of the four transformations, label reduction plays a central role because it has been the main obstacle to a compositional theory of merge-and-shrink in the past (cf. Helmert et al., 2014). Our treatment here is a significantly extended version of a conference paper (Sievers et al., 2014). The earlier paper only considered what we call *atomic* label reductions and only provided a partial characterization of the properties of atomic label reductions. We extend this to a complete characterization of atomic label reductions and prove that analyzing the properties of non-atomic label reductions involves **coNP**-complete decision problems.

  Our formal treatment of the pruning transformation is entirely new. Earlier work on merge-and-shrink at most contains a brief mention of this transformation with no theory (e.g., Helmert et al., 2014, Section 4.3).

- Section 9 provides an extensive discussion of related literature on merge-and-shrink and beyond. Starting from the evolution of merge-and-shrink and its use for computing abstraction heuristics, it covers a wide range of other applications of the framework, both existing ones and possible future research directions. It also discusses more far-reaching connections of merge-and-shrink to the computer science literature, including automata theory, computer-aided verification, constraint programming, and knowledge representation.

We refrain from presenting experimental results due to the already significant length of this paper. However, Section 9.1 discusses many papers that experimentally evaluate different instantiations of the merge-and-shrink approach and demonstrate its usefulness.

2. Background

In this section we provide the necessary background on concepts used to model state-space search problems. We begin by recalling some basic notations and terminology about functions.

Definition 1 (Inverse of a Function)**.** *Let $f : X \to Y$ be a function. For $y \in Y$, the* inverse *of f is defined as the set $f^{-1}(y) = \{x \in X \mid f(x) = y\}$. We extend this definition to subsets $Y' \subseteq Y$ by defining $f^{-1}(Y') = \{x \in X \mid f(x) \in Y'\}$.*

For notational convenience it will be useful to formally define *partial functions* from a set X to a set Y, i.e., functions defined on some subset of X that map into the set Y.

Definition 2 (Partial Function)**.** *Let X and Y be sets. We say that f is a* partial function *from X to Y, in symbols $f : X \nrightarrow Y$, if f is a function from some subset $X' \subseteq X$ to Y. We write $dom(f)$ for X', the domain of f. If $dom(f) = X$, f is called a* total function*.*

For $x \in X$ with $x \notin dom(f)$, we say that $f(x)$ is undefined*. All expressions involving undefined values are undefined. For example, if $f : X \times Y \nrightarrow Z$ and $g : X \nrightarrow X$ and $g(x)$ is undefined, then $f(g(x), y)$ is undefined for all $y \in Y$.*

We will frequently consider the composition $g \circ f$ of two partial functions, defined as $(g \circ f)(x) = g(f(x))$. It is easy to see that $dom(g \circ f) = dom(f) \cap f^{-1}(dom(g))$, i.e., $g \circ f$ is defined for a given value x iff $x \in dom(f)$ and $f(x) \in dom(g)$ (which is equivalent to saying $x \in f^{-1}(dom(g))$).

Next, we define labeled transition systems as a natural way to represent state spaces, and we consider heuristic search as a technique for finding solutions.

Definition 3 (Labeled Transition System)**.** *A* labeled transition system *(or* transition system *for short) is a tuple $\Theta = \langle S, L, c, T, S_\mathrm{I}, S_\mathrm{G} \rangle$ where S is a finite set of* states*, L is a finite set of* transition labels*, $c : L \to \mathbb{R}_0^+$ is a* label cost function*, $T \subseteq S \times L \times S$ is a set of* labeled transitions*, $S_\mathrm{I} \subseteq S$ is the set of* initial states *and $S_\mathrm{G} \subseteq S$ is the set of* goal states*.*

A transition system can be viewed as a directed graph whose nodes correspond to states of the state space and whose arcs correspond to transitions between states. Transitions are labeled, and the label typically identifies the cause of the transition in the state space, such as an event happening in a discrete-event system or an action executed by a planning agent. Labels can induce multiple transitions of a transition system, and they are associated with a cost incurred by the transition.

The following definition introduces some notation and terminology.

Definition 4 (Paths, Costs, Plans)**.** *Let $\Theta = \langle S, L, c, T, S_\mathrm{I}, S_\mathrm{G} \rangle$ be a transition system. We write $s \xrightarrow{\ell} s'$ to denote a transition $\langle s, \ell, s' \rangle$ from s to s' with label ℓ, and we may write $s \xrightarrow{\ell} s' \in \Theta$ for $s \xrightarrow{\ell} s' \in T$, and similarly, $s \in \Theta$ for $s \in S$, whenever Θ is not specified further.*

A path *from $s \in S$ to $s' \in S$ is a sequence $\pi = \langle t_1, \dots, t_n \rangle$ of transitions such that there exist states $s = s_0, \dots, s_n = s'$ with $t_i = (s_{i-1} \xrightarrow{\ell_i} s_i) \in T$ for all $1 \leq i \leq n$. As a special case, the empty path $\langle\rangle$ is a path from s to s for all $s \in S$. The* cost *$c(\pi)$ of such a path is the accumulated cost of the labels of the transitions, i.e., $c(\pi) = \sum_{i=1}^{n} c(\ell_i)$. We allow empty paths $\pi = \langle\rangle$ iff $s = s'$. The cost of π is 0 in this case.*

*A path from some state s to some goal state $s' \in S_\mathrm{G}$ is also called an s-*plan*. An s-plan for some initial state $s \in S_\mathrm{I}$ is also called a* plan*. If a plan (or s-plan) has minimal cost among all plans (or s-plans), it is called* optimal*.*

Our definition of transition systems is a slight generalization of the usual definition of transition systems in automated planning and related areas because it permits a *set* of initial states rather than exactly one initial state. The merge-and-shrink framework does not actually require transition systems with multiple initial states, but it can be formalized more cleanly if we permit (empty)

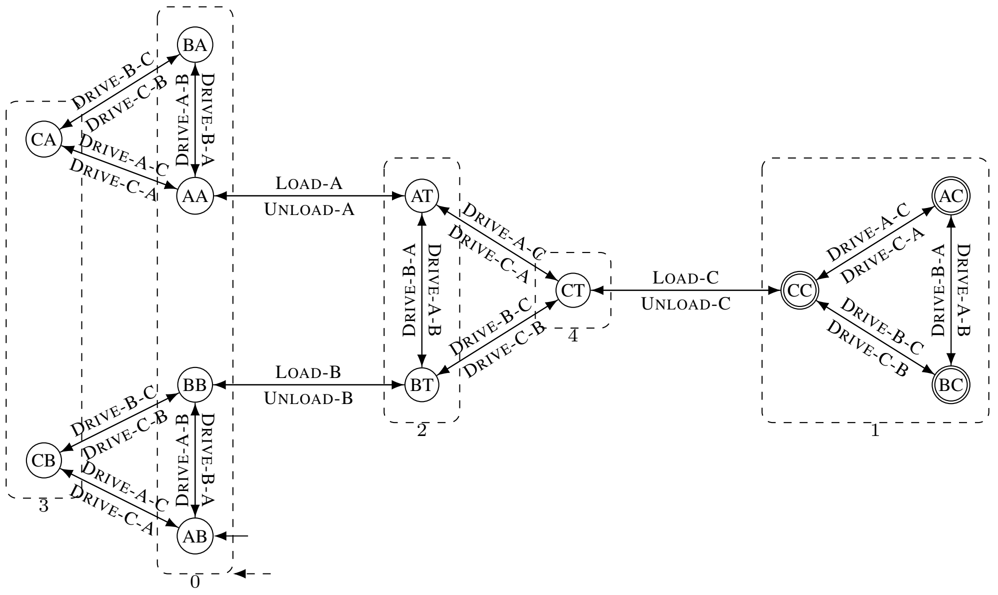


Figure 1: Example of a transition system. Also: induced transition system $\Theta(\Pi)$ of the example planning task Π of Figure 4.

transition systems with *no* initial states, and having a set of initial states is an easy way to make this possible. A side advantage is that definitions for initial and goal states can often look more similar if they are both defined as sets. The transition systems that we consider as inputs to our algorithms generally have exactly one initial state, and we will refer to it simply as *the* initial state. However, our algorithms work equally well with multiple initial states, with the semantics that a plan for any initial state counts as a plan for the transition system.

Figure 1 shows an example transition system. Ignore the dashed boxes for the moment. States are denoted by circles, goal states by double circles, and the (only) initial state (i.e., AB) has an unlabeled incoming arrow. We draw only one transition with two arrows to denote that there are transitions in both directions between the two states, and each of the two labels of such transitions only corresponds to one direction. We will give some intuition for the transition system in Section 4 where we discuss planning tasks and factored transition systems and show that this transition system is induced by a planning task. For now, we just use it as an opaque example of a transition system.

A common approach for finding plans in transition systems is by *heuristic search*, which explores the states of a transition system with the help of distance estimators called *heuristics*. We briefly recall the concept of heuristics (e.g., Pearl, 1984).

Definition 5 (Heuristic). *Let Θ be a transition system with states S and label cost function c. A* heuristic *for Θ is a function $h_\Theta : S \to \mathbb{R}_0^+ \cup \{\infty\}$. A heuristic h_Θ is called*

- perfect *if $h_\Theta(s) = h^*_\Theta(s)$ for all states s of Θ, where $h^*_\Theta(s)$ is the cost of an optimal s-plan or ∞ if no s-plan exists.*
- goal-aware *if $h_\Theta(s) = 0$ for all goal states s of Θ.*
- consistent *if $h_\Theta(s) \leq c(\ell) + h_\Theta(t)$ for all transitions $s \xrightarrow{\ell} t \in \Theta$.*

- admissible *if* $h_\Theta(s) \leq h^*_\Theta(s)$ *for all states* s *of* Θ.

We denote heuristics by h instead of h_Θ if the transition system is clear from context. The value $h(s)$ produced by a heuristic for state $s \in S$ is called the *heuristic estimate* or *heuristic value* for this state. Heuristic search algorithms use heuristic estimates as approximations of the actual cost (also called *perfect heuristic*) $h^*(s)$ of reaching a goal state from s.

The perfect heuristic is clearly of interest because it produces the best possible estimates. Admissible heuristics result in optimal solutions when used within search algorithms like A* (Hart, Nilsson, & Raphael, 1968) or IDA* (Korf, 1985). For admissible heuristics, higher heuristic values mean better approximations of h^* because heuristic values are bounded from above by h^* and hence underestimation is the only possible form of inaccuracy. If a heuristics is admissible and consistent, then A* never needs to *reopen* a state (Dechter & Pearl, 1985). In this setting, higher heuristic values are generally preferable because they lead to less search effort, with some caveats (Dechter & Pearl, 1985; Holte, 2010). Goal-awareness together with consistency implies admissibility. As we will see in the following section, we can use transformations defined on a transition system to derive heuristics.

3. Transformations of Transition Systems

At the core of the merge-and-shrink approach is the notion of *transformations*, which relate a given transition system to a transformed one through state and label mappings. In this section, we study such transformations, in particular discussing desirable properties of transformations which translate into desirable properties of heuristics based on such transformations, such as admissibility or perfection.

Definition 6 (Transformation)**.** *Let* Θ *be a transition system with states* S *and labels* L*, and let* Θ' *be a transition system with states* S' *and labels* L'*. A* transformation *of* Θ *into* Θ' *is a tuple* $\tau = \langle \Theta', \sigma, \lambda \rangle$*, where* Θ' *is called the* transformed transition system*,* $\sigma : S \nrightarrow S'$ *is called the* state mapping*, and* $\lambda : L \nrightarrow L'$ *is called the* label mapping*. We call* Θ *the* original transition system *of* τ*.*

Transformations provide mappings between transition systems, specified by the transformed transition system, a state mapping relating the states of the original transition system to the states of the transformed one, and analogously a label mapping relating the label sets. Our definition of transformations permits *partial* state and label mappings, i.e., the transformation is allowed to remove certain states and labels. For example, this can be used to prune irrelevant states and labels. A common use of transformations is to define *abstractions* of transition systems, which can be used to derive a heuristic for the original transition system (among other uses).

Definition 7 (Heuristic Induced by a Transformation)**.** *Let* $\tau = \langle \Theta', \sigma, \lambda \rangle$ *be a transformation of a transition system* Θ *into transition system* Θ'*. The* heuristic for Θ induced by τ*,* h^τ_Θ *(or* h^τ *for short), is defined as* $h^\tau_\Theta(s) = h^*_{\Theta'}(\sigma(s))$ *for all states* $s \in dom(\sigma)$ *and* $h^\tau_\Theta(s) = \infty$ *for all other states* $s \in \Theta$*.*

In other words, the heuristic value of a state s of Θ is the perfect heuristic value of the transformed state $\sigma(s)$ in the transformed transition system Θ', or ∞ if $\sigma(s)$ is undefined.

3.1 Properties of Transformations

We now define desirable properties of transformations of transition systems.

Definition 8 (Properties of Transformations)**.** *Let $\tau = \langle \Theta', \sigma, \lambda \rangle$ be a transformation of a transition system $\Theta = \langle S, L, c, T, S_{\mathrm{I}}, S_{\mathrm{G}} \rangle$ into a transition system $\Theta' = \langle S', L', c', T', S'_{\mathrm{I}}, S'_{\mathrm{G}} \rangle$. The following list defines properties that τ may have, along with a short-hand name for each property. For example, we say that τ satisfies* $\mathbf{CONS_S}$ *if τ is state-conservative, as defined in the first list entry.*

$\mathbf{CONS_S}$ τ *is* state-conservative *if $dom(\sigma) = S$, i.e., σ is a total function.*

$\mathbf{CONS_L}$ τ *is* label-conservative *if $dom(\lambda) = L$, i.e., λ is a total function.*

$\mathbf{CONS_C}$ τ *is* cost-conservative *if $\forall \ell \in L$: $\ell \in dom(\lambda) \rightarrow c'(\lambda(\ell)) \leq c(\ell)$.*

$\mathbf{CONS_T}$ τ *is* transition-conservative *if*
$\forall s \xrightarrow{\ell} t \in T$: $s \in dom(\sigma) \wedge t \in dom(\sigma) \wedge \ell \in dom(\lambda) \rightarrow \sigma(s) \xrightarrow{\lambda(\ell)} \sigma(t) \in T'$.

$\mathbf{CONS_I}$ τ *is* initial-state-conservative *if $\forall s \in S_{\mathrm{I}}$: $s \in dom(\sigma) \rightarrow \sigma(s) \in S'_{\mathrm{I}}$.*

$\mathbf{CONS_G}$ τ *is* goal-state-conservative *if $\forall s \in S_{\mathrm{G}}$: $s \in dom(\sigma) \rightarrow \sigma(s) \in S'_{\mathrm{G}}$.*

$\mathbf{IND_S}$ τ *is* state-induced *if σ is surjective, i.e., if $\forall s' \in S' \; \exists s \in S$: $s \in \sigma^{-1}(s')$.*

$\mathbf{IND_L}$ τ *is* label-induced *if λ is surjective, i.e., if $\forall \ell' \in L' \; \exists \ell \in L$: $\ell \in \lambda^{-1}(\ell')$.*

$\mathbf{IND_C}$ τ *is* cost-induced *if $\forall \ell' \in L' \; \exists \ell \in L$: $\ell \in \lambda^{-1}(\ell') \wedge c(\ell) = c'(\ell')$*

$\mathbf{IND_T}$ τ *is* transition-induced *if*
$\forall s' \xrightarrow{\ell'} t' \in T' \; \exists s \xrightarrow{\ell} t \in T$: $s \in \sigma^{-1}(s') \wedge t \in \sigma^{-1}(t') \wedge \ell \in \lambda^{-1}(\ell')$.

$\mathbf{IND_I}$ τ *is* initial-state-induced *if $\forall s' \in S'_{\mathrm{I}} \; \exists s \in S_{\mathrm{I}}$: $s \in \sigma^{-1}(s')$.*

$\mathbf{IND_G}$ τ *is* goal-state-induced *if $\forall s' \in S'_{\mathrm{G}} \; \exists s \in S_{\mathrm{G}}$: $s \in \sigma^{-1}(s')$.*

$\mathbf{REF_C}$ τ *is* cost-refinable *if $\forall \ell' \in L' \; \forall \ell \in \lambda^{-1}(\ell')$: $c(\ell) = c'(\ell')$.*

$\mathbf{REF_T}$ τ *is* transition-refinable *if*
$\forall s' \xrightarrow{\ell'} t' \in T' \; \forall s \in \sigma^{-1}(s') \; \exists s \xrightarrow{\ell} t \in T$: $t \in \sigma^{-1}(t') \wedge \ell \in \lambda^{-1}(\ell')$.

$\mathbf{REF_G}$ τ *is* goal-state-refinable *if $\forall s' \in S'_{\mathrm{G}} \; \forall s \in \sigma^{-1}(s')$: $s \in S_{\mathrm{G}}$.*

Based on these basic properties, we define the following derived properties. In general, $\mathbf{A} = \mathbf{B} + \mathbf{C}$ means that τ has property $\mathbf{A}$ if it has properties $\mathbf{B}$ and $\mathbf{C}$, and we group together related properties like $\mathbf{CONS_X} + \mathbf{CONS_Y}$ by writing them as $\mathbf{CONS_{X+Y}}$.

- conservative*:* $\mathbf{CONS} = \mathbf{CONS_{S+L+C+T+I+G}}$
- induced*:* $\mathbf{IND} = \mathbf{IND_{S+L+C+T+I+G}}$
- refinable*:* $\mathbf{REF} = \mathbf{REF_{C+T+G}}$

*Conservative transformations (***CONS***) are also called* abstractions *or* homomorphisms. *Abstractions that are also induced (***CONS** $+$ **IND***) are called* induced abstractions *or* strict homomorphisms. *Abstractions that are refinable (***CONS** $+$ **REF***) are called* exact *transformations. An* exact induced *transformation combines all three properties (***CONS** $+$ **IND** $+$ **REF***).*

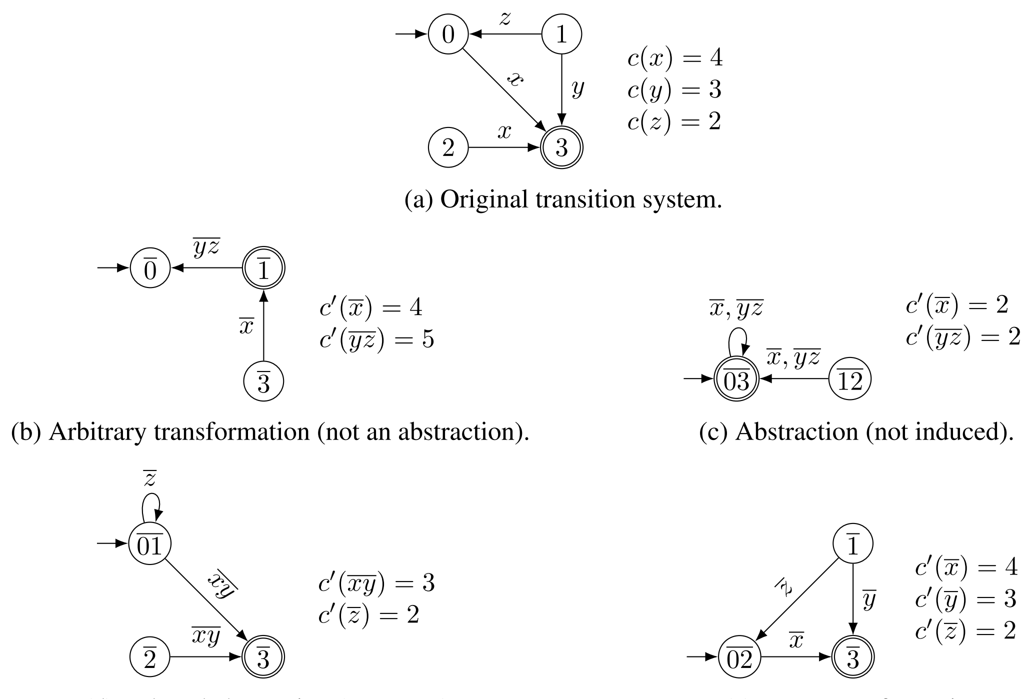
Informally speaking, a transformation τ of a transition system Θ into transition system Θ' is an abstraction (homomorphism) if all behaviors possible in Θ are preserved by τ: every state or label of Θ has a corresponding state or label in Θ' ($\mathbf{CONS_{S+L}}$), every transition of Θ has a corresponding abstract transition in Θ' ($\mathbf{CONS_T}$) of the same or lower cost ($\mathbf{CONS_C}$), and every initial or goal state has a corresponding abstract initial or goal state ($\mathbf{CONS_{I+G}}$). As we will show (in Theorem 3), this is sufficient for deriving admissible and consistent heuristics from τ.

Among these abstractions, induced abstractions (strict homomorphisms) are in some sense the most accurate ones. They must not contain any abstract states ($\mathbf{IND_S}$) or labels ($\mathbf{IND_L}$) beyond those that the state and label mappings map to. While they include all transitions, initial states and goal states that an abstraction must include, they do not include any additional ones beyond those required by the abstraction property ($\mathbf{IND_{T+I+G}}$). Finally, transformed label costs must correspond to the cost of some original label ($\mathbf{IND_C}$), which together with cost-conservativeness implies that the cost of a transformed label must be the minimum cost of its preimage labels. It is not difficult to show that for every total state mapping σ and total label mapping λ, there exists a unique transition system Θ' such that $\tau = \langle \Theta', \sigma, \lambda \rangle$ is an induced abstraction. Hence, induced abstractions are uniquely described by their state and label mappings, and we say that Θ' is the transition system induced by σ and λ. Induced abstractions are practically desirable because they provide the largest possible heuristic values (and hence, because of admissibility, the most accurate possible heuristics) among all abstractions with the same state and label mappings. They are also theoretically desirable because they can be fully understood and analyzed in terms of the state and label mappings.

Exact transformations are conservative "in both directions": intuitively, refinability means that all behaviors possible in Θ' are also possible in Θ. All transformed transitions $s' \xrightarrow{\ell'} t'$ can be mapped back to original transitions $s \xrightarrow{\ell} t$ for *all* preimages s of s' ($\mathbf{REF_T}$), all preimages of goal states are goal states ($\mathbf{REF_G}$), and the label mapping does not affect the label costs ($\mathbf{REF_C}$). Together with the abstraction property, this implies that Θ and Θ' behave in essentially the same way. To make this formal, we will prove (in Theorem 5) that heuristics based on exact transformations are perfect.

We remark that we define transition-refinability in such a way that given a transition $s' \xrightarrow{\ell'} t'$ of Θ', Θ has a corresponding transition *for all* preimages s of s' and *some* preimage t of t'. One could alternatively consider a definition where Θ must have a transition *for all* t and *some* s. Both definitions give rise to notions of abstract paths being refinable to concrete paths, but the alternative definition does not lead to a perfect heuristic. One could of course also require corresponding transitions to exist for *all* s and *all* t, but this is unnecessarily restrictive as our weaker property already leads to a perfect heuristic.

In a similar vein, one could define an *initial-state-refinable* property $\mathbf{REF_I}$ analogous to $\mathbf{REF_G}$, requiring that all preimages of initial states are initial states. But this property would not be useful for any of the applications of transformations that we describe in this paper. We ultimately aim to construct abstractions for *forward search* which allow us to compute distance estimates from arbitrary states to the goal states. If instead we considered abstractions for *backward search*, we would need to compute distance estimates from the initial state(s) to arbitrary (subgoal) states. For



(a) Original transition system.

(b) Arbitrary transformation (not an abstraction).

(c) Abstraction (not induced).

(d) Induced abstraction (not exact).

(e) Exact transformation.

Figure 2: Four different transformations of the transition system in part (a).

this purpose, we would need $\mathbf{REF_I}$ instead of $\mathbf{REF_G}$ and a variant of $\mathbf{REF_T}$ that quantifies over *all* t and *some* s. In the interest of brevity in an already long paper, we do not study these properties.

3.1.1 EXAMPLE

Figure 2 illustrates four example transformations with different properties. The original transition system is shown in part (a) of the figure. The other parts of the figure show transformations of this transition system. We use undecorated numbers and letters to denote states and labels of the original transition system and overlined symbols to denote states and labels of the transformed transition systems. If a state s of the original transition system is mapped to some state of the transformed transition system, then the name of the state includes $\overline{s}$. For example, a transformed state whose preimage consists of states 0 and 3 is denoted by $\overline{03}$. We proceed analogously for labels. All transformations are state-induced and label-induced, so there are no transformed states and labels other than those that the original states and labels map to.

Figure 2(b) shows the first transformation, an example of a non-abstraction transformation. Indeed, it satisfies only two of the six properties of a conservative transformation, as it is label-conservative and initial-state-conservative. It is not state-conservative because the state mapping is undefined for state 2 (no abstract state includes $\overline{2}$). It is not goal-state-conservative because $\overline{3}$ is not a goal state even though 3 is. It is not transition-conservative because the transformed transition system has no transition corresponding to $0 \xrightarrow{x} 3$. Finally, it is not cost-conservative because the cost of $\overline{yz}$ is higher (5) than the costs of y and z (4 and 3).

Figure 2(c) shows an abstraction: all states and labels have corresponding transformed states and labels, all transitions induce transformed transitions, all initial/goal states induce transformed initial/goal states, and all labels are mapped to labels of the same or lower cost. The abstraction is not induced because the transition $\overline{03} \xrightarrow{\overline{yz}} \overline{03}$ is not induced by any original transition (violating $\mathbf{IND_T}$) and also because the label cost 2 of $\overline{x}$ differs from the cost 4 of its only preimage label x (violating $\mathbf{IND_C}$).

Figure 2(d) shows an induced abstraction: all states, labels, transitions, initial/goal states and label costs are induced by the original transition system. The transformation is not exact for several reasons: for example, it is not transition-refinable because the transition $\overline{01} \xrightarrow{\overline{z}} \overline{01}$ has no matching original transition for the preimage 0 of $\overline{01}$, as neither $0 \xrightarrow{z} 0$ nor $0 \xrightarrow{z} 1$ are transitions of the original transition system. The transformation is also not cost-refinable because the cost of label $\overline{xy}$ (3) is lower than the cost of x (4).

Finally, Figure 2(e) shows an exact induced transformation: it is conservative, induced and refinable.

3.1.2 RELATIONSHIP TO BÄCKSTRÖM AND JONSSON

Bäckström and Jonsson (2013) describe a framework that models abstractions as transformations from a labeled digraph $\mathbb{G}$ to a labeled digraph $\mathbb{G}'$. As in our case, transformations must specify how the states and labels of the two digraphs are related. A major difference is that we relate the states and labels of the two digraphs by (partial) functions, while Bäckström and Jonsson consider more general relations. A transformation in their setting is represented by a set-valued function f that maps states of $\mathbb{G}$ to sets of states of $\mathbb{G}'$ (with further constraints that essentially specify that f defines a bijection between equivalence classes of the states of $\mathbb{G}$ and $\mathbb{G}'$) and an arbitrary relation R between the labels of $\mathbb{G}$ and the labels of $\mathbb{G}'$.

For example, this notion of transformation allows mapping a single state to multiple states, and it is reversible in the sense that for each transformation from $\mathbb{G}$ to $\mathbb{G}'$, there exists an inverse transformation from $\mathbb{G}'$ to $\mathbb{G}$. We restrict ourselves to (functional) state and label mappings because these are simpler and sufficient for our purposes. A further difference is that the transition graph formalism used by Bäckström and Jonsson does not include notions of initial states, goal states or label costs, although of course these can be associated with transition graphs externally.

Bäckström and Jonsson also study properties of transformations, some of which are quite similar to properties we define, the main difference being that most of their properties do not consider labels but only depend on the relationship between states. In some more detail, Bäckström and Jonsson define several "method properties" for the state mapping f, including $\mathbf{M}_\uparrow$, meaning that f is a total function (rather than set-valued); $\mathbf{R}_\uparrow$, meaning in our notation that if $s \xrightarrow{\ell} t \in \mathbb{G}$, then there is $s' \xrightarrow{\ell'} t' \in \mathbb{G}'$ such that $R(\ell, \ell')$; and $\mathbf{C}_\uparrow$, meaning in our notation that if $R(\ell, \ell')$ and $s \xrightarrow{\ell} t \in \mathbb{G}$, then there is $s' \xrightarrow{\ell'} t' \in \mathbb{G}'$ such that $s' \in f(s)$ and $t' \in f(t)$. They call a transformation a homomorphism if it satisfies $\mathbf{M}_\uparrow\mathbf{R}_\uparrow\mathbf{C}_\uparrow$.[1] Our properties $\mathbf{CONS_{S+L+T}}$ correspond to this notion of homomorphism, while our definition of abstraction/homomorphism requires additional conditions on initial and goal states and label costs, which are not present in their formalism.

Bäckström and Jonsson further define the converse properties $\mathbf{R}_\downarrow$ and $\mathbf{C}_\downarrow$ in the obvious way (e.g., a transformation from $\mathbb{G}$ to $\mathbb{G}'$ satisfies $\mathbf{C}_\downarrow$ if it semantically corresponds to a transformation

1. Bäckström and Jonsson use concatenation to denote the combination of properties where we use the "+" symbol. Moreover, for any symbol $\mathbf{X}$ where $\mathbf{X}_\uparrow$ and $\mathbf{X}_\downarrow$ are properties, their combination is abbreviated as $\mathbf{X}_\updownarrow$.

from $\mathbb{G}'$ to $\mathbb{G}$ which satisfies $\mathbf{C}_{\uparrow}$) and call a transformation satisfying $\mathbf{M}_{\uparrow}\mathbf{R}_{\updownarrow}\mathbf{C}_{\updownarrow}$ a strong homomorphism. This corresponds to adding $\mathbf{IND_T}$ to $\mathbf{CONS_{S+L+T}}$. Again, for a transformation to be an induced abstraction/strict homomorphism in our sense, we require additional conditions on initial and goal states and label costs. Interestingly, the strong homomorphisms of Bäckström and Jonsson do not have analogues of $\mathbf{IND_S}$ and $\mathbf{IND_L}$. For example, according to their definition, a strong homomorphism may introduce an arbitrary number of additional states with no incident transitions. In our setting, this would violate $\mathbf{IND_S}$. Their strong homomorphisms may also introduce additional labels and transitions for these labels, for example by mapping each original transition $s \xrightarrow{\ell} t$ to two transitions $s \xrightarrow{\ell_1} t$ and $s \xrightarrow{\ell_2} t$, where ℓ_1 and ℓ_2 act as two independent copies of ℓ. In our setting, this would violate $\mathbf{IND_L}$.

Despite these differences the approach by Bäckström and Jonsson is very similar in spirit and execution to ours. Indeed, it was one of the major inspirations for our definition of transformations between transition systems as well as for the more general idea of studying the merge-and-shrink framework in terms of a family of transformation properties.

3.2 Composition of Transformations

An important property of transformations is that they *compose*: if τ is a transformation of Θ into Θ' and τ' is a transformation of Θ' into Θ'', then τ and τ' can be combined into a single transformation of Θ into Θ''. In this section we prove this composition property and show that the composed transformation inherits all common properties (such as conservativeness or inducedness) of its components. This is a key result because it permits us to understand the joint behavior of any sequence of transformations in terms of the properties of the individual transformation steps. Later, we will introduce the merge-and-shrink framework as a family of transformations. It follows that we can understand the properties of merge-and-shrink by understanding the individual transformations in the family. We begin by defining the composition of transformations.

Definition 9 (Composition of Transformations)**.** *Let $\tau = \langle \Theta', \sigma, \lambda \rangle$ be a transformation of a transition system Θ into a transition system Θ', and let $\tau' = \langle \Theta'', \sigma', \lambda' \rangle$ be a transformation of Θ' into a transition system Θ''. The* composition *of τ' and τ is the transformation of Θ into Θ'' defined as $\tau' \circ \tau = \langle \Theta'', \sigma' \circ \sigma, \lambda' \circ \lambda \rangle$.*

We remind the reader that state and label mappings can be partial and refer to Definition 2 for the composition of partial functions. It is easy to verify that the composition of two transformations is indeed a transformation. The following theorem shows that a composed transformation inherits the common properties of its component transformations. All proofs for this section can be found in Appendix A.

Theorem 1. *Let X be any of the properties of transformations from Definition 8. Let τ be a transformation of transition system Θ into transition system Θ' with property X, and let τ' be a transformation of Θ' into transition system Θ'' with property X. Then the composed transformation $\tau'' = \tau' \circ \tau$ also has the property X.*

3.3 Effect of Properties of Transformations on Heuristics

The properties of transformations affect the properties of heuristics induced by these transformations. In this section we study this relationship, with an emphasis on admissible and consistent heuristics and on perfect heuristics.

If $\pi = \langle s_0 \xrightarrow{\ell_1} s_1, \dots, s_{n-1} \xrightarrow{\ell_n} s_n \rangle$ is a path in a transition system Θ and $\tau = \langle \Theta', \sigma, \lambda \rangle$ is a transformation of Θ, we write $\tau(\pi)$ for the *transformation* $\langle \sigma(s_0) \xrightarrow{\lambda(\ell_1)} \sigma(s_1), \dots, \sigma(s_{n-1}) \xrightarrow{\lambda(\ell_n)} \sigma(s_n) \rangle$ of π. Note that $\tau(\pi)$ is not necessarily a path of Θ'. In particular, by our general convention for functions that refer to undefined values, $\tau(\pi)$ is undefined if any $\sigma(s_i)$ or $\lambda(\ell_i)$ is undefined. Moreover, even if $\tau(\pi)$ is defined, it can include transitions that are not transitions of Θ'. We call $\tau(\pi)$ a *legal path* in Θ' if it is defined and a path in Θ'. We extend the definition of the cost of a path as the sum of the costs of the labels in the sequence to all cases where $\tau(\pi)$ is defined, whether or not it is legal. If $\pi = \langle\rangle$ is the empty path from state s to itself, we consider $\tau(\pi)$ legal iff $\sigma(s)$ is defined. That is, for empty paths, the legality of $\tau(\pi)$ depends on which state s we associate with the path. (This is a slight abuse of notation because the state s is not explicit from the notation of π.)

Conversely, if $\pi' = \langle s'_0 \xrightarrow{\ell'_1} s'_1, \dots, s'_{n-1} \xrightarrow{\ell'_n} s'_n \rangle$ is a path in Θ', by $\tau^{-1}(\pi')$ we denote the *refinements* of π', i.e., $\tau^{-1}(\pi') = \{\langle s_0 \xrightarrow{\ell_1} s_1, \dots, s_{n-1} \xrightarrow{\ell_n} s_n \rangle \mid s_i \in \sigma^{-1}(s'_i)$ for all $0 \leq i \leq n$ and $\ell_i \in \lambda^{-1}(\ell'_i)$ for all $1 \leq i \leq n\}$. Similarly to the preceding discussion, transition sequences in $\tau^{-1}(\pi')$ may include transitions that are not transitions of Θ. An element of $\tau^{-1}(\pi')$ is called a *legal path* in Θ if it is actually a path in Θ.

3.3.1 FORMAL RESULTS

The first result of this section is concerned with conservative transformations.

Theorem 2. *Let Θ be a transition system with states S and label costs c, and let $\tau = \langle \Theta', \sigma, \lambda \rangle$ be a transformation of Θ into transition system Θ' with label costs c'. Let π be a path in Θ. Then:*

1. *If τ is state- and label-conservative ($\mathbf{CONS_{S+L}}$), then $\tau(\pi)$ is defined.*
2. *If τ is transition-conservative ($\mathbf{CONS_{T}}$) and $\tau(\pi)$ is defined, then $\tau(\pi)$ is a legal path in Θ'.*
3. *If τ is transition- and goal-state-conservative ($\mathbf{CONS_{T+G}}$), π is an s-plan and $\tau(\pi)$ is defined, then $\tau(\pi)$ is a $\sigma(s)$-plan for Θ'.*
4. *If τ is state-, label-, transition-, initial-state- and goal-state-conservative ($\mathbf{CONS_{S+L+T+I+G}}$) and π is a plan for Θ, then $\tau(\pi)$ is a plan for Θ'.*
5. *If τ is cost-conservative ($\mathbf{CONS_{C}}$) and $\tau(\pi)$ is defined, then $c'(\tau(\pi)) \leq c(\pi)$.*

This theorem shows that conservative transformations preserve the existence of paths and plans and do not lead to an increase in plan cost. This directly translates to admissibility and consistency of heuristics based on such transformations.

Theorem 3. *Let τ be a transformation of a transition system Θ. The heuristic h^τ for Θ induced by τ is*

1. *goal-aware if τ is state- and goal-state-conservative ($\mathbf{CONS_{S+G}}$),*
2. *consistent if τ is state-, label-, cost- and transition-conservative ($\mathbf{CONS_{S+L+C+T}}$), and*
3. *admissible if τ is state-, label-, cost-, transition- and goal-state-conservative ($\mathbf{CONS_{S+L+C+T+G}}$).*

In summary, conservative transformations (abstractions) give rise to admissible and consistent heuristics. We remark that the only conservativeness property we do *not* need is initial-state-conservativeness ($\mathbf{CONS_I}$), as induced heuristics do not depend on the initial state. Initial-state-conservativeness will become important in Section 8, which also shows that goal-awareness, consistency and admissibility already hold under slightly weaker conditions (Theorem 22 on page 837). Strictly speaking, this later result makes Theorem 3 redundant, but we think it is useful to already state this simpler result here to keep some more complicated concepts limited to Section 8.

In the following, we explore conditions under which transformations give rise to perfect heuristics. As a first step, we study some key properties of refinable transformations.

Theorem 4. *Let Θ be a transition system with label costs c, and let $\tau = \langle \Theta', \sigma, \lambda \rangle$ be a transformation of Θ into transition system Θ' with label costs c'. Let π' be a path from state s' to state t' in Θ', and let $s \in \sigma^{-1}(s')$. Then:*

1. *If τ is transition-refinable ($\mathbf{REF_T}$), then there exists a legal path $\pi \in \tau^{-1}(\pi')$ from s to some state $t \in \sigma^{-1}(t')$ in Θ.*
2. *If τ is transition-refinable and goal-state-refinable ($\mathbf{REF_{T+G}}$) and π' is an s'-plan for Θ', then there exists an s-plan $\pi \in \tau^{-1}(\pi')$ for Θ.*
3. *If τ is cost-refinable ($\mathbf{REF_C}$), then $c(\pi) = c'(\pi')$ for all $\pi \in \tau^{-1}(\pi')$.*

The theorem shows that with refinable transformations, plans in the transformed transition system can be "transformed back" into plans of the original transition system, at the same cost. This result is roughly converse to Theorem 2, and it implies that the shortest path costs in the transformed transition system under a refinable transformation can never be lower than the corresponding shortest path costs in the original transition system. This leads to the following result.

Theorem 5. *Let τ be a transformation of a transition system Θ. The heuristic h^τ for Θ induced by τ is*

1. *lower-bounded by h^* ($h^*(s) \leq h^\tau(s)$ for all states s) if τ is refinable ($\mathbf{REF}$) and*
2. *perfect if τ is exact ($\mathbf{CONS} + \mathbf{REF}$), or more generally if τ is $\mathbf{CONS_{S+L+C+T+G}} + \mathbf{REF}$.*

The main message of this theorem is that exact transformations give rise to perfect heuristics. (The slightly more general variant of the second result takes into account that among the six conservativeness properties, initial-state-conservativeness is not required.) Of course, this is no coincidence: we chose the name "exact" for conservative and refinable transformations precisely for this reason.

3.3.2 RELATIONSHIP TO BÄCKSTRÖM AND JONSSON

Earlier, we discussed the relationship of our definition of transformations and their properties to earlier work by Bäckström and Jonsson (2013). Similar relationships can be identified for the results in this section.

Bäckström and Jonsson study several "metric" properties for concepts related to path costs. Because their notion of transition systems does not include costs, for the discussion of metric properties they augment transition systems with cost functions c between arbitrary pairs of states. Their cost functions are unrelated to the transition labels and only loosely related to the transition structure: in principle, they consider arbitrary cost functions with the only requirement that $c(s,t) = \infty$ iff there is no path from state s to state t. In particular, there is no requirement that path costs are based on transition costs; it is permissible to have $c(s,s) > 0$ or for c to violate the triangle inequality. However, all their results relevant to this discussion are with respect to a more limited class of cost functions induced by *weights* on transitions in the natural way, which is much closer to our model. The main difference is that we associate costs with transition labels, while Bäckström and Jonsson associate costs directly with transitions. Having transition labels as an intermediary between transitions and their costs is important for factored representations of transition systems, which we discuss in the following section.

Bäckström and Jonsson do not include goal states in their formalization, so they study notions of admissibility and consistency for distances between arbitrary pairs of states rather than from a given state to the nearest goal state. Their main result on admissibility and consistency is that their notion of homomorphisms (transformations with property $\mathbf{M}_{\uparrow}\mathbf{R}_{\uparrow}\mathbf{C}_{\uparrow}$ in their notation, corresponding to our property $\mathbf{CONS_{S+L+T}}$ as discussed in Section 3.1) gives rise to admissible and consistent heuristics if we additionally require the equivalent of property $\mathbf{CONS_C}$ for their way of modeling transition costs. This result (Theorem 15 in their paper) is analogous to our Theorem 3.

Of course, the observation that homomorphisms induce admissible and consistent heuristics is a very well-known basic property of abstractions, so at a high level, this is neither a new contribution by Bäckström and Jonsson nor is it a new contribution by us. Rather, in both works the value of the results comes from the way that these well-known observations are connected to a general model of transformations. In this sense, proving that abstractions induce admissible and consistent heuristics in both works acts as a sanity test that (in both cases) the notions of "abstractions" and "induced heuristics" were defined in a reasonable way.

Bäckström and Jonsson do not discuss conditions for perfect heuristics, so their work does not contain a result equivalent to Theorem 5, which establishes that exact transformations induce perfect heuristics. However, they do cover several notions of refinability, leading to results with some similarity to our Theorem 4.[2] Specifically, they consider properties of transformations under which every path in the transformed transition system is *downward state refinable* (in the following *refinable* for short).

They study several variants of refinability. Expressed in our notation, the three main variants can be described as follows for a given transformation $\tau = \langle \Theta', \sigma, \lambda \rangle$ of transition system Θ:

- *trivial* refinability ($\mathbf{P}_{\mathbf{T}\downarrow}$ in the notation of Bäckström and Jonsson): for every path from s' to t' in Θ', there exists a path from some $s \in \sigma^{-1}(s')$ to some $t \in \sigma^{-1}(t')$ in Θ.
- *weak* refinability ($\mathbf{P}_{\mathbf{W}\downarrow}$): for every path $s'_0 \to \cdots \to s'_k$ in Θ', there exist states $s_0 \in \sigma^{-1}(s'_0), \dots, s_k \in \sigma^{-1}(s'_k)$ such that there exist paths from s_{i-1} to s_i in Θ for all $1 \leq i \leq k$.
- *strong* refinability ($\mathbf{P}_{\mathbf{S}\downarrow}$): for every path $s'_0 \to \cdots \to s'_k$ in Θ' and for all states $s_0 \in \sigma^{-1}(s'_0), \dots, s_k \in \sigma^{-1}(s'_k)$, there exist paths from s_{i-1} to s_i in Θ for all $1 \leq i \leq k$.

Bäckström and Jonsson (2012a) show that strong refinability implies weak refinability, which in turn implies trivial refinability, while the converse statements do not hold. (Note that strong

2. Some of the following results are due to an earlier, more detailed treatment of refinement by the same authors (Bäckström & Jonsson, 2012a).

refinability only implies weak refinability if σ is surjective. Only then can "$\forall s \in \sigma^{-1}(s') : \varphi$" imply "$\exists s \in \sigma^{-1}(s') : \varphi$" for a property φ. In the work of Bäckström and Jonsson, state mappings are always total and surjective, i.e., our properties $\mathbf{CONS_S}$ and $\mathbf{IND_S}$ hold by definition.)

One major difference between trivial/weak/strong refinability and our notion of refinability employed in Theorem 4 is that Bäckström and Jonsson treat transition systems as unlabeled digraphs for the purposes of refinement; transition labels are ignored. Also, their notions are centered on paths, not transitions, which in particular means that a single transition may be refined into an arbitrarily long path.

Both of these choices mean that their notion of refinement cannot be used for quantitative properties of heuristics like the relationship $h^{\tau}(s) \geq h^{*}(s)$ that holds for our notion of refinement. However, refinement in the style of Bäckström and Jonsson *can* in principle be used for a qualitative variation of this property based on reachability, namely that $h^{\tau}(s) = \infty$ whenever $h^{*}(s) = \infty$, i.e., *perfect dead end detection*. Of the above three notions of refinability, only strong refinability guarantees perfect dead end detection: weak refinability (and by implication trivial refinability) is too weak because of the existential quantification over $s_0 \in \sigma^{-1}(s'_0)$. Strong refinability does guarantee perfect dead end detection, but is more restrictive than necessary for this purpose: it suffices to require that for every path from s' to t' in Θ' and all $s \in \sigma^{-1}(s')$, there exists a path from s to *some* $t \in \sigma^{-1}(t')$ in Θ. This condition is equivalent to a fourth (unnamed) variant of refinability in the work of Bäckström and Jonsson (2012a, 2013), which they denote by $\mathbf{P}_{\downarrow}$.

3.4 Summary

We conclude our discussion of transformations with a brief summary of the main results. We introduced the concept of transformations and their properties. We showed that transformations can be composed and that this composition inherits the common properties of the component transformations. We have seen that conservative transformations (abstractions) give rise to admissible and consistent heuristics, while exact (conservative and refinable) transformations give rise to perfect heuristics. These relationships between transformation properties and heuristic properties, together with the earlier observation that induced abstractions are in a formal sense the "most informative" abstractions, are the reason why we consider conservativeness, inducedness and refinability desirable properties of transformations. In Sections 5–8, we will come back to these properties of transformations in the context of the merge-and-shrink framework. In particular, this will allow us to conclude under which conditions the merge-and-shrink transformations give rise to admissible or perfect heuristics.

4. Factored Representations

In the previous section, we defined a theory of transformations of transition systems and their properties and showed how such transformations can be composed. In this section, we describe how to use *factored* representations of transition systems and transformations. Such factored representations have the great advantage of being more compact than "flat" (nonfactored) representations, so that they can serve as the basis of efficient algorithms.

In particular, we define *factored transition systems*, which are represented as tuples of component transition systems (factors), as a compact representation of a single nonfactored transition system. We also define *factored mappings*, which can serve as a compact representation of state mappings and of heuristic functions. Together with label mappings as defined previously (which do not need a factored representation), these factored representations can be used to define *factored transformations* as transformations operating directly on factored representations.

The key insight underlying factored representations is that large sets can be compactly described as "products" of small components. We will refer to these components as *variables*. Variable-based representations are for example used to describe states in planning formalisms such as STRIPS (Fikes & Nilsson, 1971) or SAS$^+$ (Bäckström & Nebel, 1995), to describe assignments in constraint satisfaction problems (Dechter, 2003), or to describe the joint state of a system of components in model checking (Clarke, Grumberg, & Peled, 1999). We will refer to a collection of finite-domain variables of this kind as a *variable space*.

Definition 10 (Variable Space). *A* variable space *is a tuple $\mathcal{V} = \langle v_1, \dots, v_n \rangle$ of variables with a finite domain. We write $dom(v)$ for the domain of v, which can be an arbitrary finite set of values.*

We assume throughout the paper that all variables in a variable space have a separate identity (i.e., $\mathcal{V}$ does not contain duplicate variables), so that we can also treat variable spaces as *sets* (rather than tuples) of variables where convenient. For example, we will write $v \in \mathcal{V}$ to denote that $\mathcal{V}$ contains the variable v. It would of course be possible to define variable spaces as sets of variables in the first place, but having a total order on the variables is often convenient for specifying algorithms that work on variable spaces in an unambiguous way, for example when we want to iterate over the variables in a variable space.

The main purpose of variable spaces is to compactly represent the set of assignments to the variables. For example, n variables with domains of size k compactly describe k^n possible assignments. In this sense, we can think of a variable space as a *factored set*. We now formalize the notion of assignments and some related concepts.

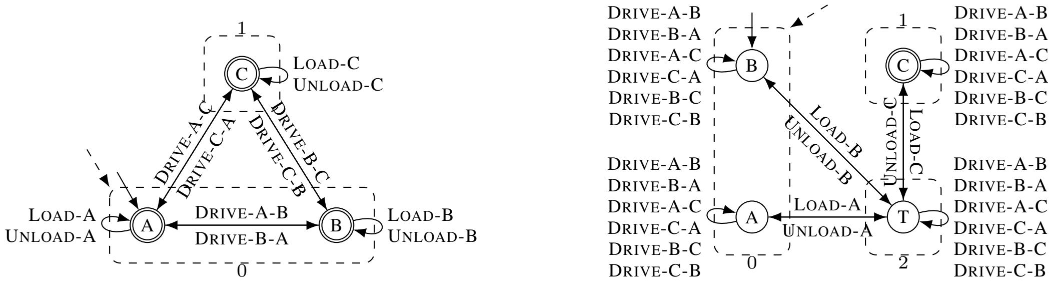
Definition 11 (Assignment, Partial Assignment, Consistent Assignments). *Let $\mathcal{V} = \langle v_1, \dots, v_n \rangle$ be a variable space. A* partial assignment *for $\mathcal{V}$ is a valuation of some subset $V \subseteq \mathcal{V}$, i.e., a function α defined on V that maps each element $v \in V$ to some element in $dom(v)$. We write $\alpha[v]$ for this element of $dom(v)$ to visually distinguish assignments from other mappings. We write $vars(\alpha)$ to denote the set of variables on which α is defined.*[3]

Two partial assignments α and α' are consistent *if $\alpha[v] = \alpha'[v]$ for all $v \in vars(\alpha) \cap vars(\alpha')$.*

A partial assignment with $vars(\alpha) = \mathcal{V}$ is called an assignment *for $\mathcal{V}$. We will also write assignments as tuples of assigned values, i.e., use the notation $\langle d_1, \dots, d_n \rangle$ for the assignment $\{v_1 \mapsto d_1, \dots, v_n \mapsto d_n\}$. We write $\mathcal{A}(\mathcal{V})$ for the set of all assignments for $\mathcal{V}$.*

Equipped with these basic concepts, we can now study factored representations of transition systems and their transformations in detail. We introduce factored transition systems in Section 4.1 and show how classical planning tasks induce factored transition systems in Section 4.2. In Section 4.3 we define factored mappings, describe how they can be used to represent state mappings of factored transition systems, and show how they can be composed efficiently. Finally, we put the pieces together in Section 4.4 where we define factored transformations, discuss their use for deriving heuristics, and give a high-level account of the merge-and-shrink framework.

3. Partial assignments can of course be viewed as partial functions (Definition 2). In this view, $vars(\alpha)$ is simply the domain of definition of α, which we denote by $dom(\alpha)$ for partial functions in general. But we believe that using $dom(\alpha)$ could lead to confusion with the notation $dom(v)$ for the domain of *variables*, which is an unrelated use of the word "domain". (Both dom notations are firmly established conventions, one used in mathematics in general and the other used in areas of artificial intelligence that deal with variable spaces, such as automated planning and constraint programming.)



(a) Θ_1. Also: atomic factor Θ^T for v_T.

(b) Θ_2. Also: atomic factor Θ^P for v_P.

Figure 3: Example of a factored transition system. Also: induced factored transition system of the example planning task of Figure 4.

4.1 Factored Transition Systems

We first define factored transition systems.

Definition 12 (Factored Transition System)**.** *A* factored transition system *is a tuple $F = \langle \Theta^1, \dots, \Theta^n \rangle$ of transition systems where each transition system Θ^i has the same set of labels L and the same label cost function c, i.e., $\Theta^i = \langle S^i, L, c, T^i, S^i_{\mathrm{I}}, S^i_{\mathrm{G}} \rangle$ for all $1 \leq i \leq n$.*

A factored transition system consists of transition systems, also called *factors*, sharing the same labels with the same cost. Figure 3 shows an example factored transition system with two factors Θ_1 and Θ_2. Ignore the dashed boxes for the moment. We intentionally avoid providing detailed intuition for the example now because we want to use it to show how factored transition systems work at a mechanical level. We will give some intuition for the example in the next subsection, where we discuss planning tasks.

The purpose of factored transition systems is to provide a concise representation of large transition systems by means of their *product systems*, which we define next.

Definition 13 (Product)**.** *Let $F = \langle \Theta^1, \dots, \Theta^n \rangle$ be a factored transition system with $\Theta^i = \langle S^i, L, c, T^i, S^i_{\mathrm{I}}, S^i_{\mathrm{G}} \rangle$ for all $1 \leq i \leq n$. The* product *(also:* product system, synchronized product*) of F is the transition system defined as $\bigotimes F = \langle S^\otimes, L, c, T^\otimes, S^\otimes_{\mathrm{I}}, S^\otimes_{\mathrm{G}} \rangle$, where $S^\otimes = \prod_{i=1}^n S^i$, $T^\otimes = \{\langle s^1, \dots, s^n \rangle \xrightarrow{\ell} \langle t^1, \dots, t^n \rangle \mid s^i \xrightarrow{\ell} t^i \in T^i$ for all $1 \leq i \leq n\}$, $S^\otimes_{\mathrm{I}} = \prod_{i=1}^n S^i_{\mathrm{I}}$, and $S^\otimes_{\mathrm{G}} = \prod_{i=1}^n S^i_{\mathrm{G}}$.*[4]

The product system is the transition system which is implicitly represented through the synchronized behavior of its factors. We can think of a factored transition system F as a variable space with additional structure. In this view, each factor $\Theta^i \in F$ is a variable whose domain consists of the states S^i of the factor, and the states of the product system are exactly the assignments of F, i.e., $\mathcal{A}(F)$. In other words, a product state is defined by specifying the component state of each factor. Following our general conventions for variable spaces, we can write such states using assignment notation $\{\Theta^i \mapsto s^i \mid 1 \leq i \leq n\}$ or as tuples $\langle s^1, \dots, s^n \rangle$.

Labels are used to synchronize the factors of the factored transition system via the labeled transitions: there is a transition between two states in the product system iff all factors have a

4. The notation $\prod_{i=1}^n A^i$ stands for the Cartesian product of the sets A^i, i.e., $\prod_{i=1}^n A^i = A^1 \times \cdots \times A^n$.

transition between the corresponding component states labeled with the same label. A state is an initial/goal state in the product if all its components are initial/goal states in the respective factors. This implies that a sequence of labels corresponds to a plan in the factored transition system iff it corresponds to a plan in all factors.

Consider the factored transition system with the two factors shown in Figure 3, again ignoring the dashed boxes. The product system of this factored transition system is the transition system shown in Figure 1 on page 785, apart from the "names" of states: for example, state AB should formally be written as $\langle \mathrm{A}, \mathrm{B} \rangle$ to properly refer to a state of the product system. In the following, we will treat transition systems as equivalent if they only differ in the names of states. With this convention, the product operation is associative and commutative. To exemplify the definition of the product, we see that in the first factor there is a transition A $\xrightarrow{\text{DRIVE-A-B}}$ B, and in the second factor there is a self-looping transition B $\xrightarrow{\text{DRIVE-A-B}}$ B, and hence in the product there must be a transition AB $\xrightarrow{\text{DRIVE-A-B}}$ BB. Figure 1 indeed shows such a transition. The goal states of the product system are all those where the second component reads C because only state C is a goal state in the second factor, and all states are goal states in the first factor.

4.2 Classical Planning

While the concepts we present in this work apply to arbitrary factored transition systems, we illustrate them through their application to classical planning (e.g., Ghallab et al., 2004), one of the areas in which factored representations of transition systems arise. Planning deals with finding plans, i.e., paths from an initial state to some goal state, in transition systems induced by planning tasks.

We consider planning tasks in the SAS$^+$ formalism (Bäckström & Nebel, 1995), augmented with action costs.

Definition 14 (Planning Task)**.** *A* planning task *is a tuple* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_\mathrm{I}, s_\mathrm{G} \rangle$ *with the following components:*

- $\mathcal{V}$ *is a variable space. The variables in* $\mathcal{V}$ *are called* state variables*, and the (partial) assignments for* $\mathcal{V}$ *are called* (partial) states.
- $\mathcal{O}$ *is a finite set of* operators*, where each operator* $o \in \mathcal{O}$ *has an associated partial state* $pre(o)$ *called the* precondition *of* o*, an associated partial state* $eff(o)$ *called the* effect *of* o*, and an associated non-negative value* $cost(o) \in \mathbb{R}^+_0$*, called the* cost *of* o*.*
- s_I *is a state called the* initial state.
- s_G *is a partial state called the* goal.

A planning task Π induces a transition system $\Theta(\Pi)$, which gives it its semantics. The states of $\Theta(\Pi)$ are the assignments for the variables $\mathcal{V}$ of Π, labels and their costs correspond to operators of Π, and there is a transition $s \xrightarrow{\ell} t$ in $\Theta(\Pi)$ if $pre(\ell)$ is consistent with s, and t is the state that is consistent with $eff(\ell)$ and satisfies $t[v] = s[v]$ for all $v \notin vars(eff(\ell))$. There is exactly one initial state in $\Theta(\Pi)$, namely s_I. The goal states of $\Theta(\Pi)$ are all states that are consistent with the goal of Π. A *plan* for Π is a plan for $\Theta(\Pi)$ (cf. Definition 4). *Optimal planning* is the problem of finding an optimal plan for a given planning task or showing that no plan exists.

Figure 4 shows a simple logistics planning task with one truck T and one package P. There are three locations A, B, and C. The truck T can move from and to all locations via drive operators.

$$
\begin{array}{lll}
\multicolumn{3}{l}{\Pi = \langle \mathcal{V}, \mathcal{O}, s_\mathrm{I}, s_\mathrm{G} \rangle \text{ with}} \\
\multicolumn{3}{l}{\mathcal{V} = \langle v_T, v_P \rangle \text{ with } \mathit{dom}(v_T) = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\} \text{ and } \mathit{dom}(v_P) = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{T}\}} \\
\mathcal{O} = \{ & \textsc{Drive-A-B}: & \mathit{pre}(\textsc{Drive-A-B}) = \{v_T \mapsto \mathrm{A}\} \\
& & \mathit{eff}(\textsc{Drive-A-B}) = \{v_T \mapsto \mathrm{B}\} \\
& & \mathit{cost}(\textsc{Drive-A-B}) = 1, \\
& \textsc{Drive-A-C}: & \mathit{pre}(\textsc{Drive-A-C}) = \{v_T \mapsto \mathrm{A}\} \\
& & \mathit{eff}(\textsc{Drive-A-C}) = \{v_T \mapsto \mathrm{C}\} \\
& & \mathit{cost}(\textsc{Drive-A-C}) = 1, \\
& \textsc{Drive-B-A}: & \mathit{pre}(\textsc{Drive-B-A}) = \{v_T \mapsto \mathrm{B}\} \\
& & \mathit{eff}(\textsc{Drive-B-A}) = \{v_T \mapsto \mathrm{A}\} \\
& & \mathit{cost}(\textsc{Drive-B-A}) = 1, \\
& \textsc{Drive-B-C}: & \mathit{pre}(\textsc{Drive-B-C}) = \{v_T \mapsto \mathrm{B}\} \\
& & \mathit{eff}(\textsc{Drive-B-C}) = \{v_T \mapsto \mathrm{C}\} \\
& & \mathit{cost}(\textsc{Drive-B-C}) = 1, \\
& \textsc{Drive-C-A}: & \mathit{pre}(\textsc{Drive-C-A}) = \{v_T \mapsto \mathrm{C}\} \\
& & \mathit{eff}(\textsc{Drive-C-A}) = \{v_T \mapsto \mathrm{A}\} \\
& & \mathit{cost}(\textsc{Drive-C-A}) = 1, \\
& \textsc{Drive-C-B}: & \mathit{pre}(\textsc{Drive-C-B}) = \{v_T \mapsto \mathrm{C}\} \\
& & \mathit{eff}(\textsc{Drive-C-B}) = \{v_T \mapsto \mathrm{B}\} \\
& & \mathit{cost}(\textsc{Drive-C-B}) = 1, \\
& \textsc{Load-A}: & \mathit{pre}(\textsc{Load-A}) = \{v_T \mapsto \mathrm{A}, v_P \mapsto \mathrm{A}\} \\
& & \mathit{eff}(\textsc{Load-A}) = \{v_P \mapsto \mathrm{T}\} \\
& & \mathit{cost}(\textsc{Load-A}) = 1, \\
& \textsc{Load-B}: & \mathit{pre}(\textsc{Load-B}) = \{v_T \mapsto \mathrm{B}, v_P \mapsto \mathrm{B}\} \\
& & \mathit{eff}(\textsc{Load-B}) = \{v_P \mapsto \mathrm{T}\} \\
& & \mathit{cost}(\textsc{Load-B}) = 1, \\
& \textsc{Load-C}: & \mathit{pre}(\textsc{Load-C}) = \{v_T \mapsto \mathrm{C}, v_P \mapsto \mathrm{C}\} \\
& & \mathit{eff}(\textsc{Load-C}) = \{v_P \mapsto \mathrm{T}\} \\
& & \mathit{cost}(\textsc{Load-C}) = 1, \\
& \textsc{Unload-A}: & \mathit{pre}(\textsc{Unload-A}) = \{v_T \mapsto \mathrm{A}, v_P \mapsto \mathrm{T}\} \\
& & \mathit{eff}(\textsc{Unload-A}) = \{v_P \mapsto \mathrm{A}\} \\
& & \mathit{cost}(\textsc{Unload-A}) = 1, \\
& \textsc{Unload-B}: & \mathit{pre}(\textsc{Unload-B}) = \{v_T \mapsto \mathrm{B}, v_P \mapsto \mathrm{T}\} \\
& & \mathit{eff}(\textsc{Unload-B}) = \{v_P \mapsto \mathrm{B}\} \\
& & \mathit{cost}(\textsc{Unload-B}) = 1, \\
& \textsc{Unload-C}: & \mathit{pre}(\textsc{Unload-C}) = \{v_T \mapsto \mathrm{C}, v_P \mapsto \mathrm{T}\} \\
& & \mathit{eff}(\textsc{Unload-C}) = \{v_P \mapsto \mathrm{C}\} \\
& & \mathit{cost}(\textsc{Unload-C}) = 1\} \\
\multicolumn{3}{l}{s_\mathrm{I} = \{v_T \mapsto \mathrm{A}, v_P \mapsto \mathrm{B}\}} \\
\multicolumn{3}{l}{s_\mathrm{G} = \{v_P \mapsto \mathrm{C}\}}
\end{array}
$$

Figure 4: Simple logistics planning task with one truck, one package and three locations.

The package P can be loaded into T if it is at the same location as T, and it can be unloaded to the location of T if it is in the truck. All operators have the same cost 1. The induced transition system $\Theta(\Pi)$ is shown in Figure 1 on page 785. States are labeled with two letters, the first denoting the position of the truck (i.e., the value of v_T), the second denoting the position of the package (the value of v_P). The initial state, marked with an unlabeled incoming arrow, is the state AB, i.e., the state where the truck is at A and the package is at B. All states whose second component reads C are goal states because the package is at C in such states. Looking at the transition labeled DRIVE-A-B from state AB to BB, it is clear that the corresponding operator is applicable in the state AB because the truck is at A, and that the state BB is the successor state because the truck is at B and the position of the package remains the same.

A planning task Π also induces a factored representation of $\Theta(\Pi)$ in a natural way. The factors of this representation are called the *atomic factors* of Π.

Definition 15 (Atomic Factor). *Let* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_\mathrm{I}, s_\mathrm{G} \rangle$ *be a planning task. The* atomic factor *for variable* $v \in \mathcal{V}$ *of* Π *is the transition system* $\Theta^v = \langle \mathit{dom}(v), \mathcal{O}, c, T^v, S^v_\mathrm{I}, S^v_\mathrm{G} \rangle$ *where c maps each label* $\ell \in \mathcal{O}$ *to the cost* $\mathit{cost}(\ell)$ *of the operator,* $T^v = \{d \xrightarrow{\ell} d' \mid (v \notin \mathit{vars}(\mathit{pre}(\ell)) \vee \mathit{pre}(\ell)[v] = d) \wedge ((v \notin \mathit{vars}(\mathit{eff}(\ell)) \wedge d' = d) \vee \mathit{eff}(\ell)[v] = d')\}$, $S^v_\mathrm{I} = \{s_\mathrm{I}[v]\}$, *and* $S^v_\mathrm{G} = \{s_\mathrm{G}[v]\}$ *if* $v \in \mathit{vars}(S^v_\mathrm{G})$ *and* $S_\mathrm{G} = \mathit{dom}(v)$ *otherwise.*

An atomic factor represents the behavior of a single state variable of a planning task, with states of the factor corresponding to possible values of its associated state variable. Breaking down the rather complicated definition of the transitions of an atomic factor, two conditions must be met for a transition $\langle d, \ell, d' \rangle$ to exist. Firstly, the operator ℓ either has no precondition on v, or it requires the value d of v as a precondition. Secondly, the operator either does not affect v and we have $d' = d$, or the operator sets v to d' in its effect. Atomic factors are closely related to the concept of *domain transition graphs* (Jonsson & Bäckström, 1998), with the difference that atomic factors have initial and goal states and that they represent *all* operators of the planning task, including self-looping transitions for operators that do not change the value of the variable.

Using atomic factors, we can now define the induced factored transition system of a planning task.

Definition 16 (Induced Factored Transition System). *The* induced factored transition system *of a planning task* Π *with state variables* $\mathcal{V} = \langle v_1, \dots, v_n \rangle$ *is the factored transition system* $F(\Pi) = \langle \Theta^{v_1}, \dots, \Theta^{v_n} \rangle$.

The factored transition system shown in Figure 3 is the induced factored transition system of the example planning task of Figure 4. The transition system shown in Figure 3(a) is the atomic factor for variable v_T, and the one in Figure 3(b) is the atomic factor for variable v_P. Note that for our running example, we use the shorter notation Θ^T and Θ^P rather than Θ^{v_T} and Θ^{v_P}.

An important observation is that the product system of the induced factored transition system of a planning task is identical (apart from names of states) to the induced transition system of the planning task.[5] We have already seen this in our example: the product of the induced factored transition system shown in Figure 3 corresponds to the induced transition system shown in Figure 1. Hence $F(\Pi)$ faithfully represents all aspects of the planning task Π. This is one of the major motivations for using transformations of factored transition systems as the basis of a planning algorithm: we

5. This result was first established by Helmert et al. (2007); see their Theorem 8 and following discussion.

obtain all the compactness benefits of factored representations while preserving all aspects of the planning task.

4.3 Factored Mappings

In order to use factored representations in a transformation-based approach, we do not just need factored representations of transition systems but also of state mappings. For example, if we want to use transformations to derive an abstraction of a factored transition system, we need a way to map states in the original factored transition system ("concrete states") to corresponding states in the transformed factored transition system ("abstract states").

For this purpose we define *factored mappings*, a tree-like data structure that can represent arbitrary functions defined on variable spaces, such as the states of a planning task. As discussed in the previous section, factored transition systems can naturally be viewed as variable spaces where each factor is a variable ranging over the component states of the factor, so factored transition systems and factored mappings work together naturally.

Factored mappings have previously also been called "cascading tables" (Helmert et al., 2014; Torralba, 2015) and "merge-and-shrink representations" (Helmert, Röger, & Sievers, 2015). The following two definitions are based on the treatment of merge-and-shrink representations by Helmert et al. (2015).

Definition 17 (Factored Mapping)**.** Factored mappings (FM*s) over a variable space $\mathcal{V}$ are inductively defined as follows. An FM σ has an associated finite* value set $vals(\sigma) \neq \emptyset$ *and an associated* table σ^{tab}. *σ is either* atomic *or a* merge.

- *If σ is atomic, then it has an associated variable $v \in \mathcal{V}$. Its table is a partial function $\sigma^{\text{tab}} : dom(v) \nrightarrow vals(\sigma)$.*
- *If σ is a merge, then it has a* left component *FM σ_{L} and a* right component *FM σ_{R}. Its table is a partial function $\sigma^{\text{tab}} : vals(\sigma_{\text{L}}) \times vals(\sigma_{\text{R}}) \nrightarrow vals(\sigma)$.*

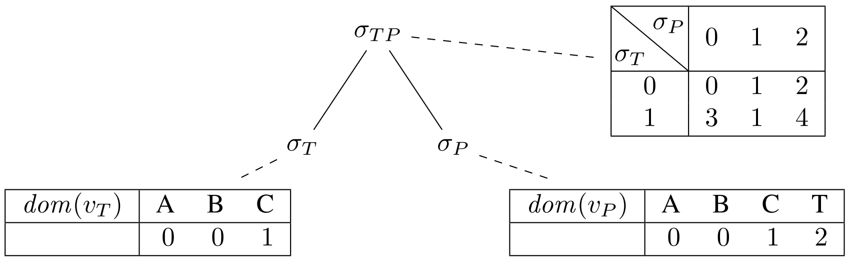
In words, FMs can be viewed as binary trees with merges as inner nodes and atomic FMs as leaves. Leaves have an associated variable and define mappings from values of this variable to some set of values. Inner nodes determine how the mappings of their children are combined.

Definition 18 (Represented Function)**.** *Let σ be an FM over a variable space $\mathcal{V}$. σ* represents *the function $[\![\sigma]\!] : \mathcal{A}(\mathcal{V}) \nrightarrow vals(\sigma)$ which is inductively defined as follows:*

- *If σ is atomic with associated variable v, then $[\![\sigma]\!](\alpha) = \sigma^{\text{tab}}(\alpha[v])$.*
- *If σ is a merge, then $[\![\sigma]\!](\alpha) = \sigma^{\text{tab}}([\![\sigma_L]\!](\alpha), [\![\sigma_R]\!](\alpha))$.*[6]

An FM σ over a variable space $\mathcal{V}$ represents a (partial) function defined on the assignments for $\mathcal{V}$. To compute the function value of a merge FM, the FM recursively computes the values computed by its component FMs and uses these values to index its associated 2-dimensional table to look up the result. The recursion ends at atomic FMs, which look up the values stored for the assignment to their associated variable in their associated 1-dimensional table. It is easy to see that FMs over $\mathcal{V}$ can represent arbitrary functions defined on assignments for $\mathcal{V}$.

6. We remind the reader that this expression is undefined if $[\![\sigma_L]\!](\alpha)$ or $[\![\sigma_R]\!](\alpha)$ is undefined (Definition 2).



$\sigma_T \backslash \sigma_P$	0	1	2
0	0	1	2
1	3	1	4

$dom(v_T)$	A	B	C
	0	0	1

$dom(v_P)$	A	B	C	T
	0	0	1	2

Figure 5: An FM representing an abstraction mapping of the induced transition system of the example planning task of Figure 4.

Our definition of FMs is slightly more general than earlier definitions (Helmert et al., 2015), which require that an FM may only contain one atomic FM for each variable. That is, in the tree representation of the FM, all leaves refer to different variables. FMs that follow this restriction are called *orthogonal*. We permit non-orthogonal FMs because there is no strong reason to disallow them, even though existing work on merge-and-shrink only considers the orthogonal case. A small exception is Section 7.1 of Helmert et al. (2014), which briefly mentions representing additive heuristic ensembles as a non-orthogonal merge-and-shrink heuristic.

Consider the (orthogonal) example FM shown in Figure 5. We generally illustrate an FM by drawing its underlying tree structure, labeling each node with the FM it corresponds to and connecting it to its table via a dotted line. Leaf nodes correspond to atomic FMs and as such do not have children in the tree. Inner nodes correspond to merge FMs and hence have two child nodes, representing their component FMs, connected to them by solid lines. When a table entry is undefined, we generally write this as $\perp$. (Undefined values are not used in this example; they are first used in Section 8.) As can be seen from the two leaf FMs, the FM is defined over the variable space $\langle v_T, v_P \rangle$ with $dom(v_T) = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$ and $dom(v_P) = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{T}\}$. The visualization does not show the value sets, which can be inferred from the tables if we assume that there are no unused values: $vals(\sigma_T) = \{0, 1\}$, $vals(\sigma_P) = \{0, 1, 2\}$, and $vals(\sigma_{TP}) = \{0, 1, 2, 3, 4\}$.

To illustrate the represented function of the example, consider the assignment $\alpha = \{v_T \mapsto \mathrm{A}, v_P \mapsto \mathrm{T}\}$. We can compute $[\![\sigma_{TP}]\!](\alpha)$ as follows:

$$\begin{aligned}
&[\![\sigma_{TP}]\!](\alpha)\\
\overset{1}{=}\;&\sigma_{TP}^{\mathsf{tab}}([\![\sigma_T]\!](\alpha), [\![\sigma_P]\!](\alpha))\\
\overset{2}{=}\;&\sigma_{TP}^{\mathsf{tab}}(\sigma_T^{\mathsf{tab}}(\alpha[v_T]), \sigma_P^{\mathsf{tab}}(\alpha[v_P]))\\
\overset{3}{=}\;&\sigma_{TP}^{\mathsf{tab}}(\sigma_T^{\mathsf{tab}}(\mathrm{A}), \sigma_P^{\mathsf{tab}}(\mathrm{T}))\\
\overset{4}{=}\;&\sigma_{TP}^{\mathsf{tab}}(0, 2)\\
\overset{5}{=}\;&2
\end{aligned}$$

In the first step, the assignment is passed down to the components of σ_{TP}. In the second step, the computation of the represented functions of σ_T and σ_P is resolved to table lookups in their one-dimensional tables because both FMs are atomic. In the third step, the values assigned to the associated variables of the atomic FMs are evaluated. In the fourth step, the values of the tables are looked up and returned, concluding the recursion. Finally, in the fifth step, the returned values are used to index the two-dimensional table of σ_{TP}, which gives 2 as the final result.

The FM σ_{TP} shown in Figure 5 represents an *abstraction* of the example planning task of Figure 4 on page 799. The dashed boxes in Figures 1 and 3 on pages 785 and 797 aggregate states that the example FMs map to the same abstract state. For example, $[\![\sigma_T]\!](\mathrm{A}) = [\![\sigma_T]\!](\mathrm{B}) = 0$, and hence states A and B are included in a dashed box annotated with the number 0 in Figure 3(a). To continue the example of the computation above, the merge FM σ_{TP} maps the value 0 computed by σ_T and the value 2 computed by σ_P to the value 2, which means that all states where the truck is at A or B (specified through σ_T) and the package is in the truck (specified through σ_P) are aggregated, as indicated by the dashed box around the states AT and AB in Figure 1. The dashed box is annotated with the number 2 to show that $[\![\sigma_{TP}]\!](\mathrm{AT}) = [\![\sigma_{TP}]\!](\mathrm{AB}) = 2$.

When using FMs to represent abstraction mappings, each component FM can be viewed as an individual abstraction, so that the overall mapping is defined as a combination of many abstractions. In the example, we cannot distinguish whether truck T is at A or B because σ_T maps them to the same value, and similarly we cannot distinguish whether P is at A or B because σ_P drops this distinction. The merge FM σ_{TP} then further abstracts these two abstractions by dropping the distinction of the truck being at A, B, or C if the package is at the goal location C (both $\langle 0, 1\rangle$ and $\langle 1, 1\rangle$ are mapped to the same value 1).

4.3.1 RELATIONSHIP TO PATTERN DATABASES

We now briefly discuss the relationship of FMs to *pattern databases (PDBs)* (Culberson & Schaeffer, 1998; Edelkamp, 2001). PDBs are lookup tables that store one entry for each possible variable assignment for a subset P (called the *pattern*) of the variables. They can be represented as FMs by using one leaf for each variable in the pattern, combining these leaves with merges in any way (the precise shape of the tree does not matter), and using bijective table functions for every component FM except the root FM, whose table function encodes the heuristic function.

The main difference between the two representations is that PDBs only involve a single table lookup, whereas FMs use nested lookups. The overall space complexity of the representations and the lookup time are of the same magnitude in both cases. In particular, the lookup time for a PDB is $O(|P|)$ because we need to compute a perfect hash value for the given assignment to P, and the FM requires $|P|$ table lookups for the leaves and $|P| - 1$ table lookups for the merges, each taking constant time. With suitably efficient implementations (e.g., Sievers, Ortlieb, & Helmert, 2012), PDBs can be expected to have an advantage in terms of the constant factors involved.

The advantage of FMs becomes apparent when we do not use bijective table functions, which makes it possible to apply abstractions “along the way” and hence obtain potentially much smaller representations for the same abstraction heuristic compared to a PDB. For example, a PDB-style abstraction heuristic that uses information about all state variables in a complex planning task is practically infeasible because it requires computing and storing heuristic values for all states of the task. In contrast, merge-and-shrink heuristics based on FMs commonly use information about all state variables by using an FM representation. It is not difficult to prove that FMs are more general than PDBs in the sense that there exist functions that FMs can compactly represent but PDBs cannot (e.g., Nissim et al., 2011; Helmert et al., 2014). For a more comprehensive discussion of the general representational power and the required size of FMs, we refer to Helmert et al. (2015).

4.3.2 FACTORED-TO-FACTORED MAPPINGS

So far, we have only considered factored mappings that map to a "flat" value set: the input to such factored mappings is factored, but the output is not. We now consider a more general notion of factored mappings where the output is also factored. Such "factored-to-factored" mappings allow mapping from one variable space to another variable space. This makes them particularly suitable for representing mappings between states of factored transition systems.

A factored-to-factored mapping that maps to a variable space with k variables can simply be defined as k individual mappings to "flat" sets, one for each output variable. This is completely analogous to other contexts such as multivariate calculus. For example, a function $f : \mathbb{R}^n \to \mathbb{R}^m$ can be defined by m functions $f_1, \dots, f_m : \mathbb{R}^n \to \mathbb{R}$, one for each output component.

Definition 19 (Factored-to-Factored Mapping)**.** *Let $\mathcal{V} = \langle v_1, \dots, v_n \rangle$ and $\mathcal{V}' = \langle v'_1, \dots, v'_m \rangle$ be variable spaces. Let $\sigma_1, \dots, \sigma_m$ be FMs where each σ_j is defined over $\mathcal{V}$ and maps to $dom(v'_j)$. $\Sigma = \langle \sigma_1, \dots, \sigma_m \rangle$ is called a* factored-to-factored (F2F) mapping from $\mathcal{V}$ to $\mathcal{V}'$. *It represents the partial function $[\![\Sigma]\!] : \mathcal{A}(\mathcal{V}) \nrightarrow \mathcal{A}(\mathcal{V}')$ defined as $[\![\Sigma]\!](\alpha) = \langle \sigma_1(\alpha), \dots, \sigma_m(\alpha) \rangle$.*[7]

In the following, we will use the word *factored mapping* both for the original class of FMs from Definition 17 and the factored-to-factored variant. When we want to emphasize that we do not have an F2F mapping, we will refer to it as *factored-to-nonfactored* (F2N). We use capital Greek letters such as Σ to denote F2F mappings, and (as before) lower-case Greek letters such as σ for F2N mappings.

As in the F2N case, *orthogonal* F2F mappings are of particular interest. Recall that an F2N mapping is called orthogonal if all its atomic components refer to different variables. We extend this definition to F2F mappings by requiring that all atomic components of all F2N mappings that make up the F2F mapping refer to different variables, i.e., all F2N mappings are orthogonal and use disjoint sets of variables.

Because F2N mappings can represent arbitrary functions from variable spaces to nonfactored sets, F2F mappings can represent arbitrary functions from variable spaces to variable spaces. Perhaps the simplest F2F mapping is the *identity mapping*, whose k-th component is a *projection* of the input to its k-th component.

Definition 20 (Projection F2N Mapping and Identity F2F Mapping)**.** *Let $\mathcal{V} = \langle v_1, \dots, v_n \rangle$ be a variable space, and let $1 \leq i \leq n$. The* projection of $\mathcal{V}$ to v_i *is an atomic FM $\pi_i^{\mathcal{V}}$ that is defined over $\mathcal{V}$, has the associated variable v_i, the value set $dom(v_i)$, and a table function mapping d_i to d_i for all $d_i \in dom(v_i)$. The* identity mapping for $\mathcal{V}$ *is the F2F mapping* $\mathrm{id}^{\mathcal{V}} = \langle \pi_1^{\mathcal{V}}, \dots, \pi_n^{\mathcal{V}} \rangle$. *We omit $\mathcal{V}$ from the notation and write π_i and* id *where $\mathcal{V}$ is clear from context.*

It is easy to see that projections and identity mappings are FMs with compact representations that follow the usual semantics: we have $[\![\pi_i]\!](\alpha) = d_i$ and $[\![\mathrm{id}]\!](\alpha) = \alpha$ for all assignments $\alpha = \langle d_1, \dots, d_n \rangle$. The identity mapping considers each variable exactly once and is therefore an example of an orthogonal F2F mapping.

4.3.3 COMPOSITION OF FACTORED MAPPINGS

An important property of F2F mappings is that they are *composable*: if Σ is an F2F mapping from $\mathcal{V}$ to $\mathcal{W}$ and Γ is an F2F mapping from $\mathcal{W}$ to $\mathcal{U}$, then we can construct an F2F mapping $\Delta = \Gamma \circ \Sigma$

7. Again, following Definition 2, $[\![\Sigma]\!](\alpha)$ is undefined as soon as any $\sigma_i(\alpha)$ is undefined.

from $\mathcal{V}$ to $\mathcal{U}$ such that $[\![\Delta]\!] = [\![\Gamma]\!] \circ [\![\Sigma]\!]$, i.e., $[\![\Delta]\!](\alpha) = [\![\Gamma]\!]([\![\Sigma]\!](\alpha))$ for all $\alpha \in \mathcal{A}(\mathcal{V})$. We now describe how $\Gamma \circ \Sigma$ can be efficiently constructed from Γ and Σ.

In the following, we write $\mathit{Merge}(\sigma', \sigma'', T)$ to denote a merge FM σ with $\sigma_{\mathrm{L}} = \sigma'$, $\sigma_{\mathrm{R}} = \sigma''$ and $\sigma^{\text{tab}} = T$, and we write $\mathit{Atomic}(v, T)$ to denote an atomic FM σ with variable v and $\sigma^{\text{tab}} = T$.

$$\langle \gamma_1, \dots, \gamma_k \rangle \circ \Sigma = \langle \gamma_1 \circ \Sigma, \dots, \gamma_k \circ \Sigma \rangle \tag{1}$$

$$\mathit{Merge}(\gamma_{\mathrm{L}}, \gamma_{\mathrm{R}}, \gamma^{\text{tab}}) \circ \Sigma = \mathit{Merge}(\gamma_{\mathrm{L}} \circ \Sigma, \gamma_{\mathrm{R}} \circ \Sigma, \gamma^{\text{tab}}) \tag{2}$$

$$\mathit{Atomic}(w_j, \gamma^{\text{tab}}) \circ \langle \sigma_1, \dots, \sigma_m \rangle = \mathit{apply}(\gamma^{\text{tab}}, \sigma_j) \tag{3}$$

$$\mathit{apply}(T, \mathit{Merge}(\sigma_{\mathrm{L}}, \sigma_{\mathrm{R}}, \sigma^{\text{tab}})) = \mathit{Merge}(\sigma_{\mathrm{L}}, \sigma_{\mathrm{R}}, T \circ \sigma^{\text{tab}}) \tag{4}$$

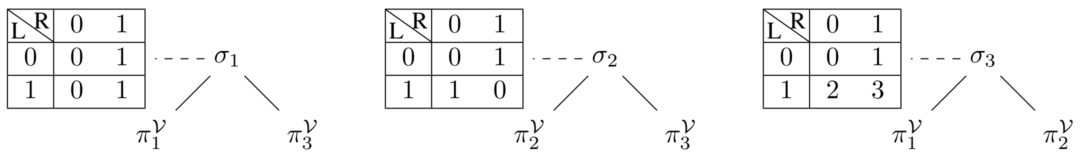
$$\mathit{apply}(T, \mathit{Atomic}(v, \sigma^{\text{tab}})) = \mathit{Atomic}(v, T \circ \sigma^{\text{tab}}) \tag{5}$$

Equations 1–5 describe how to compose FMs. We provide an example in Figure 6 below, after explaining the equations in detail. The first equation describes that two F2F mappings Γ and Σ can be composed by composing each component of Γ individually with Σ. This leaves the problem of composing an F2N mapping γ with an F2F mapping Σ, which is described in Equation 2 for the case where γ is a merge and in Equation 3 for the case where γ is atomic. If γ is a merge (Equation 2), we simply recurse over the components. If γ is atomic (Equation 3), we must select the component of Σ that corresponds to the variable of γ and then apply the table function of γ to the value computed by this component. We write $\mathit{apply}(T, \sigma)$ for the FM which first computes the same value as σ and then applies the function T to this result. (Formally, $[\![\mathit{apply}(T, \sigma)]\!](\alpha) = T([\![\sigma]\!](\alpha))$ for all assignments α.) Equations 4 and 5 show how $\mathit{apply}(T, \sigma)$ can be computed, with Equation 4 covering the case where σ is a merge and Equation 5 covering the case where it is atomic. In both cases, the key operation is to combine the table T with the table of σ, which is the normal function composition $T \circ T'$, where in this case T is a partial function from values to values and T' is either a partial function from pairs of values to values (in Equation 4) or from values to values (in Equation 5). Because T and T' are simply represented as tables, the combined table can be computed in linear time in the number of table entries of T by computing $T(d)$ for each table entry d of T'.

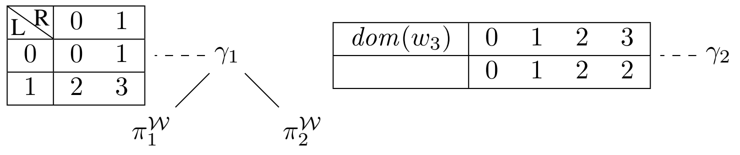
It is easy to verify from the semantics of FMs that Equations 1–5 indeed describe the composition of two FMs. The equations give rise to an obvious recursive algorithm for computing the composition. To bound the runtime of this algorithm, we make the following observations. Firstly, Equations 3–5 can be processed in linear time in the size of their inputs. Secondly, the combined effect of Equations 1 and 2 is to "push" the composition with Σ into each leaf of all component FMs of Γ, at which point we apply Equations 3–5 to each leaf. Together with the first observation this implies that we can compute $\Gamma \circ \Sigma$ in time $O(\|\Gamma\| \cdot \|\Sigma\|)$, where $\|\cdot\|$ denotes the representation size of an FM, which is defined as the total number of table entries in all component FMs (Helmert et al., 2015).

If Γ is an orthogonal F2F mapping, the runtime bound can be tightened: in this case, the right-hand side of Equation 3 selects each F2N mapping σ_j of Σ at most once in the overall computation, and therefore the runtime for computing $\Gamma \circ \Sigma$ can be bounded by $O(\|\Gamma\| + \|\Sigma\|)$, i.e., is linear in the size of the inputs. Moreover, it is easy to see that $\Gamma \circ \Sigma$ is orthogonal if Γ and Σ are orthogonal. Of course, from the runtime bounds we can also derive identical size bounds for $\Gamma \circ \Sigma$ because an output of size N cannot be computed in fewer than N computation steps.

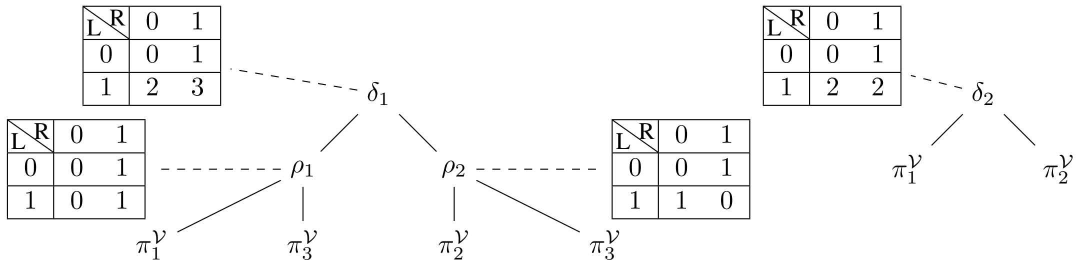
We conclude this discussion with an example. Let $\mathcal{V} = \langle v_1, v_2, v_3 \rangle$ be a variable space where each variable has domain $\{0, 1\}$. Let $\mathcal{W} = \langle w_1, w_2, w_3 \rangle$ be a variable space with $\mathit{dom}(w_1) =$



(a) Σ: F2F mapping from $\mathcal{V}$ to $\mathcal{W}$.



(b) Γ: F2F mapping from $\mathcal{W}$ to $\mathcal{U}$.



(c) $\Gamma \circ \Sigma$: F2F mapping from $\mathcal{V}$ to $\mathcal{U}$.

Figure 6: Composing F2F mappings.

$dom(w_2) = \{0, 1\}$ and $dom(w_3) = \{0, 1, 2, 3\}$. Finally, let $\mathcal{U} = \langle u_1, u_2\rangle$ be a variable space with $dom(u_1) = \{0, 1, 2, 3\}$ and $dom(u_2) = \{0, 1, 2\}$. Consider the example shown in Figure 6. Recall that $\pi_i^{\mathcal{V}}$ denotes the projection FM to the variable with index i of a variable space $\mathcal{V}$. For readability, we omit tables of projection FMs in all illustrations of the figure. For tables of merge FMs, we use L and R to denote their left and right components; for tables of atomic FMs, we give their associated variable.

Part (a) of the figure shows the three component F2N mappings of an F2F mapping $\Sigma = \langle\sigma_1, \sigma_2, \sigma_3\rangle$ from $\mathcal{V}$ to $\mathcal{W}$. All three FMs are merge FMs with two (different) projection FMs as components. Since the variables of the FMs are not disjoint, Σ is an example of a non-orthogonal FM. Part (b) of the figure shows the two component F2N mappings of an F2F mapping $\Gamma = \langle\gamma_1, \gamma_2\rangle$ from $\mathcal{W}$ to $\mathcal{U}$. γ_1 is a merge with components $\pi_1^{\mathcal{W}}$ and $\pi_2^{\mathcal{W}}$, and γ_2 is atomic with variable w_3 and represents an abstraction of the domain of w_3.

Part (c) of the figure shows the composed F2F mapping $\Delta = \Gamma \circ \Sigma$ from $\mathcal{V}$ to $\mathcal{U}$ obtained through Equations 1–5. The FM on the left in the figure, δ_1, is the result of composing γ_1 with Σ. Since γ_1 is a merge FM, Equation 2 applies:

$$\mathit{Merge}(\pi_1^{\mathcal{W}}, \pi_2^{\mathcal{W}}, \gamma_1^{\text{tab}}) \circ \Sigma = \mathit{Merge}(\pi_1^{\mathcal{W}} \circ \Sigma, \pi_2^{\mathcal{W}} \circ \Sigma, \gamma_1^{\text{tab}})$$

We obtain that the resulting FM, δ_1, is a merge FM with the same table function as γ_1, but with the components $\pi_1^{\mathcal{W}}$ and $\pi_2^{\mathcal{W}}$ recursively composed with Σ. Because both $\pi_1^{\mathcal{W}}$ and $\pi_2^{\mathcal{W}}$ are atomic with variables w_1 and w_2, Equation 3 applies to both:

$$\begin{aligned} \mathit{Atomic}(w_1, \mathrm{id}^{w_1}) \circ \langle \sigma_1, \ldots, \sigma_m \rangle &= \mathit{apply}(\mathrm{id}^{w_1}, \sigma_1) \\ \mathit{Atomic}(w_2, \mathrm{id}^{w_2}) \circ \langle \sigma_1, \ldots, \sigma_m \rangle &= \mathit{apply}(\mathrm{id}^{w_2}, \sigma_2) \end{aligned}$$

We get that the two resulting FMs, ρ_1 and ρ_2 (the components of δ_1), are FMs which correspond to the FMs σ_1 and σ_2 of Σ, but with the table functions of $\pi_1^{\mathcal{W}}$ and $\pi_2^{\mathcal{W}}$ combined with their tables according to Equation 4 since σ_1 and σ_2 are merges:

$$\begin{aligned} \mathit{apply}(\mathrm{id}^{w_1}, \mathit{Merge}(\pi_1^{\mathcal{V}}, \pi_3^{\mathcal{V}}, \sigma_1^{\mathrm{tab}})) &= \mathit{Merge}(\pi_1^{\mathcal{V}}, \pi_3^{\mathcal{V}}, \mathit{combine}(\mathrm{id}^{w_1}, \sigma_1^{\mathrm{tab}})) \\ \mathit{apply}(\mathrm{id}^{w_2}, \mathit{Merge}(\pi_2^{\mathcal{V}}, \pi_3^{\mathcal{V}}, \sigma_2^{\mathrm{tab}})) &= \mathit{Merge}(\pi_1^{\mathcal{V}}, \pi_3^{\mathcal{V}}, \mathit{combine}(\mathrm{id}^{w_2}, \sigma_2^{\mathrm{tab}})) \end{aligned}$$

Because the tables of $\pi_1^{\mathcal{W}}$ and $\pi_2^{\mathcal{W}}$ are identity functions, the resulting combined tables are just the tables of σ_1 and σ_2. Note that id^{w_1} is a function defined on $dom(w_1)$ and σ_1 represents a function from $\mathcal{V}$ to w_1, so this is well-defined. We remark that σ_1 and ρ_1 are equivalent in the sense that they represent the same functions, and similarly for σ_2 and ρ_2.

The FM on the right of part (c) of the figure, δ_2, is the result of composing γ_2 with Σ. Since γ_2 is an atomic FM with variable w_3, Equation 3 applies:

$$\mathit{Atomic}(w_3, \gamma_2^{\mathrm{tab}}) \circ \langle \sigma_1, \ldots, \sigma_m \rangle = \mathit{apply}(\gamma_2^{\mathrm{tab}}, \sigma_3)$$

Because σ_3 is a merge, we apply Equation 4:

$$\mathit{apply}(\gamma_2^{\mathrm{tab}}, \mathit{Merge}(\pi_1^{\mathcal{V}}, \pi_2^{\mathcal{V}}, \sigma_3^{\mathrm{tab}})) = \mathit{Merge}(\pi_1^{\mathcal{V}}, \pi_2^{\mathcal{V}}, \mathit{combine}(\gamma_2^{\mathrm{tab}}, \sigma_3^{\mathrm{tab}}))$$

Put together, we obtain that the resulting FM, δ_2, is a merge FM with the table of γ_2 combined with the table of σ_3 and the same components $\pi_1^{\mathcal{V}}$ and $\pi_2^{\mathcal{V}}$ as σ_3.

4.4 Factored Transformations

We are now ready to put the pieces together to define *factored transformations* of factored transition systems. They are a natural generalization of (nonfactored) transformations: they consist of the same components (a transformed transition system, a state mapping, and a label mapping), but use factored transition systems in place of nonfactored ones and F2F mappings as a factored representation of state mappings. Since labels are nonfactored also in factored transition systems, label mappings can be used as in nonfactored transformations.

Definition 21 (Factored Transformation)**.** *A* factored transformation *of a factored transition system F with label set L into a factored transition system F' with label set L' is a tuple $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$, where*

- *F' is called the* transformed factored transition system*,*
- *Σ is an F2F mapping from F to F' called the* state mapping*, and*
- *$\lambda : L \nrightarrow L'$ is called the* label mapping.

The transformation induced by τ_{F} *is a transformation of $\bigotimes F$ into $\bigotimes F'$ which is defined as $\tau = \langle \bigotimes F', [\![\Sigma]\!], \lambda \rangle$.*

$$\begin{tikzcd}
F \arrow[rrr, dashed, "\text{product system}"] \arrow[dd] \arrow[dddd, bend right=40, "\tau'_{\mathrm{F}} \circ \tau_{\mathrm{F}} = \langle F'', \Sigma' \circ \Sigma, \lambda' \circ \lambda \rangle"'] & & & \bigotimes F \arrow[dd] \arrow[dddd, bend left=40, "\tau' \circ \tau = \langle \bigotimes F'', \llbracket \Sigma' \rrbracket \circ \llbracket \Sigma \rrbracket, \lambda' \circ \lambda \rangle"] \\
& \tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle \arrow[r, dashed, bend left, "\text{induced transformation}"] & \tau = \langle \bigotimes F', \llbracket \Sigma \rrbracket, \lambda \rangle & \\
F' \arrow[rrr, dashed, "\text{product system}"] \arrow[dd] & & & \bigotimes F' \arrow[dd] \\
& \tau'_{\mathrm{F}} = \langle F'', \Sigma', \lambda' \rangle \arrow[r, dashed, bend left, "\text{induced transformation}"] & \tau' = \langle \bigotimes F'', \llbracket \Sigma' \rrbracket, \lambda' \rangle & \\
F'' \arrow[rrr, dashed, "\text{product system}"] & & & \bigotimes F''
\end{tikzcd}$$

Figure 7: Composition of transformations, composition of factored transformations, and the relationship between (compositions of) nonfactored and factored transformations.

In the following, we may refer to factored transformations as transformations where this does not lead to ambiguity. Like nonfactored transformations, factored transformations can be composed naturally, since the only nontrivial part of such a composition is the composition of the F2F mappings, which we defined in the previous subsection.

To summarize the concepts of factored and nonfactored transformations, Figure 7 illustrates the relationship between nonfactored and factored transformations, and the compositions of both. It shows factored transformations and their compositions on the left and the induced transformations and their compositions on the right. We conclude that for the final result, it does not matter whether we work on a factored representation or a nonfactored representation of a transition system; in the end, we can always obtain a transformation of the original transition system (or the product of the original factored transition system) into the transformed transition system.

4.4.1 A Factored Transformation Framework

We now present a generic framework to compute arbitrary factored transformations of a given factored transition system through repeated composition of "basic" factored transformations. Later, we will discuss how the merge-and-shrink framework can be understood as a particular instantiation of this generic framework.

Algorithm 1 shows pseudocode for the framework. The main invariant of the algorithm is that $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$ is a factored transformation of the input factored transition system F into the current factored transition system F'. This invariant is first established in lines 2–4, which initializes τ_{F} to $\langle F, \mathrm{id}^F, \mathrm{id}^L \rangle$, where L is the label set of F. Hence, τ_{F} initially represents the identity transformation.

The algorithm continues with the main loop (lines 5–10), where in each iteration, it selects a transformation $\langle F'', \Sigma', \lambda' \rangle$ of the current factored transition system F' using the user-specified SELECTTRANSFORMATION function (line 6). It then applies this transformation to F', which means

Algorithm 1 Factored Transformation Framework

Input: Factored transition system F, function SELECTTRANSFORMATION that selects the next basic transformation to apply, function TERMINATE that decides when to terminate.
Output: Factored transformation $\langle F', \Sigma, \lambda \rangle$ of F into F'.

```
 1: function FACTOREDTRANSFORMATIONFRAMEWORK(
                $F$, SELECTTRANSFORMATION, TERMINATE)
        ▷ Set the current transformation $\tau_F = \langle F', \Sigma, \lambda \rangle$ to the identity tranformation of $F$.
 2:     $F' \leftarrow F$
 3:     $\Sigma \leftarrow \mathrm{id}^F$
 4:     $\lambda \leftarrow \mathrm{id}^L$, where $L$ is the set of labels of $F$
 5:     while not TERMINATE($\langle F', \Sigma, \lambda \rangle$) do
 6:         $\langle F'', \Sigma', \lambda' \rangle \leftarrow$ SELECTTRANSFORMATION($\langle F', \Sigma, \lambda \rangle$)
            ▷ Update the current transformation $\tau_F = \langle F', \Sigma, \lambda \rangle$ to be the composition of $\tau_F$
              with the selected transformation $\langle F'', \Sigma', \lambda' \rangle$.
 7:         $F' \leftarrow F''$
 8:         $\Sigma \leftarrow \Sigma' \circ \Sigma$
 9:         $\lambda \leftarrow \lambda' \circ \lambda$
10:     end while
11:     return $\langle F', \Sigma, \lambda \rangle$
12: end function
```

to compose it with τ_F (lines 7–9), restoring the invariant. The main loop stops if the user-specified TERMINATE criterion triggers (line 5), at which point the algorithm returns τ_F (line 11).

As discussed in the previous section, the resulting factored transformation induces a nonfactored transformation $\tau = \langle \bigotimes F', [\![\Sigma]\!], \lambda \rangle$ of $\bigotimes F$ into $\bigotimes F'$. Our theory of nonfactored transformations shows that τ has all properties, such as conservativeness or inducedness, that are shared by all basic transformations chosen by SELECTTRANSFORMATION (Theorem 1 in Section 3.2). Therefore, we can understand the properties of τ in terms of the basic transformations from which it is composed. Furthermore, we know how these transformation properties relate to properties of heuristics induced by τ, such as admissibility and exactness (Theorems 3 and 5 in Section 3.3). In the following, we study some further properties of the induced heuristics that are specific to *factored* transformations.

4.4.2 LOCAL AND GLOBAL HEURISTICS

There are two ways to derive heuristics from factored transformations $\tau_F = \langle F', \Sigma, \lambda \rangle$ of a factored transition system F. The first way, which we focused on above, is to study the heuristic induced by the *nonfactored* transformation of $\bigotimes F$ induced by τ_F. In the following, we refer to this heuristic as the *global* heuristic of τ_F. The advantage of the global heuristic is that it takes into account all information available in τ_F. The disadvantage is that actually computing the global heuristic requires the computation of the product $\bigotimes F'$, which is often not computationally feasible.

A less expensive alternative is to observe that each factor $\Theta' \in F'$, along with the corresponding component state mapping $\sigma \in \Sigma$, can be used to define a heuristic in its own right. Given a state s of F, we can compute $s' = \sigma(s)$ and then use the perfect heuristic for s' in Θ' as a heuristic value. This heuristic only takes into account the information of one of the components of τ_F, but can be computed efficiently. We will refer to these component-based heuristics as *local* heuristics.

Computing local heuristics involves two conceptually different computational steps: first, we use τ_{F} to map F to F', and then we use a single factor of F' to approximate the "global" goal distance within $\bigotimes F'$.

Definition 22 (Factor Heuristics and Local Heuristics)**.** *Let $F = \langle \Theta_1, \dots, \Theta_n \rangle$ be a factored transition system. The* factor heuristic *for $\Theta_i \in F$ is defined as $h_i(s) = h^*_{\Theta_i}(s[\Theta_i])$ for all $s \in \mathcal{A}(F)$. The* max-factor heuristic of F *is defined as $h^{\mathrm{mf}}_F = \max_{1 \leq i \leq n} h_i$.*

Let $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$ be a factored transformation of a factored transition system F. The local heuristic for F induced by τ_{F} *is defined as $h^{\mathrm{loc},\tau_{\mathrm{F}}}_F(s) = h^{\mathrm{mf}}_{F'}([\![\Sigma]\!](s))$ for all $s \in \mathcal{A}(F)$. (If $s \notin dom([\![\Sigma]\!])$, we define $h^{\mathrm{mf}}_{F'}([\![\Sigma]\!](s)) = \infty$.)*

Note that, rather than defining separate local heuristics for each component of τ_{F}, we only define a single local heuristic as the maximum of these individual heuristics. This is sufficient for our purposes and reduces notational clutter.

The application of the factored transformation framework to heuristic search that we describe in the following ultimately returns the local heuristic of the final transformation as its result. In this context, it is desirable to only apply transformations that do not reduce the quality of the local heuristic. Otherwise, we might unnecessarily end up with a heuristic that is worse than a previously computed intermediate result. To make these observations more formal, we now define properties that describe the effect of a factored transformation on the local heuristic.

Definition 23 (Locally Conservative Factored Transformations)**.** *Let τ_{F} be a factored transformation of a factored transition system F. The following list defines properties that τ_{F} may have, along with a short-hand name for each property (analogously to Definition 8).*

LOC$_{\leq}$ τ_{F} *is* locally nonincreasing *if $h^{\mathrm{loc},\tau_{\mathrm{F}}}_F(s) \leq h^{\mathrm{mf}}_F(s)$ for all states $s \in \mathcal{A}(F)$.*

LOC$_{\geq}$ τ_{F} *is* locally nondecreasing *if $h^{\mathrm{loc},\tau_{\mathrm{F}}}_F(s) \geq h^{\mathrm{mf}}_F(s)$ for all states $s \in \mathcal{A}(F)$.*

LOC$_{=}$ τ_{F} *is* locally equal *if $h^{\mathrm{loc},\tau_{\mathrm{F}}}_F(s) = h^{\mathrm{mf}}_F(s)$ for all states $s \in \mathcal{A}(F)$.*

In other words, **LOC**$_{=}$ $=$ **LOC**$_{\leq}$ $+$ **LOC**$_{\geq}$.

In comparison to the **CONS**, **IND**, and **REF** properties introduced in Section 3, which conjunctively imply that the global heuristic is perfect, these local properties are somewhat less critical. However, they are certainly of interest, as seen by the fact that they have been discussed in the literature before, for example in the context of h-preserving shrink strategies (cf. Section 5).

In the context of the factored transformation framework of Algorithm 1, let τ_{F} be the current factored transformation, and let τ''_{F} be the next factored transformation computed as $\tau''_{\mathrm{F}} = \tau'_{\mathrm{F}} \circ \tau_{\mathrm{F}}$, where τ'_{F} is the basic transformation selected in a given iteration of the main loop. If τ'_{F} is locally nondecreasing, we obtain $h^{\mathrm{loc},\tau_{\mathrm{F}}}_F \leq h^{\mathrm{loc},\tau''_{\mathrm{F}}}_F$, and similarly for the locally nonincreasing and locally equal case. Therefore, the properties of τ'_{F} dictate how the local heuristic evolves as Algorithm 1 proceeds.

When dealing with transformations that give rise to admissible heuristics, it is therefore desirable for τ'_{F} to be locally nondecreasing, as this avoids a loss of heuristic information. If τ'_{F} is locally equal, the local heuristic does not change at all from one iteration to the next. Therefore, from the perspective of the local heuristic, the most desirable basic transformations are those that are

Algorithm 2 Merge-and-shrink Heuristics

```
Input: Planning task $\Pi$, function SELECTTRANSFORMATION that selects the next basic transformation to apply, function TERMINATE that determines when to stop applying transformations.
Output: Merge-and-shrink heuristic $h^{\text{M\&S}}_{\Pi}$ for $\Pi$.
1: function MERGEANDSHRINK($\Pi$, SELECTTRANSFORMATION, TERMINATE)
       ▷ Compute the induced factored transition system of $\Pi$.
2:     $F \leftarrow F(\Pi)$
       ▷ Call the factored transformation framework.
3:     $\langle F', \Sigma, \lambda \rangle \leftarrow$ FACTOREDTRANSFORMATIONFRAMEWORK(
                   $F$, SELECTTRANSFORMATION, TERMINATE)
       ▷ Compute the merge-and-shrink heuristic, where $\tau_{\text{F}} = \langle F', \Sigma, \lambda \rangle$.
4:     $h^{\text{M\&S}}_{\Pi} \leftarrow h^{\text{loc},\tau_{\text{F}}}_{F}$
5:     return $h^{\text{M\&S}}_{\Pi}$
6: end function
```

locally nondecreasing, but *not* locally equal, which implies that the new local heuristic dominates the previous one.

We conclude the discussion of local heuristics by showing that the properties affecting the local heuristic compose.

Theorem 6. *Let X be any of the properties of transformations from Definition 23. Let τ_{F} be a factored transformation of F into F' with property X, and let τ'_{F} be a factored transformation of F' into F'' with property X. Then $\tau'_{\text{F}} \circ \tau_{\text{F}}$ has the property X.*

The proof can be found in Appendix B.

4.4.3 MERGE-AND-SHRINK HEURISTICS

The factored transformation framework of Algorithm 1 is also called the *merge-and-shrink framework*, named after the two transformations that were introduced first (Dräger, Finkbeiner, & Podelski, 2006), although not under these names. Its most common use is to derive heuristics for planning tasks, which are consequently called *merge-and-shrink heuristics*. Algorithm 2 shows pseudocode for the computation of merge-and-shrink heuristics.

The function MERGEANDSHRINK takes a planning task Π as its input (line 1). It first computes the factored transition system induced by Π (line 2) and then uses the factored transformation framework to compute a factored transformation of $F(\Pi)$. This factored transition system serves as the basis for computing the merge-and-shrink heuristic for Π, which is defined as the local heuristic induced by the factored transformation from $F(\Pi)$ to F'. Formally, the factored transformation $\tau_{\text{F}} = \langle F', \Sigma, \lambda \rangle$ induces the merge-and-shrink heuristic $h^{\text{M\&S}}_{\Pi} = h^{\text{loc},\tau_{\text{F}}}_{F}$.

From the definition of local heuristics, computing $h^{\text{M\&S}}_{\Pi}$ involves three steps. Firstly, $[\![\Sigma]\!]$ maps the given state s of $F(\Pi)$ to a state s' of F'. This step is represented by the F2F mapping Σ. Secondly, the factor heuristics map the components of s' to a tuple of heuristic values, one for each factor of F'. This can also be represented by an F2F mapping, where each contained F2N mapping is atomic, mapping states of one factor to their goal distances. Finally, the resulting heuristic value is the maximum of this tuple. Because the first two steps involve two F2F mappings applied in sequence, they can be combined into a single F2F mapping representing the composition of $[\![\Sigma]\!]$

and the factor heuristics (Section 4.3.3). In the common case where F' consists of a single factor, the third step trivializes and the merge-and-shrink heuristic can be represented as a single F2N mapping.

4.4.4 MERGE-AND-SHRINK STRATEGIES

Both the factored transformation framework (Algorithm 1) and its use for the computation of heuristics (Algorithm 2) leave two important choice points unspecified, namely when to terminate FACTOREDTRANSFORMATIONFRAMEWORK (function TERMINATE) and how to select the basic transformations (function SELECTTRANSFORMATION).

To resolve the latter, the algorithm must make two kinds of decisions. Firstly, it must decide which general type of the available four types of basic transformation to use in each iteration: *shrink* transformations apply abstractions to one of the factors of F'; *merge* transformations combine two factors of F' into one, reducing the size of F' by 1; *label reductions* map the label set of F' to a smaller set; and *prune* transformations discard (a subset of the) states. We describe these four types of transformations in detail in Sections 5–8. Secondly, it must decide how to instantiate the given transformation type (i.e., how to shrink, merge, etc.).

We refer to the first kind of decisions as the *general strategy* and to the second kind of decisions as *transformation strategies*, which are subdivided into *shrink strategies*, *merge strategies*, *label reduction strategies*, and *pruning strategies*. As the literature on merge-and-shrink heuristic shows, there are many possible ways of defining these strategies (e.g., Sievers, 2017). We will discuss the most important aspects of the transformation strategies in Sections 5–8, with an emphasis on how the transformation strategy affects the properties (such as conservativeness, exactness etc.) of the resulting transformation.

For merge-and-shrink heuristics, a natural choice for SELECTTRANSFORMATION is to only use exact transformations. A natural choice for TERMINATE is to continue applying transformations until only a single factor remains. The combination of these two ideas guarantees that the resulting heuristic is perfect. However, common planning benchmarks are so large that reduction to a single factor while only using exact transformations is usually infeasible (but cf. Nissim et al., 2011). Practical general strategies therefore sacrifice both desiderata when necessary. In particular, they impose size limits on intermediate factors, applying non-exact shrink transformations when necessary, and they use a time limit to terminate early when the merge-and-shrink computation takes too long. For a more detailed discussion of these and other aspects of general strategies, we refer to Sievers (2018).

Each of the following four sections studies one of the four merge-and-shrink transformation types in detail. All sections are structured in a similar way. Firstly, we define the factored transformations in terms of their transformed factored transition systems, state mappings, and label mappings. Secondly, we discuss aspects of efficient implementation. In particular, we describe special-purpose in-place implementations of the composition step of Algorithm 1 (lines 7–9) for the specific transformation type. Finally, we investigate the formal properties of the transformations.

5. Shrink Transformation

In this section, we define the shrink transformation of the merge-and-shrink framework. Proofs of theorems can be found in Appendix C.

For a transition system $\Theta = \langle S, L, c, T, S_\text{I}, S_\text{G} \rangle$ and a (total) state mapping α defined on S, the *transition system induced by* Θ *and* α is defined as $\Theta^\alpha = \langle \alpha(S), L, c, \{\langle \alpha(s), \ell, \alpha(t) \rangle \mid \langle s, \ell, t \rangle \in T\}, \alpha(S_\text{I}), \alpha(S_\text{G})\} \rangle$. When Θ is a factor of some factored transition system, we also call α a *local abstraction*.[8]

Definition 24 (Shrink Transformation)**.** *Let* $F = \langle \Theta_1, \dots, \Theta_n \rangle$ *be a factored transition system. Let* α *be a local abstraction for* Θ_k *for some* $1 \leq k \leq n$*. The* shrink transformation *for* α *and* Θ_k *is the factored transformation* $\tau_\text{F} = \langle F', \Sigma, \lambda \rangle$ *of* F *where:*

- $F' = \langle \Theta'_1, \dots, \Theta'_n \rangle$ *with* $\Theta'_i = \Theta_i$ *for all* $i \neq k$ *and* $\Theta'_k = \Theta^\alpha_k$.
- $\Sigma = \langle \sigma_1, \dots, \sigma_n \rangle$ *where* $\sigma_i = \pi_i$ *for all* $i \neq k$*, and* σ_k *is an atomic FM with variable* Θ_k *and* $\sigma^\text{tab}_k(s_k) = \alpha(s_k)$ *for all* $s_k \in \Theta_k$.
- $\lambda = \text{id}$ *is the identity label mapping.*

In words, a shrink transformation maps the given factored transition system to a transformed system that is exactly the same, except that one factor Θ is replaced by Θ^α, where α is a local abstraction.

We call such a transformation *shrinking* because we usually choose a local abstraction α that maps the states of Θ to some smaller set, hence reducing the number of states in the transformed (factored) transition system. Shrinking offers a controlled way of reducing the size of the factors of a factored transition system. The details of how the local abstraction α reduces the states of a factor determines the qualitative loss of information, hence offering a trade-off between heuristic quality and size of individual factors.

5.1 Efficient Computation

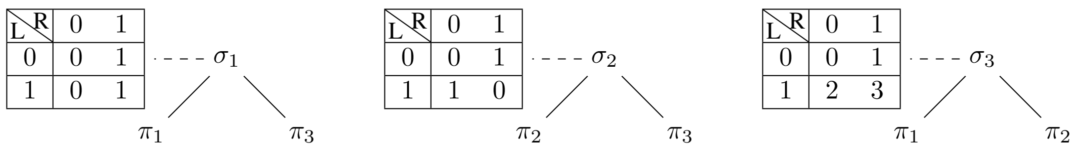
Recall that a shrink transformation will always be composed with any previous transformation within the factored transformation framework (Algorithm 1, lines 6–9). We discussed how to compose FMs in general in Section 4.3 (Equations 1–5). In the following, we briefly discuss how this can be done more efficiently as an in-place modification of the previous transformation. This is a purely computational optimization.

It is easy to see that all aspects of the previous transformation other than the modified factor and the corresponding FM are unchanged and thus can be reused. The local abstraction α of factor Θ_k can be used for an in-place modification of Θ_k and its F2N mapping σ_k: states of the factor that are mapped to the same abstract state by α are collapsed and their transitions are combined, which can be done efficiently with appropriate data structures. In the implementation of merge-and-shrink heuristics in the Fast Downward planner (Helmert, 2006), all transitions of a transition system are stored in a single list grouped by labels. With this representation, it suffices to replace each transition $s \to t$ by $\alpha(s) \to \alpha(t)$ and remove the duplicates in the resulting list. The F2N mapping σ_k can also be updated in-place by simply applying the state mapping α to the table of its root FM.

5.2 Example

Figure 8 shows an example of how a shrink transformation affects a given F2F mapping. Part (a) shows the F2F mapping Σ before shrinking. Part (b) shows the F2F mapping Γ that represents the

8. Until now we used α to denote assignments. In the following we use α to denote abstractions. Both are established notations and we think that it is clear from context if α stands for an assignment or an abstraction.



L \ R	0	1
0	0	1
1	0	1

L \ R	0	1
0	0	1
1	1	0

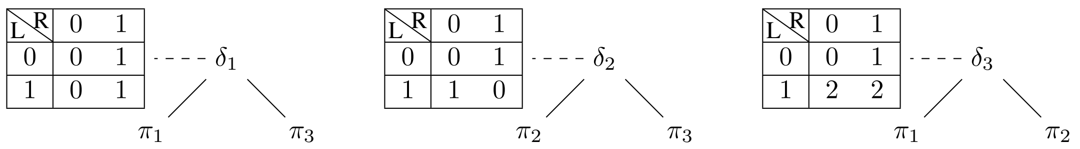
L \ R	0	1
0	0	1
1	2	3

(a) $\Sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$: a given F2F mapping from $\mathcal{V}$ to $\mathcal{W}$.

$\gamma_1 = \pi_1$ $\gamma_2 = \pi_2$

$dom(w_3)$	0	1	2	3
	0	1	2	2

(b) $\Gamma = \langle \gamma_1, \gamma_2, \gamma_3 \rangle$: F2F mapping from $\mathcal{W}$ to $\mathcal{U}$ for a shrink transformation.



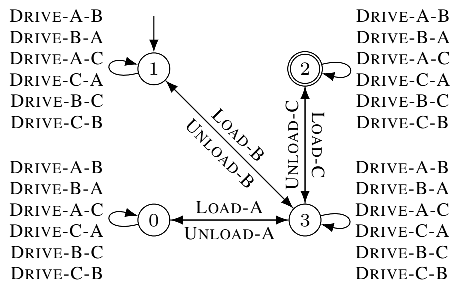
L \ R	0	1
0	0	1
1	0	1

L \ R	0	1
0	0	1
1	1	0

L \ R	0	1
0	0	1
1	2	2

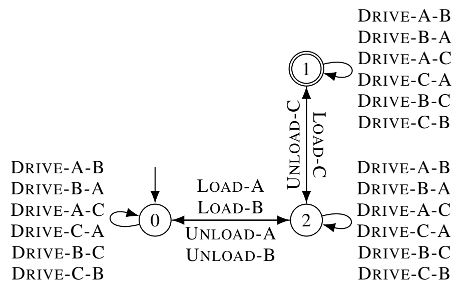
(c) $\Gamma \circ \Sigma = \langle \delta_1, \delta_2, \delta_3 \rangle$: composition F2F mapping from $\mathcal{V}$ to $\mathcal{U}$.

Figure 8: Composing an F2F mapping for a shrink transformation with a previous transformation.



σ_P

$dom(v_P)$	A	B	C	T
	0	1	2	3



σ'_P

$dom(v_P)$	A	B	C	T
	0	0	1	2

(a) Atomic factor Θ^P (top) and F2N mapping σ_P (bottom).

(b) Transformed factor (top) and transformed F2N mapping (bottom).

Figure 9: Example of a shrink transformation based on a local abstraction α for the factor Θ^P of the induced factored transition system of the example planning task of Figure 4, where α maps states 0 and 1 to state 0 and renames states 2 to 1 and 3 to 2.

shrink step. We see that the first two components are trivial projections because the shrink transformation only affects the third component. The third component illustrates the local abstraction α applied to the third component with $\alpha(0) = 0$, $\alpha(1) = 1$ and $\alpha(2) = \alpha(3) = 2$. Part (c) shows the resulting F2F mapping $\Gamma \circ \Sigma$. We see that $\Gamma \circ \Sigma$ is very similar to the original F2F mapping Σ, with the only difference that α has been applied to each entry of the table at the root of the shrunk factor. Hence, while $\Gamma \circ \Sigma$ can in principle be computed with the generic algorithm for composing F2F mappings described in Section 4.3.3, we can more efficiently compute it by modifying Σ in place.

To illustrate the effect of shrinking on transition systems, consider the factored transition system induced by the example planning task of Figure 4. Figure 9(a) shows the atomic factor Θ^P (top) and the F2N mapping σ_P representing the identity state mapping (bottom). An example shrink transformation τ based on a local abstraction α for the atomic factor Θ^P combines states 0 and 1 (which correspond to values A and B of variable v_P). In Figure 9(b), we see the shrunk transition system $\tau(\Theta^P)$ (top) and the transformed F2N mapping $\tau(\sigma_P)$ (bottom). The former can be obtained from the original factor by collapsing states 0 and 1 into a new state 0, and renaming state 2 to 1 and state 3 to 2 (for continuous numbering). The transitions of the original states 0 and 1 are combined at the new state 0, and everything else is unchanged. The transformed F2N mapping maps both values A and B to the new state 0 of the shrunk transition system, and maps C to 1 and T to 2. It can be obtained by applying α to the table of σ_P.

5.3 Properties

We now investigate the formal properties of shrink transformations.

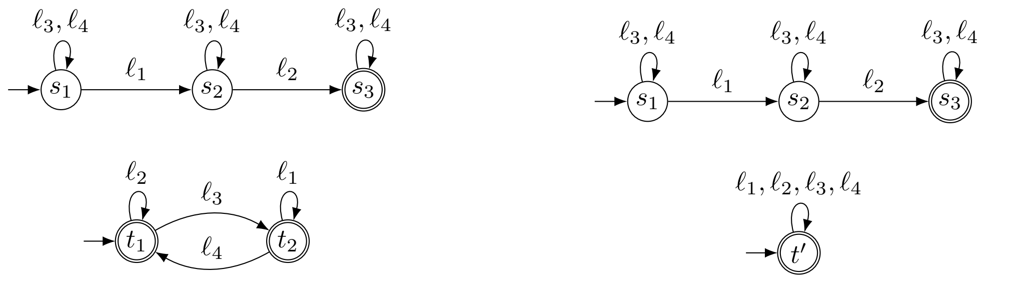
Theorem 7. *All shrink transformations are induced abstractions, i.e., they satisfy* **CONS** + **IND**. *They are also cost-refinable, i.e., satisfy* $\mathbf{REF_C}$*, and locally nonincreasing, i.e., satisfy* $\mathbf{LOC_{\leq}}$*.*

Because shrink transformations are (induced) abstractions, heuristics based on them are admissible and consistent (Theorem 3). The theorem also states that shrink transformations preserve label costs and cannot increase maximum local heuristic values. This means that a shrink transformation cannot lead to an immediate improvement of heuristic values within the merge-and-shrink framework. Rather, the motivation for shrinking is to reduce the representation size of the factored transition system.

Besides the properties mentioned in Theorem 7, we introduced four further transformation properties in Definitions 8 and 23: transition-refinable ($\mathbf{REF_T}$), goal-state-refinable ($\mathbf{REF_G}$), locally nondecreasing ($\mathbf{LOC_{\geq}}$) and locally equal ($\mathbf{LOC_{=}}$). It is easy to see that shrink transformations guarantee none of these properties in general. For example, regarding $\mathbf{LOC_{\geq}}$, shrink transformations can be arbitrarily bad: in the extreme case, if there is only one factor (with at least one goal state) and we abstract all its states into the same state, we end up with the zero heuristic. In the following, we give a sufficient criterion for shrink transformations to be $\mathbf{LOC_{\geq}}$ (and hence also $\mathbf{LOC_{=}}$, because Theorem 7 guarantees $\mathbf{LOC_{\leq}}$) as well as exact characterizations for $\mathbf{REF_T}$ and $\mathbf{REF_G}$.

Theorem 8. *Consider a shrink transformation τ_F based on a local abstraction α of factor Θ such that $\alpha(s_1) = \alpha(s_2)$ implies $h^*_\Theta(s_1) = h^*_\Theta(s_2)$ for all states $s_1, s_2 \in \Theta$. Then τ_F is locally nondecreasing (*$\mathbf{LOC_{\geq}}$*) and hence locally exact (*$\mathbf{LOC_{=}}$*).*

Shrink transformations that satisfy the requirements of Theorem 8, i.e., only combine states with the same (local) goal distance, are called *h-preserving* in the literature (Helmert et al., 2007). As discussed in the previous section, transformations without this property can result in heuristics that are worse than the intermediate local heuristics that were already previously available during the merge-and-shrink construction, which is highly undesirable. As a consequence, non-h-preserving shrinking has only been considered in the literature in cases where it is absolutely necessary, i.e., when the general merge-and-shrink strategy needs to enforce a size limit on abstract transition systems that is smaller than the number of distinct h-values in the local abstraction (Helmert et al., 2007). Note, however, that the condition of Theorem 8 is sufficient but not necessary for local exactness: because the local heuristic is based on the *maximum* of the heuristics provided by the individual factors, it is possible that the loss in heuristic information for one factor does not affect the overall local heuristic value.



(a) Factored transition system F. (b) Shrunk factored transition system F'.

Figure 10: h-preserving shrink transformation based on local abstraction α: part (a) shows a factored transition system F with factors Θ_1 (top) and Θ_2 (bottom), and part (b) shows the shrunk factored transition system F' with factors $\Theta'_1 = \Theta_1$ (top) and Θ'_2 the transition system induced by Θ_2 and α (bottom).

Even locally exact shrink transformations are in general not refinable and can hence lead to loss in accuracy for the *global* heuristic. For example, consider the transformation shown in Figure 10: part (a) shows the original factored transition system, and part (b) shows a transformed factored transition system under an h-preserving shrink transformation. Despite the local equivalence of the factor heuristics, the global heuristic deteriorates under this transformation. For example, assuming all labels to have cost 1, we have $h^*_F(\langle s_1, t_1\rangle) = 4$ (from the plan $\langle \ell_3, \ell_1, \ell_4, \ell_2\rangle$), but $h^*_{F'}(\langle s_1, t'\rangle) = 2$ for the corresponding abstract state (from the plan $\langle \ell_1, \ell_2\rangle$).

The reason why the heuristic can get worse is that the transformation is not transition-refinable. For example, consider the transition $\langle s_1, t'\rangle \xrightarrow{\ell_1} \langle s_2, t'\rangle$ of $\bigotimes F'$. This transition can not be refined for *all* preimages $\langle s_1, t_1\rangle, \langle s_1, t_2\rangle$ of $\langle s_1, t'\rangle$ because there is no transition labeled with ℓ_1 from $\langle s_1, t_1\rangle$ in $\bigotimes F$. Hence, the transformation is not **REF$_\mathbf{T}$**. (It is, however, goal-state-refinable. It is easy to see that h-preserving shrink transformations must be goal-state-refinable if all labels have nonzero cost.)

Because **REF$_\mathbf{T}$** and **REF$_\mathbf{G}$** are the only properties missing from Theorem 7 for shrink transformations to be exact (perfectly preserving the global heuristic), shrink transformations with these properties are of particular interest. We will now show that these refinability properties are related to the concept of *bisimulation* (Milner, 1980).

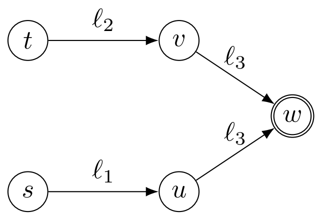
Definition 25 (Bisimulation)**.** *Let $\Theta = \langle S, L, c, T, S_\mathrm{I}, S_\mathrm{G}\rangle$ be a transition system. An equivalence relation $\sim$ on S can have the following two properties:*

BISIM1 *$s_1 \sim s_2$ implies that either $s_1, s_2 \in S_\mathrm{G}$ or $s_1, s_2 \notin S_\mathrm{G}$.*

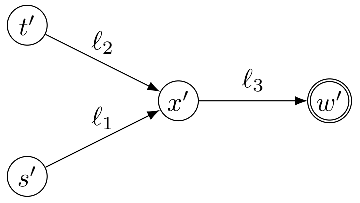
BISIM2 *for all pairs of states $s_1, s_2 \in S$ with $s_1 \sim s_2$ and all $s_1 \xrightarrow{\ell} t_1 \in T$, there exists a state $t_2 \in S$ with $s_2 \xrightarrow{\ell} t_2 \in T$ and $t_1 \sim t_2$.*

We call $\sim$ a bisimulation *for Θ if it satisfies both* **BISIM1** *and* **BISIM2**.

For every transition system, there exists a unique *coarsest* bisimulation, i.e., a bisimulation $\sim$ that satisfies $s \sim t$ whenever $s \sim' t$ for *any* bisimulation relation $\sim'$ (Milner, 1980). We call states s and t *bisimilar* if $s \sim t$ under the coarsest bisimulation.



(a) Transition system Θ with bisimilar states u and v.



(b) Transition system Θ^{α} induced by Θ and α.

Figure 11: Transition system Θ and transition system Θ^{α} induced by local abstraction α based on bisimulation which maps states u and v to x'.

Intuitively, states are bisimilar if they are both goal states or both non-goal states, and outgoing transitions labeled with the same label lead to bisimilar states. Figure 11(a) shows an example transition system Θ where states u and v are bisimilar, i.e., $u \sim v$, because both have outgoing transitions labeled with the same label (ℓ_3) that lead to bisimilar states (here: the same state w). On the other hand, s and t are not bisimilar ($s \not\sim t$) because their outgoing transitions, although leading to bisimilar states, are labeled with different labels.

Equivalence relations $\sim$ over a set of states S (whether they are bisimulations or not) naturally induce an abstraction mapping $\alpha_{\sim}$ on S, where every state s is mapped to its equivalence class. Under such a mapping, we have $\alpha_{\sim}(s_1) = \alpha_{\sim}(s_2)$ iff $s_1 \sim s_2$. Nissim et al. (2011) introduced shrink transformations based on bisimulation, i.e., where the local abstraction combines states that are bisimilar. They proved that shrinking based on bisimulation is “information-preserving” in the sense that merge-and-shrink heuristics where all shrink transformations are based on bisimulation are perfect. The following result provides a more fine-grained view of this result within our transformation framework.

Theorem 9. *Let τ_{F} be a shrink transformation based on the local abstraction α of factor Θ where α is induced by the equivalence relation $\sim$ on the states of Θ. Then:*

1. *If $\sim$ has the property* **BISIM1***, then τ_{F} has the property* $\mathbf{REF_G}$.
2. *If $\sim$ has the property* **BISIM2***, then τ_{F} has the property* $\mathbf{REF_T}$.
3. *If $\sim$ is a bisimulation, then τ_{F} is exact induced (***CONS** + **IND** + **REF***) and locally equal (*$\mathbf{LOC_=}$*).*

Consider again the transition system Θ shown in Figure 11(a). Figure 11(b) shows the transition system Θ^{α} induced by Θ and the local abstraction α based on bisimulation. It is not difficult to verify that the states and transitions of the induced transition system can be refined to the corresponding states and transitions of the original transition system Θ. In particular, for the transition $x' \xrightarrow{\ell_3} w'$, we have that for *all* preimages v, u of x', there exist transitions labeled with ℓ_3 that lead to w, a preimage of w': both $v \xrightarrow{\ell_3} w$ and $u \xrightarrow{\ell_3} w$ are transitions of Θ. The example also demonstrates that bisimilar states have the same perfect heuristic value.

To complete our discussion of the properties of shrink transformations, we now show that, bisimulation provides not just a sufficient criterion, but a *characterization* of exact shrink transformations. If a shrink transformation is $\mathbf{REF_G}$ and $\mathbf{REF_T}$, it *must* be based on a bisimulation (not necessarily the coarsest one). This claim comes with a small technical caveat: it is not true for factored transition systems that are *trivially unsolvable* or that contain *dead labels* in the sense of the following definition. However, we will see that these restrictions are not limiting.

Definition 26 (Trivially Unsolvable Transition Systems, Dead Labels)**.** *A transition system Θ is* trivially unsolvable *if Θ has no goal states. A factored transition system F is* trivially unsolvable *if $\bigotimes F$ is trivially unsolvable.*

A label ℓ is dead *in a transition system Θ if Θ has no transition with the label ℓ. A label ℓ of a factored transition system F is dead if it is dead in $\bigotimes F$.*

From the definition of $\bigotimes F$, it is easy to see that F is trivially unsolvable iff at least one of its factors is trivially unsolvable and that ℓ is dead in F iff it is dead in at least one of its factors. We can now state the converse result of Theorem 9.

Theorem 10. *Let τ_F be a shrink transformation of a factored transition system F based on the local abstraction α of factor Θ, where α is induced by the equivalence relation $\sim$ on the states of Θ. Then:*

1. *If τ_F has the property* $\mathbf{REF_G}$ *and F is not trivially unsolvable, then $\sim$ has the property* **BISIM1**.
2. *If τ_F has the property* $\mathbf{REF_T}$ *and F has no dead labels, then $\sim$ has the property* **BISIM2**.
3. *If τ_F is exact, F is not trivially unsolvable and F has no dead labels, then $\sim$ is a bisimulation.*

The restriction to factored transition systems that are not trivially unsolvable in the first part of the theorem is necessary because in a trivially unsolvable transition system, all transformations are (trivially) goal-state-refinable due to the absence of goal states, but of course not all equivalence relations have the property **BISIM1**. The restriction is no real limitation: as the name suggests, trivially unsolvable transition systems never have solutions, and this is easily and efficiently detectable within the merge-and-shrink framework (for example, by shrinking each factor to a single state).

Similarly, dead labels must be ruled out for the second part of the theorem: if ℓ is dead, all transitions labeled with ℓ are refinable (because there are none), but ℓ might still cause **BISIM2** to be violated. Again, this restriction is not a limitation: if ℓ is a dead label of F, we can remove all transitions with label ℓ from all factors of F without changing $\bigotimes F$ (which means that this is an exact transformation). After this transformation, because ℓ is now dead in *all* factors, it no longer affects the bisimulation properties.

We remark that the previous literature on merge-and-shrink abstractions contains no equivalent result to Theorem 10. We believe that this is largely because it is difficult to formulate a result like Theorem 10 without the language of transformation properties introduced in this paper. Indeed, being able to precisely and concisely state results like Theorems 7–10 is one of the major motivations for introducing the transformation framework in this paper.

We conclude this section by summarizing our results on the properties of shrinking.

Theorem 11. *Let τ_F be a shrink transformation of a factored transition system that is not trivially unsolvable and has no dead labels. Let $\sim$ be the equivalence relation on which the local abstraction of τ_F is based. Then:*

- τ_{F} *satisfies* **CONS** + **IND** + **REF$_{\mathbf{C}}$** + **LOC$_{\leq}$**.
- *If* τ_{F} *is* h*-preserving, then it additionally satisfies* **LOC$_{\geq}$** *(and hence also* **LOC$_{=}$***).*
- τ_{F} *satisfies* **REF$_{\mathbf{G}}$** *iff* $\sim$ *satisfies* **BISIM1**.
- τ_{F} *satisfies* **REF$_{\mathbf{T}}$** *iff* $\sim$ *satisfies* **BISIM2**.
- τ_{F} *is exact iff* $\sim$ *is a bisimulation.*
- *If* $\sim$ *is a bisimulation, then* τ_{F} *is* h*-preserving.*

6. Merge Transformation

In this section, we discuss merge transformations, which together with shrink transformations defined in the previous section form the backbone of the merge-and-shrink framework. Proofs of theorems can be found in Appendix D.

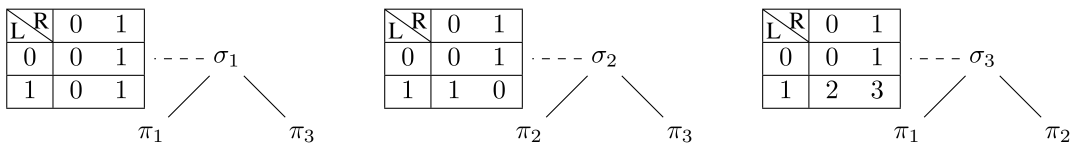
Formally defining merge transformations requires defining subtuples of a tuple. To facilitate the notation of such subtuples, given a set X of indices, we write *indices*(X) for the tuple that contains all elements of X exactly once and in ascending order. For example, we have *indices*$(\{1, \dots, 6\} \setminus \{2, 4\})$ = *indices*$(\{1, 3, 5, 6\}) = \langle 1, 3, 5, 6 \rangle$.

Definition 27 (Merge Transformation)**.** *Let* $F = \langle \Theta_1, \dots, \Theta_n \rangle$ *be a factored transition system. The* merge transformation *for* $j \neq k$ *with* $1 \leq j, k \leq n$ *is the factored transformation* $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$ *of* F *into the factored transition system* F' *where:*

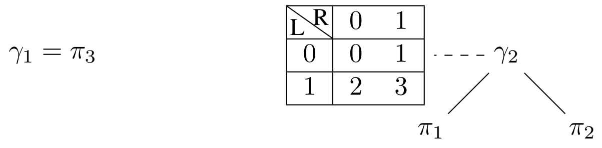
- $F' = \langle \Theta_{i_1}, \dots, \Theta_{i_{n-2}}, \Theta_{\otimes} \rangle$ *with* $i_1, \dots, i_{n-2}$ = *indices*$(\{1, \dots, n\} \setminus \{j, k\})$ *and* $\Theta_{\otimes} = \Theta_j \otimes \Theta_k$.
- $\Sigma = \langle \pi_{i_1}, \dots, \pi_{i_{n-2}}, \sigma_{\otimes} \rangle$ *with* $i_1, \dots, i_{n-2}$ = *indices*$(\{1, \dots, n\} \setminus \{j, k\})$*, where* $\sigma_{\otimes}$ *is a merge FM with left component* π_j*, right component* π_k*, and* $\sigma_{\otimes}^{\text{tab}}(s_j, s_k) = \langle s_j, s_k \rangle$ *for all states* $s_j \in \Theta_j$ *and* $s_k \in \Theta_k$.
- $\lambda = \mathrm{id}$ *is the identity label mapping.*

The merge transformation replaces two factors of the given factored transition system by their product system, leaves all other factors unchanged, and does not change the set of labels. The state mapping consists of projection FMs for all unchanged factors (representing the identity function for these factors) and of a merge FM that maps pairs of states of the replaced factors to their corresponding product state.

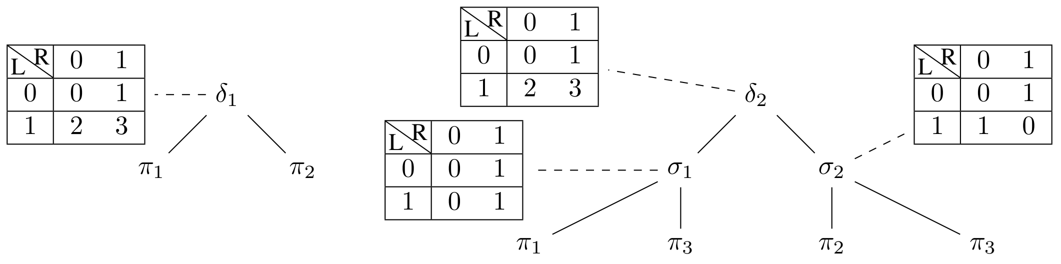
It is easy to see that if we apply a sequence of merge and shrink transformations, each atomic factor contributes to exactly one factor in the resulting factored transition system. Alternatives that allow atomic factors to contribute to multiple resulting factors are discussed in the literature under the name *non-orthogonal merging* (Helmert et al., 2007, 2014), which is related to the concept of *fluent merging* (van den Briel, Benton, Kambhampati, & Vossen, 2007a; van den Briel, Kambhampati, & Vossen, 2007b). We discuss this further in Section 9.2.6.



(a) $\Sigma = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$: a given F2F mapping from $\mathcal{V}$ to $\mathcal{W}$.



(b) $\Gamma = \langle \gamma_1, \gamma_2 \rangle$: F2F mapping from $\mathcal{W}$ to $\mathcal{U}$ for a merge transformation merging σ_1 and σ_2.



(c) $\Gamma \circ \Sigma = \langle \delta_1, \delta_2 \rangle$: composition F2F mapping from $\mathcal{V}$ to $\mathcal{U}$.

Figure 12: Composing an F2F mapping for a merge transformation with a previous transformation.

6.1 Efficient Computation

Like we did for shrink transformations, we now discuss how we can compose merge transformations with other transformations more efficiently than with the general approach described in Section 4.3 (Equations 1–5). That is, we discuss how to efficiently compute the composed factored transformation $\tau'_\text{F} \circ \tau_\text{F}$ when τ'_F is a merge transformation.

It is easy to see that all aspects of τ_F other than the two removed factors and the corresponding FMs are unchanged and can thus be reused. Regarding the product factor, we cannot avoid computing it from scratch. However, the new F2N mapping $\sigma_\otimes$ can easily be obtained from the two F2N mappings of the two removed factors: use the two previous F2N mappings as components of $\sigma_\otimes$ and construct the table function $\sigma_\otimes^\text{tab}$.

In Definition 27, we define $\sigma_\otimes^\text{tab}(s_j, s_k) = \langle s_j, s_k \rangle$, i.e., the states of the new factor $\Theta_\otimes$ are represented as pairs, which is a clean and simple mathematical definition. However, in an implementation, it is wasteful to use composite data structures like pairs to denote such states. (After several merging steps, we would end up with deeply nested pairs of pairs.) The "names" of local states can be chosen arbitrarily, and hence it is advisable to use the simplest possible representation. In our implementation, we always represent states of factors as integers, consecutively numbered from 0. Assuming that the states of Θ_j and Θ_k are already numbered in this fashion and that Θ_k has N_k states, we can achieve the desired consecutive numbering for the states of $\Theta_\otimes$ by defining $\sigma_\otimes^\text{tab}(s_j, s_k) = s_j \cdot N_k + s_k$.

6.2 Example

Figure 12 shows an example of how a merge transformation affects a given F2F mapping. As described in the preceding discussion, we use integers for factor states throughout, rather than representing states of a factor obtained by merging as pairs.

Part (a) shows the F2F mapping Σ before merging. Part (b) shows the F2F mapping Γ that represents the merge step, merging the first two components. We see that the first component is a trivial projection because the merge transformation only affects the first two components of $\mathcal{V}$ and hence the third component of $\mathcal{V}$ remains as the first component of $\mathcal{W}$. The second component is a merge FM with projection FMs (onto the first and second component of $\mathcal{V}$) as its children. Part (c) shows the resulting F2F mapping $\Gamma \circ \Sigma$. It has two components: the first one is identical to the third component of the first transformation, reflecting that Γ does not change this component. The second component can be efficiently constructed by taking the first and second component of Σ and combining them into a merge FM in the same way that the second component of Γ merges its children.

To illustrate the effect of merging on transition systems, consider the factored transition system derived from the example planning task of Figure 4 that consists of the atomic factor Θ^T and the factor $\tilde{\sigma}_P$, which corresponds to the shrunk atomic factor $\tau(\Theta^P)$ of the shrinking example in Figure 9. Figures 13(a) and 13(b) show the two factors (top) and the F2N mappings σ_T and $\tilde{\sigma}_P$ (bottom). Figure 13(c) illustrates the result of merging the two components. The top part shows the transformed factored transition system that consists of a single factor, which is the product $\Theta_\otimes = \Theta^T \otimes \tilde{\Theta}^P$ (cf. the illustration of computing products in Section 4 on page 795). The bottom part shows the F2N mapping $\sigma_\otimes$.

As a remark on how shrinking and merging interact, we observe that $\Theta_\otimes$ differs from the induced transition system $\Theta(\Pi)$ shown in Figure 1. The difference, apart from names of states, stems from the merge transformation being performed on the shrunk factor. The same result could have been obtained by shrinking $\Theta(\Pi)$, combining each pair of states where the truck is at one specific location and the package is at A or B into a single state.

6.3 Properties

We now investigate the formal properties of merge transformations.

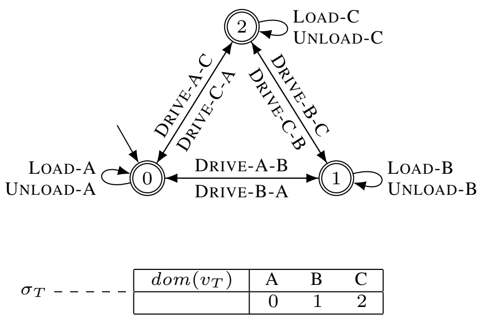
Theorem 12. *All merge transformations are exact induced, i.e., they satisfy* **CONS** + **IND** + **REF**.

Together with Theorem 1, an immediate consequence is that we can recover the entire state space of an underlying planning task from merging all individual transition systems of the induced factored transition system. That is, the global transition system $\bigotimes F$ of the induced factored transition system F of a planning task Π equals $\Theta(\Pi)$, apart from names of states. This result was already shown in previous work (e.g., Helmert et al., 2014).

Besides the properties mentioned in Theorem 12, we also consider the properties exclusive to factored transformations (Definition 23). The following theorem shows that merge transformations are locally nondecreasing.

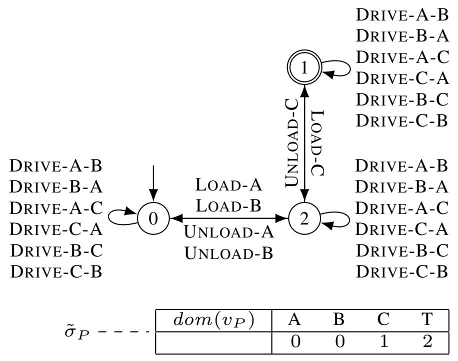
Theorem 13. *All merge transformations are locally nondecreasing, i.e., they satisfy* $\mathbf{LOC}_\geq$.

It is easy to see that merge transformations do not necessarily satisfy $\mathbf{LOC}_\leq$ (and hence $\mathbf{LOC}_=$) because the local heuristic values of the product can be larger than the local heuristic values of the



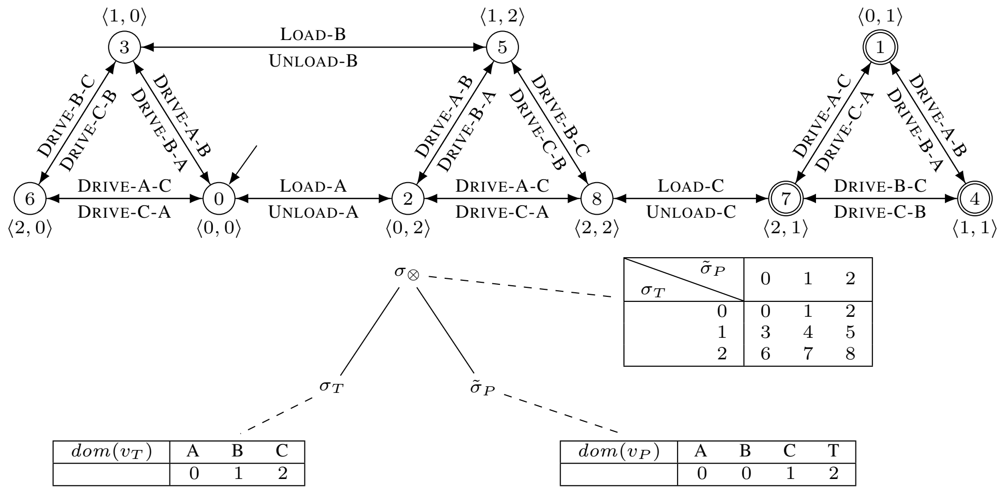
$dom(v_T)$	A	B	C
	0	1	2

(a) Atomic factor Θ^T (top) and F2N mapping σ_T (bottom).



$dom(v_P)$	A	B	C	T
	0	0	1	2

(b) Factor $\tilde{\Theta}^P$ (top), which is Θ^P shrunk as in Figure 9(b), and F2N mapping $\tilde{\sigma}_P$ (bottom).



σ_T \ $\tilde{\sigma}_P$	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8

$dom(v_T)$	A	B	C
	0	1	2

$dom(v_P)$	A	B	C	T
	0	0	1	2

(c) Transformed factored transition system with a single factor $\Theta_\otimes$ (top) and F2N mapping $\sigma_\otimes$ (bottom).

Figure 13: Example of a merge transformation applied to the factored transition system that is the result of the shrink transformation of Figure 9.

component factors. In the example of Figure 13, the maximum local heuristic of the initial state s before the merge is $h^{\mathrm{mf}}_{F}(s) = \max\{h^*_{\Theta^T}(0), h^*_{\tau(\Theta^P)}(0)\} = \max\{0, 2\} = 2$. After merging the two factors, we have $h^{\mathrm{mf}}_{F'}(s) = \max\{h^*_{\Theta_\otimes}(0)\} = 3$. Not being locally nonincreasing is of course a strength of merge transformations: they can improve the local heuristic, in principle all the way to the perfect heuristic if all factors can be merged with the given computational resources.

7. Label Reduction Transformation

In this section, we define label reduction transformations. Proofs can be found in Appendix E.

For a transition system $\Theta = \langle S, L, c, T, S_{\mathrm{I}}, S_{\mathrm{G}}\rangle$, a (total) label mapping λ defined on L, and a label cost function c' defined on $\lambda(L)$, the *transition system induced by* Θ*,* λ*, and* c' is defined as $\Theta^{\lambda,c'} = \langle S, \lambda(L), c', \{\langle s, \lambda(\ell), t\rangle \mid \langle s, \ell, t\rangle \in T\}, S_{\mathrm{I}}, S_{\mathrm{G}}\rangle$.

Definition 28 (Label Reduction Transformation)**.** *Let* $F = \langle \Theta_1, \dots, \Theta_n\rangle$ *be a factored transition system with label set* L *and label cost function* c*. Let* λ *be a total label mapping defined on* L*. The* label reduction transformation *(*label reduction *for short) for* λ *is the factored transformation* $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda\rangle$ *of* F *into a factored transition system* F' *with label set* $\lambda(L)$*, where:*

- $F' = \langle \Theta_1^{\lambda,c'}, \dots, \Theta_n^{\lambda,c'}\rangle$*, where* $c'(\ell') = \min_{\ell \in \lambda^{-1}(\ell')} c(\ell)$ *for all* $\ell' \in \lambda(L)$*.*
- Σ *is the identity mapping, i.e.,* $\Sigma = \langle \pi_1, \dots, \pi_n\rangle$*.*

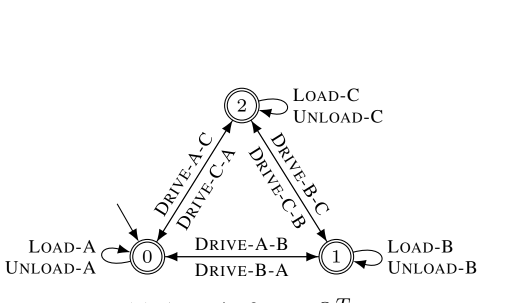
τ_{F} *is called* atomic *if* λ *only combines two labels, i.e., if there exist* $\ell_1, \ell_2 \in L$ *with* $\ell_1 \neq \ell_2$ *such that* $\lambda(\ell_1) = \lambda(\ell_2) = \ell_{12}$ *(where* $\ell_{12} \notin L$ *is a fresh label) and* $\lambda(\ell) = \ell$ *for all* $\ell \in L \setminus \{\ell_1, \ell_2\}$*. It is called* general *otherwise.*

Informally speaking, a label reduction relabels all transitions of all factors, replacing every label ℓ with the label $\lambda(\ell)$. If λ maps multiple labels to the same label, then the cost of the resulting label is the minimum of the original label costs. This keeps label costs as large as possible while remaining cost-conservative. For applications that are not aimed at deriving admissible heuristics, a more general notion of label reduction with flexible new label costs might be useful.

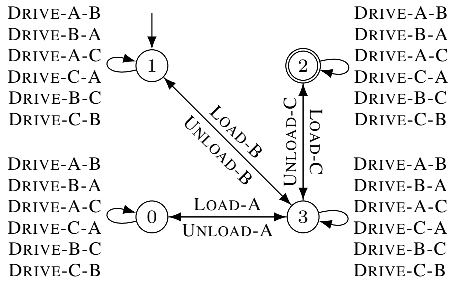
We consider two variants of label reductions based on λ: if it only combines two labels and leaves all others unchanged, the label reduction is called atomic. Otherwise, it is called general. It is easy to see that general label reductions can be understood as compositions of atomic ones. We define atomic label reductions because they are central to our analysis of the properties of label reductions in Section 7.3.

We call the transformation a label *reduction* because λ usually maps the labels of F to some smaller set. (Otherwise, the transformation would just "rename" labels, with no meaningful impact on the transition system.) Hence, a label reduction potentially collapses parallel transitions of previously different labels to identical transitions under the new labeling, thus reducing the number of transitions in the transformed (factored) transition system.

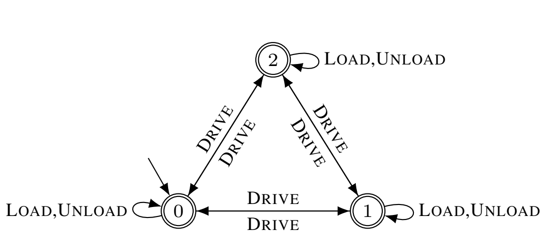
Label reduction is somewhat analogous to shrinking: where shrinking combines *states* of a factor to reduce the size of the factored transition system representation (potentially at the cost of overapproximating the possible behaviors), label reduction analogously combines *labels*. Unlike shrinking, label reduction is a global transformation of the factored transition system, i.e., it impacts all factors simultaneously. This global perspective is needed because the main purpose of labels in a



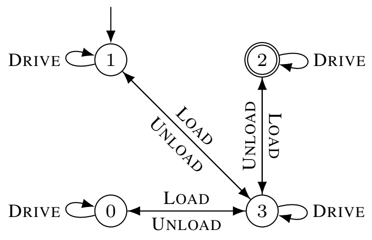
(a) Atomic factor Θ^T.



(b) Atomic factor Θ^P.



(c) Transformed factor $(\Theta^T)^{\lambda,c'}$.



(d) Transformed factor $(\Theta^P)^{\lambda,c'}$.

Figure 14: Example of a label reduction τ applied to the induced factored transition system of the example planning task of Figure 4. FMs are not shown for simplicity. Top two figures: factors of the induced factored transition system. Bottom two figures: transformed factors after applying τ which maps all labels LOAD-* onto a new label LOAD, all labels UNLOAD-* onto UNLOAD, and all labels DRIVE-*-* onto DRIVE.

factored transition system is to specify the synchronization between factors, i.e., to define the joint behavior of the factored transition system in terms of the local behaviors of the factors.

One might consider alternative names for shrinking and label reduction that make this similarity clearer. For example, shrinking could be called *state abstraction*, and label reduction could be called *label abstraction*. We prefer the terminology used in this paper due to its widespread use in the literature.

Label reduction has a positive synergy with bisimulation-based shrinking (Nissim et al., 2011). Bisimulation-based shrinking is an exact transformation that can reduce the size of the factored transition system representation (Theorem 9), and applying label reduction can increase the potential for bisimulation-based shrinking. A common general merge-and-shrink strategy therefore applies label reductions *before* shrink transformations (based on bisimulation) to leverage this synergy (Sievers, 2018).

7.1 Efficient Computation

Composing label reductions with previous transformations is straightforward: label mappings are nonfactored functions that can directly be composed. Furthermore, the state mapping of label reductions is the identity, and therefore the given FM can be reused with no change. Regarding the factored transition system, it suffices to replace each transition $s \xrightarrow{\ell} t$ by $s \xrightarrow{\lambda(\ell)} t$, taking care to remove duplicate transitions that may arise.

In the implementation of merge-and-shrink in the Fast Downward planner (Helmert, 2006), each factor keeps track of an equivalence relation over the labels, with two labels belonging to the same class if they induce exactly the same set of transitions. For each equivalence class, its transitions are represented only once. This representation is more compact than a more naive alternative and also allows for faster implementations of many operations, including label reductions. For more details, we refer to Sievers (2018).

7.2 Example

Consider the induced factored transition system of the example planning task of Figure 4 (page 799). Figures 14(a) and 14(b) show the two atomic factors. Below, in Figures 14(c) and 14(d), we see the label-reduced factors after applying a label reduction τ that maps all labels LOAD-* onto a new label LOAD, all labels UNLOAD-* onto UNLOAD, and all labels DRIVE-*-* onto DRIVE. (All new labels LOAD, UNLOAD, and DRIVE cost 1 since the example is unit-cost.) From the reduced number of labels labeling the transitions in the figure, we observe that the label-reduced factors require far fewer transitions because many originally parallel transitions are collapsed into single transitions.

For the remainder of this section we use an additional example (Figure 15) to illustrate the properties that label reductions can have. There are two factors Θ_x and Θ_y which encode a position on the x-axis (Figure 15(a)) and on the y-axis (Figure 15(b)). The first one uses labels ℓ_1, ℓ_2, ℓ_3 for state-changing transitions, and the second one uses labels ℓ_4, ℓ_5, ℓ_6. In both cases, the other labels only induce self-looping transitions. Figure 15(c) shows the product of the two factors. If we first apply a label reduction for λ that maps ℓ_1, ℓ_2, and ℓ_3 to r ("right") and ℓ_4, ℓ_5, and ℓ_6 to d ("down"), then, after merging the two label-reduced factors, the product contains more state-changing transitions (Figure 15(d)): in particular, the product now is the complete grid because all state-changing transitions of either factor now synchronize with all self-looping transitions of the other factor. The example illustrates that label reduction is not in general an exact transformation and can introduce spurious transitions. One of the main questions discussed in the rest of this section is under which conditions label reduction is or is not exact.

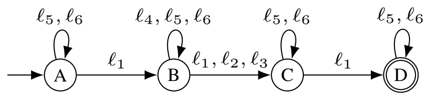
7.3 Properties

We begin our study of the transformation properties of label reduction by discussing the properties that every label reduction has.

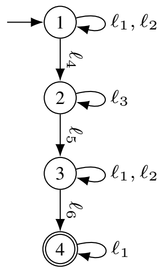
Theorem 14. *All label reduction transformations are abstractions, i.e., they satisfy* **CONS**. *They also satisfy* $\mathbf{IND_{S+L+C+I+G}}$*, are goal-refinable, i.e., satisfy* $\mathbf{REF_G}$*, and are locally nonincreasing, i.e., satisfy* $\mathbf{LOC_{\leq}}$.

Together with Theorem 3, an immediate consequence is that the heuristic induced by a label reduction is admissible and consistent. However, as the example above already illustrated, label reduction is not in general exact or induced. There are four desirable properties that do not hold in general: $\mathbf{REF_C}$, $\mathbf{LOC_{\geq}}$ (and hence $\mathbf{LOC_{=}}$), $\mathbf{IND_T}$ and $\mathbf{REF_T}$.

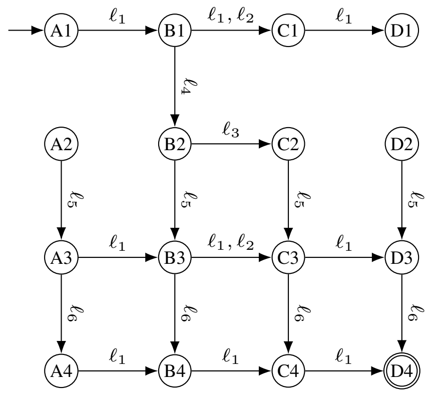
It is easy to see that label reductions do not necessarily satisfy $\mathbf{REF_C}$: every label mapping that combines two labels with different cost violates this property. As a consequence, it is also easy to construct counterexamples to $\mathbf{LOC_{\geq}}$. For example, in a factored transition system with a single



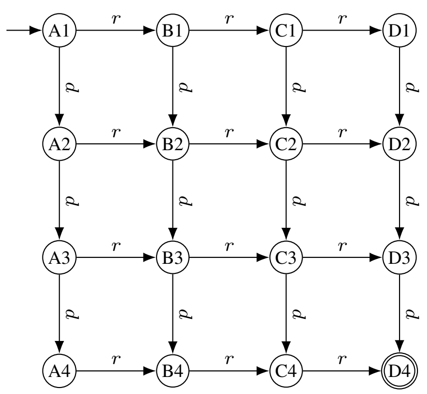
(a) Factor Θ_x.



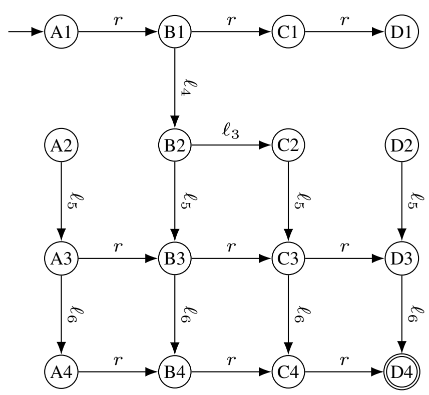
(b) Factor Θ_y.



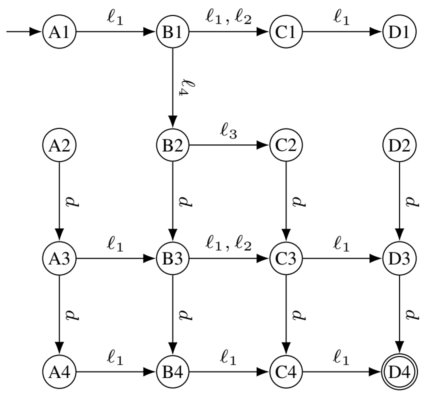
(c) Product $\Theta_x \otimes \Theta_y$.



(d) Product after applying label reduction for $\lambda(\ell_1) = \lambda(\ell_2) = \lambda(\ell_3) = r$ and $\lambda(\ell_4) = \lambda(\ell_5) = \lambda(\ell_6) = d$. Not transition-induced or transition-refinable.



(e) Product after applying atomic label reduction with $\lambda(\ell_1) = \lambda(\ell_2) = r$. Exact induced transformation because ℓ_1 globally subsumes ℓ_2.



(f) Product after applying atomic label reduction with $\lambda(\ell_5) = \lambda(\ell_6) = d$. Exact induced transformation because ℓ_5 and ℓ_6 are Θ_y-combinable.

Figure 15: Example with two factors Θ_x (a) and Θ_y (b). Part (c) shows the product $\Theta_x \otimes \Theta_y$. Parts (d)–(f) show products of label-reduced factors $\Theta_x^{\lambda,c'} \otimes \Theta_y^{\lambda,c'}$ for different label mappings λ indicated in the captions of the parts. All labels have unit cost.

factor, any label mapping that maps a label ℓ to a label of cheaper cost violates $\mathbf{LOC}_{\geq}$ if ℓ is part of a shortest path.

To see that label reductions do not satisfy $\mathbf{IND_T}$ and $\mathbf{REF_T}$ in general, consider again the example shown in Figure 15. In the product after applying the label reduction for $\lambda(\ell_1) = \lambda(\ell_2) = \lambda(\ell_3) = r$ and $\lambda(\ell_4) = \lambda(\ell_5) = \lambda(\ell_6) = d$ (Figure 15(d)), consider, e.g., A1 $\xrightarrow{d}$ A2. In the original product (Figure 15(c)), there exists no transition from A1 to A2, which means that this label reduction is neither transition-induced nor transition-refinable.

In the following, we discuss under which restrictions these four properties hold. We first discuss the cost-related properties $\mathbf{REF_C}$ and $\mathbf{LOC}_{\geq}$, which are comparatively simple.

Theorem 15. *Consider a label reduction transformation τ_{F} of a factored transition system with cost function c for label mapping λ. τ_{F} is cost-refinable (*$\mathbf{REF_C}$*) iff λ only combines labels of the same cost, i.e., $\lambda(\ell) = \lambda(\ell')$ implies $c(\ell) = c(\ell')$ for all ℓ, ℓ'.*

Theorem 16. *Consider a label reduction transformation τ_{F} of a factored transition system with cost function c for label mapping λ. If λ only combines labels of the same cost, then τ_{F} is locally non-decreasing (*$\mathbf{LOC}_{\geq}$*).*

Both results are straightforward: if we want to obtain cost-refinability and a locally equal heuristic, we should only combine labels of the same cost. Note that the second result is only a sufficient condition, not a necessary condition, as reducing the cost of a label does not necessarily increase the max-factor heuristic. It is easy to see that label reduction preserves all local heuristic values in a given factor iff no costs are reduced below their *saturated costs*, as discussed in the work on saturated cost partitioning (Seipp, Keller, & Helmert, 2020).

For the transition-related properties $\mathbf{IND_T}$ and $\mathbf{REF_T}$, the situation is more complicated. We first introduce a notation that allows us to reason about these properties at a higher level of abstraction.

Definition 29 (Transition Set)**.** *Let Θ be a transition system, and let ℓ be a label of Θ. The* transition set *for Θ and ℓ is defined as $\mathcal{T}(\Theta, \ell) = \{\langle s, s' \rangle \mid s \xrightarrow{\ell} s' \in \Theta\}$.*

In words, $\mathcal{T}(\Theta, \ell)$ denotes the set of transitions (represented as pairs of states) with the label ℓ in the transition system Θ. For a factored transition system $F = \langle \Theta_1, \dots, \Theta_n \rangle$, from the definition of synchronized products (Definition 13), it is easy to see that $\mathcal{T}(\bigotimes F, \ell)$ corresponds to $\prod_{i=1}^{n} \mathcal{T}(\Theta_i, \ell)$, except that the transition set of the product consists of pairs of tuples of the form $\langle \langle s^1, \dots, s^n \rangle, \langle t^1, \dots, t^n \rangle \rangle$, while the product of the transition sets consists of tuples of pairs of the form $\langle \langle s^1, t^1 \rangle, \dots, \langle s^n, t^n \rangle \rangle$. We can now characterize when $\mathbf{IND_T}$ and $\mathbf{REF_T}$ hold in terms of the transition sets of the factors. As a side result, we see that the two properties are equivalent to each other for label reductions.

Theorem 17. *Let $F = \langle \Theta_1, \dots, \Theta_n \rangle$ be a factored transition system with labels L. Consider the label reduction transformation τ_{F} of F for label mapping λ. Then τ_{F} satisfies* $\mathbf{IND_T}$ *iff τ_{F} satisfies* $\mathbf{REF_T}$*, and it satisfies these properties iff for all transformed labels $\ell' \in \lambda(L)$:*

$$\bigcup_{\ell \in \lambda^{-1}(\ell')} \prod_{i=1}^{n} \mathcal{T}(\Theta_i, \ell) = \prod_{i=1}^{n} \bigcup_{\ell \in \lambda^{-1}(\ell')} \mathcal{T}(\Theta_i, \ell).$$

Intuitively, a label reduction is transition-induced if it does not introduce any spurious transitions, i.e., all transitions in $\bigotimes F'$ are already present in $\bigotimes F$. For a given transformed label ℓ', the left-hand side $\bigcup_{\ell \in \lambda^{-1}(\ell')} \prod_{i=1}^{n} \mathcal{T}(\Theta_i, \ell)$ is the union of all transitions in the (original) product whose new label is ℓ', i.e., the exact set of transitions that should receive the label ℓ' under a transition-induced transformation. The right-hand side $\prod_{i=1}^{n} \bigcup_{\ell \in \lambda^{-1}(\ell')} \mathcal{T}(\Theta_i, \ell)$ denotes the transitions with label ℓ' that are actually present in the product after label reduction. This may in general include spurious transitions because combining transition labels before forming the product (= moving the set union into the set product) can allow synchronization between transitions with different original labels, as in the example of Figure 15(d).

Geometrically, we can interpret the left-hand side as a union of hypercubes (Cartesian products), while the right-hand side corresponds to computing the projection of these hypercubes to each dimension (forming the union of the projections), and then taking the product of these projections. For equality to hold, the hypercubes on the left-hand side must fully cover the product of their projections. A closely related problem was studied by Hoffmann and Kupferschmid (2005), and the following complexity result follows a similar idea as their analysis.

Definition 30 (LABELREDUCTION-$\mathbf{IND_T}$)**.** *The decision problem* LABELREDUCTION-$\mathbf{IND_T}$ *is defined as follows:*

Given: *a factored transition system F and label mapping λ*

Question: *Does the label reduction of F for λ have the property* $\mathbf{IND_T}$*?*

Theorem 18. LABELREDUCTION-$\mathbf{IND_T}$ *is* **coNP***-complete.*

The same result can of course also be stated for $\mathbf{REF_T}$ instead of $\mathbf{IND_T}$ due to the equivalence of the two properties for label reductions. This complexity result is bad news in the sense that we cannot expect to find an efficiently verifiable characterization of exact induced label reductions. Therefore, we next consider the more restricted case of atomic label reductions where only two labels are combined at a time, and general label reductions must be expressed as a sequence of atomic steps. For atomic label reductions, we will be able to provide a complete and efficiently verifiable characterization of transition-inducedness, which is based on the following definition.

Definition 31 (Properties of Labels)**.** *Let F be a factored transition system with label set L, let $\ell, \ell' \in L$ be labels, and let $\Theta \in F$ be a transition system.*

- *Label ℓ* locally subsumes *label ℓ' in Θ if $\mathcal{T}(\Theta, \ell') \subseteq \mathcal{T}(\Theta, \ell)$.*
- *Label ℓ* globally subsumes *label ℓ' in F if ℓ locally subsumes ℓ' in all $\Theta' \in F$.*
- *Labels ℓ and ℓ' are* locally equivalent *in Θ if they locally subsume each other in Θ, i.e., $\mathcal{T}(\Theta, \ell) = \mathcal{T}(\Theta, \ell')$.*
- *Labels ℓ and ℓ' are* Θ-combinable *in F if they are locally equivalent in all transition systems $\Theta' \in F$ with $\Theta' \neq \Theta$. (It does not matter whether or not they are locally equivalent in Θ.)*

Rephrased in natural language, a label ℓ globally subsumes another label ℓ' if (in all factors) all transitions labeled with ℓ' have corresponding transitions labeled with ℓ. It is evident that transitions with label ℓ' are not necessary to find plans in such factored transition systems, as we can always use transitions with label ℓ instead. (However, ℓ' may still be useful if it has lower cost than ℓ.)

Θ-combinability is a more unusual property. Two labels are Θ-combinable if they are equivalent in all factors except possibly one. In our running example (Figure 14), all labels DRIVE-*-* are

Θ^T-combinable because these labels are locally equivalent in the atomic factor Θ^P, which is the only factor other than Θ^T.

We can now characterize under which conditions atomic label reductions are transition-induced. In addition to the new properties, it also matters whether the labels involved in a label reduction are *dead*. Recall from Definition 26 on page 818 that a label ℓ is dead in transition system Θ if it labels no transition in Θ, i.e., $\mathcal{T}(\Theta, \ell) = \emptyset$.

Theorem 19. *Let F be a factored transition system with label set L. Let λ be a label mapping defined on L with $\lambda(\ell_1) = \lambda(\ell_2) = \ell_{12}$ for $\ell_1, \ell_2 \in L$, $\ell_{12} \notin L$, and $\lambda(\ell) = \ell$ for all $\ell \in L \setminus \{\ell_1, \ell_2\}$. The (atomic) label reduction of F for λ satisfies* $\mathbf{IND_T}$ *(equivalently,* $\mathbf{REF_T}$*) iff*

1. *ℓ_1 globally subsumes ℓ_2, or*
2. *ℓ_2 globally subsumes ℓ_1, or*
3. *ℓ_1 and ℓ_2 are Θ-combinable for some $\Theta \in F$, or*
4. *there exists $\Theta \in F$ such that ℓ_1 and ℓ_2 are dead in Θ.*

Intuitively, if ℓ_1 globally subsumes ℓ_2 or vice versa, then the subsumed label is redundant and label reduction is effectively the same as discarding the redundant label (apart from label cost considerations). If two labels are Θ-combinable, they behave the same in all factors except possibly Θ. In this case, label reduction cannot introduce spurious transitions: a spurious transition can only arise via "false synchronization" if there exists one factor in which a transition is labeled by ℓ_1 but not ℓ_2 and another factor in which a transition is labeled by ℓ_2 but not ℓ_1. But this would require the two labels to be non-equivalent in at least two factors, which cannot happen for Θ-combinable labels. Finally, if both labels are dead in the same factor, then the combined label is also dead and hence induces no transitions in the product.

It is important to point out that the theorem is again an exact characterization, i.e., whenever an atomic label reduction does *not* have one of the listed properties, it is not transition-induced. Moreover, all properties referenced in the theorem are easily checkable in polynomial time in the size of the factored transition system. Hence the theorem provides a complete understanding of atomic label reductions, which is good news to complement the bad (**coNP**-completeness) news for general label reductions.

Figure 15 shows two examples that illustrate the theorem. Figure 15(c) shows the product of two factors without label reduction. Figure 15(e) shows the product after applying an atomic label reduction where we combine ℓ_1 and ℓ_2 with ℓ_1 globally subsuming ℓ_2. Figure 15(f) shows the product after applying an atomic label reduction where the Θ_y-combinable labels ℓ_5 and ℓ_6 are combined. We see that all transitions in Figures 15(e) and Figure 15(f) are indeed induced by transitions in Figure 15(c).

Let us call two labels ℓ_1 and ℓ_2 *exactly combinable* if $c(\ell_1) = c(\ell_2)$ (for Theorem 15) and they satisfy any of the conditions of Theorem 19. We denote this with $\ell_1 \bigtriangleup \ell_2$.[9] The results of this section imply that combining two exactly combinable labels is an exact induced transformation.

9. We avoid a notation like $\ell_1 \sim \ell_2$ because it suggests an equivalence relation, and while exact combinability is reflexive and symmetric, it is not transitive. Some of the individual conditions in Theorem 15 are equivalence relations, specifically conditions 3. or 4. for a fixed factor Θ, but 1. and 2. are not equivalence relations, and neither is the combination of the four properties.

As noted earlier, a general label reduction can always be expressed as the composition of atomic label reductions. However, not every exact label reduction involving more than two labels can be expressed as the composition of exact label reductions of two labels. One source of difficulty are situations where $\ell_1 \bigtriangleup \ell_2$, $\ell_1 \bigtriangleup \ell_3$ and $\ell_2 \bigtriangleup \ell_3$, so any two labels in $\{\ell_1, \ell_2, \ell_3\}$ can be exactly combined, but combining all three together is not necessarily exact because, e.g., after combining ℓ_1 and ℓ_2 into ℓ_{12}, we may have $\ell_{12} \not\bigtriangleup \ell_3$.

Fortunately, things are simpler if we consider the conditions in Theorem 19 individually. If the three labels in the above example (or any larger set of labels) are exactly combinable *for the same reason* (i.e., one label globally subsumes all others, or they are Θ-combinable *for the same factor* Θ, or they are dead *in the same factor*), then all labels can be exactly combined. In this case, it does not make a difference if they are combined in a single step or two at a time because combining two labels will preserve the relevant condition of the theorem for the new label with respect to the remaining labels. For example, all DRIVE-*-* labels in Figure 14 can be exactly combined because they are all Θ^T-combinable. The merge-and-shrink algorithms based on label reduction described in the literature (Sievers et al., 2014; Sievers, 2017, 2018) exploit this fact. Roughly speaking, they identify all labels that are Θ-combinable for a given factor Θ, combine them, then iterate (possibly for different factors Θ) until no more exact label reductions due to Θ-combinability are possible.[10]

The following theorem summarizes the results of this section.

Theorem 20. *A label reduction transformation of a factored transition system F with labels L and label costs c that only combines two labels $\ell_1, \ell_2 \in L$ is an exact induced transformation, i.e., satisfies* **CONS** + **IND** + **REF**, *iff $c(\ell_1) = c(\ell_2)$ and*

1. *ℓ_1 globally subsumes ℓ_2, or*
2. *ℓ_2 globally subsumes ℓ_1, or*
3. *ℓ_1 and ℓ_2 are Θ-combinable for some $\Theta \in F$, or*
4. *there exists $\Theta \in F$ such that ℓ_1 and ℓ_2 are dead in Θ.*

General label reductions always satisfy $\mathbf{CONS} + \mathbf{IND_{S+L+C+I+G}} + \mathbf{REF_G}$ *and* $\mathbf{LOC_{\leq}}$. *They satisfy* $\mathbf{REF_C}$ *iff they only combine labels of the same cost. Cost-refinable label reductions additionally satisfy* $\mathbf{LOC_{=}}$. *Testing* $\mathbf{IND_T}$ *and* $\mathbf{REF_T}$ *is* **coNP**-*complete for general label reductions.*

Conditions 1.–4. are easily checkable in polynomial time, so there is a sharp contrast in complexity between the properties of atomic and general label reductions.

7.4 Relationship to Previous Work

Implementations of merge-and-shrink heuristics have leveraged some variations of the concept of label reduction since the first paper on merge-and-shrink abstractions in the planning literature (Helmert et al., 2007). This first incarnation of label reduction (*HHH label reduction* in the following) was not described in the original papers and treated as an implementation detail.

Early papers on merge-and-shrink in the planning literature only considered so-called *linear merge strategies*. Nissim et al. (2011) gave the first formal account of HHH label reduction, but only

10. The conditions from Theorem 19 other than Θ-combinability have not yet been experimentally explored because checking subsumption is considerably more expensive, and dead labels are believed to be rare.

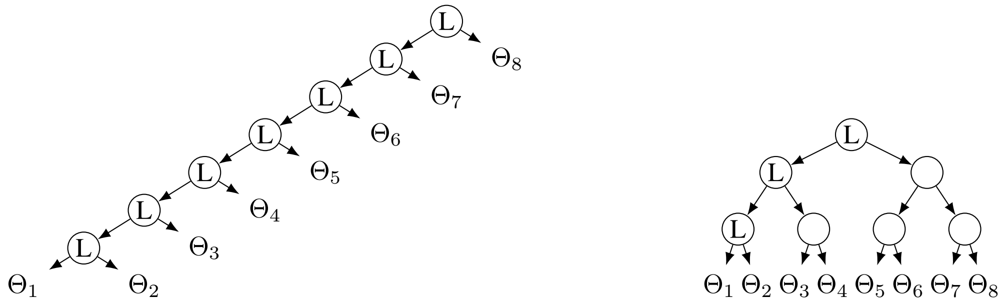


Figure 16: Two merge trees for a factored transition system with 8 factors. HHHN label reduction allows reducing labels in the merge steps marked with an "L" when Θ_1 is the pivot.

discussed linear merging. When planning researchers first considered general merge strategies, such as those used earlier in the model-checking literature (Dräger et al., 2006, 2009), it turned out that HHH label reduction breaks admissibility of merge-and-shrink heuristics. (Indeed, it often leads to heuristics that assign ∞ to all states.) Helmert et al. (2014) identified the problem of HHH label reduction with non-linear merge strategies. Their variant of label reduction (*HHHN label reduction*) guarantees admissibility, but can only be applied in restricted scenarios.

Figure 16 shows two examples of *merge trees*. Merge trees illustrate the merge strategy, i.e., the order in which different factors of a factored transition system are merged by a merge-and-shrink algorithm. Transformations affecting single factors, such as shrinking or the prune transformation described in the next section, are not shown. The left merge tree gives an example of a linear merge strategy, which can be described by a linear ordering $\langle \Theta_1, \dots, \Theta_n \rangle$ of factors: first Θ_1 is merged with Θ_2, then the merged (and possibly further transformed) factor is merged with Θ_3, and so on. The right merge tree cannot be fully characterized by an ordering of the original factors. For example, Θ_1 and Θ_2 are merged, Θ_3 and Θ_4 are merged, and then the two merged factors are merged. (Often, there would be shrinking transformations in between the merges.) This is an example of a *non-linear* merge strategy.

HHHN label reduction requires choosing one of the factors as a *pivot*. Label reduction transformations are then only allowed right before merge transformations that directly or indirectly involve the pivot. In Figure 16, these merge transformations are marked with an "L". This means that label reduction is limited to a single branch of the merge tree, which is a significant restriction for non-linear merge strategies (right example), but not for linear ones (left example). The use of pivots in HHHN label reduction can be viewed as a special case of our more general result that label reduction based on Θ-combinability (and equal label costs) is exact. Put briefly, at every stage of the merge-and-shrink computation, the current factored transition system contains exactly one factor that resulted from merges involving the pivot, and the HHHN label reduction conditions imply Θ-combinability with respect to this factor Θ.

Besides limiting the opportunities for simplification due to label reduction, the restriction to one branch of the merge tree hints at a more serious conceptual problem of HHHN label reduction. In the HHHN framework, label reduction cannot be understood as a generally applicable transformation that can be composed with other transformations: in order to determine in which cases labels can be safely reduced and in which cases they cannot, the merge-and-shrink process needs to be considered as a whole, with complex dependencies between merge and label reduction transformations that happen far apart during the overall transformation process. In other words, the HHHN theory is not compositional.

Moreover, HHHN label reduction does not allow treating factored transition systems as a self-contained representation. While all other transformations can be understood purely at the level of factors and opaque labels (i.e., labels do not have any semantic meaning), HHHN label reduction requires additional information about the "underlying meaning" of a label that must be derived from syntactic information in a separate representation, for example as a planning task. This loses much of the appeal of the merge-and-shrink framework. For example, if we obtain the same factored transition system in two different contexts, HHHN label reduction cannot treat them the same way due to this dependence on external representations.

Finally, HHHN label reduction is very complex and at the same time much more limited than the *compositional label reduction* transformation described in this section. As a consequence, it has never been used in an implementation. Before compositional label reduction was introduced, the planning literature only considered linear merge strategies and used HHH label reduction. (The original papers on merge-and-shrink in the model checking literature used non-linear merge strategies, but no label reduction.)

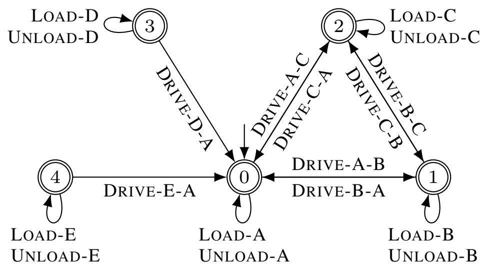
The compositional label reduction transformation described in this section was originally introduced by Sievers et al. (2014), and this section can be considered an extended version and further development of this paper. Enabled by the new theory, Sievers et al. first considered non-linear merge strategies in the planning literature, adapting ideas of Dräger et al. (2006, 2009). Since then, several further papers have focused on merge strategies, and non-linear merging has become the norm (e.g., Fan et al., 2014; Sievers et al., 2015; Sievers, Wehrle, & Helmert, 2016).

The main theoretical result of Sievers et al. (2014) is a slightly weaker version of Theorem 19: while we give a full characterization of exact label reductions combining two labels, the earlier paper treats the absence of dead labels as an additional requirement. The **coNP**-completeness result in this section is new, as is the treatment of label reduction as a factored transformation and its study in terms of transformation properties.

8. Prune Transformation

In this section we study prune transformations. Proofs of theorems can be found in Appendix F. Pruning has only been treated as a side note previously (e.g., Helmert et al., 2014, Section 4.3), but like label reduction (Section 7) it has been an important ingredient of the merge-and-shrink framework since its introduction for planning. The goal of pruning is to remove a subset of "uninteresting" states of a transition system. In all experiments with merge-and-shrink heuristics in the literature, pruning has been used to remove states that the merge-and-shrink construction algorithm recognizes as *dead*, i.e., not part of any solution.

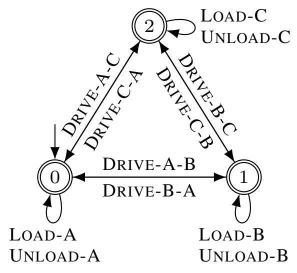
However, there are contexts in which dead states cannot be safely pruned. For example, the orbit search algorithm (Domshlak, Katz, & Shleyfman, 2015) for exploiting structural symmetries in planning tasks loses its optimality guarantees when using merge-and-shrink heuristic that prune dead states. In this section we refine our study of transformation properties to better explain this and related phenomena. Similarly to the previous sections, we develop a compositional understanding of prune transformations that allows us to explain exactly which local transformation properties are necessary to preserve global properties such as admissibility and consistency of heuristics.



σ_T

$dom(v_T)$	A	B	C	D	E
	0	1	2	3	4

(a) Atomic factor Θ^T (top) and F2N mapping σ_T (bottom).



σ'_T

$dom(v_T)$	A	B	C	D	E
	0	1	2	$\perp$	$\perp$

(b) Pruned factor (top) and transformed F2N mapping (bottom).

Figure 17: Example of a prune transformation. Only the affected factor and its associated state mapping are shown. Recall that we use $\perp$ to denote undefined values.

Like shrinking, pruning is a transformation that affects a single factor of a factored transition system. It discards a set of states, along with their incident transitions, which are *pruned* by the transformation, while not touching the remaining states, which are *kept*.

Definition 32 (Pruned Transition System)**.** *Let $\Theta = \langle S, L, c, T, S_\mathrm{I}, S_\mathrm{G}\rangle$ be a transition system. Given a subset $K \subseteq S$ of the states of Θ, the* transition system Θ pruned to K *is defined as $\Theta^K = \langle K, L, c, \{\langle s, \ell, s'\rangle \mid \langle s, \ell, s'\rangle \in T \text{ and } s, s' \in K\}, S_\mathrm{I} \cap K, S_\mathrm{G} \cap K\rangle$. We call K the set of* kept *states and $S \setminus K$ the set of* pruned *states.*

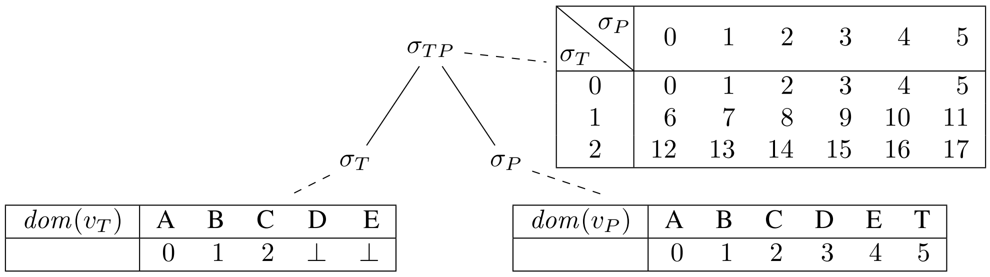
We can now define a factored transformation based on this notion of pruning.

Definition 33 (Prune Transformation)**.** *Let $F = \langle \Theta_1, \dots, \Theta_n\rangle$ be a factored transition system, let S_k be the set of states of Θ_k for some $1 \le k \le n$, and let $K \subseteq S_k$. The* prune transformation *for K and Θ_k is the factored transformation $\tau_\mathrm{F} = \langle F', \Sigma, \lambda\rangle$ of F where:*

- *$F' = \langle \Theta'_1, \dots, \Theta'_n\rangle$ with $\Theta'_i = \Theta_i$ for all $i \neq k$ and $\Theta'_k = \Theta_k^K$.*
- *$\Sigma = \langle \sigma_1, \dots, \sigma_n\rangle$ where $\sigma_i = \pi_i$ for all $i \neq k$, and σ_k is an atomic FM with variable Θ_k, $\sigma_k^{\text{tab}}(s_k) = s_k$ for all $s_k \in K$, and $\sigma_k^{\text{tab}}(s_k)$ is undefined for all $s_k \notin K$.*
- *$\lambda = \mathrm{id}$ is the identity label mapping.*

Prune transformations are our first examples of transformations where the state mapping is not total. Recall from Definition 7 that for a transformation $\tau = \langle \Theta', \sigma, \lambda\rangle$, we have $h^\tau_\Theta(s) = \infty$ if $\sigma(s)$ is undefined. For the factored transition system this means that we obtain infinite heuristic values for pruned states in the corresponding factor heuristic.

Efficiently composing a prune transformation with a previous transformation within the factored transformation framework is analogous to the case of shrink transformations: only one factor and the corresponding component F2N mapping need to be updated, which can be done in place as discussed in Section 5.1.



σ_T \ σ_P	0	1	2	3	4	5
0	0	1	2	3	4	5
1	6	7	8	9	10	11
2	12	13	14	15	16	17

$dom(v_T)$	A	B	C	D	E
	0	1	2	$\bot$	$\bot$

$dom(v_P)$	A	B	C	D	E	T
	0	1	2	3	4	5

Figure 18: An FM representing the state mapping from the induced transition system of the modified example planning task to the transition system pruned with the prune transformation shown in Figure 17.

8.1 Example

Consider the running example planning task, this time extended with new locations D and E from which the truck T can move to A but to which it cannot move. Figure 17(a) shows the atomic factor for Θ^T (top) and the F2N mapping σ_T representing the identity state mapping (bottom). (As in previous examples, we represent symbolic values like A and B as consecutive integers to match the common implementation.)

Figure 17(b) illustrates a prune transformation τ for the atomic factor Θ^T. Only the affected factor is shown. We keep the factor states $\{0, 1, 2\}$, corresponding to the locations $\{\text{A}, \text{B}, \text{C}\}$, and prune the factor states $\{3, 4\}$, corresponding to the locations $\{\text{D}, \text{E}\}$.

To illustrate non-atomic FMs that involve pruning, consider an FM for the running example, modified to reflect the additional locations D and E. Figure 18 shows the FM that results when we first prune D and E from v_T and then merge the resulting factor with the factor for v_P. The only aspect that changes when computing the function value of such an FM is that whenever the result for a component FM is undefined, the result for the parent FM is also undefined. For example, for $\alpha = \{v_T \mapsto \text{D}, v_P \mapsto \text{T}\}$ we can compute $[\![\sigma_{TP}]\!](\alpha)$ as follows:

$$\begin{aligned} & [\![\sigma_{TP}]\!](\alpha) \\ &= \sigma_{TP}^{\text{tab}}([\![\sigma_T]\!](\alpha), [\![\sigma_P]\!](\alpha)) \\ &= \sigma_{TP}^{\text{tab}}(\sigma_T^{\text{tab}}(\alpha[v_T]), \sigma_P^{\text{tab}}(\alpha[v_P])) \\ &= \sigma_{TP}^{\text{tab}}(\sigma_T^{\text{tab}}(\text{D}), \sigma_P^{\text{tab}}(\text{T})) \\ &= \sigma_{TP}^{\text{tab}}(\bot, 5) \\ &= \bot \end{aligned}$$

We also remark that in the resulting merged factor, all abstract states corresponding to $\text{D} \in dom(v_P)$ and $\text{E} \in dom(v_P)$ (abstract states 3, 4, 9, 10, 15, 16 in Figure 18) are dead and therefore could be pruned in a further prune transformation.

8.2 Reachability and Heuristic Properties

Prune transformations are especially useful in cases where they prune states that cannot possibly be part of any solution. In Section 2, we introduced admissible and consistent heuristics, and in Section 3.3 we discussed properties of transformations that imply these heuristic properties.

In many cases, for example within an A* search, admissibility and consistency are stronger requirements than necessary. For example, A* still produces optimal solutions if we assign infinite heuristic values to states that cannot be reached from the initial state, even when this violates admissibility. One of the most useful properties of prune transformations is that they can eliminate many such unreachable states. For a precise discussion of this topic, we now define appropriate variants of admissibility and consistency, along with transformation properties that imply these heuristic properties. In Section 8.3 we then discuss the conditions under which prune transformations have the desired properties.

8.2.1 REACHABILITY

As a first step, we formalize different notions of reachability.

Definition 34 (Forward-Reachable, Backward-Reachable, Alive, and Dead States)**.** *Let* $\Theta = \langle S, L, c, T, S_\mathrm{I}, S_\mathrm{G} \rangle$ *be a transition system. A state* $s \in S$ *is called*

- forward-reachable *if there exists a path from some state* $s' \in S_\mathrm{I}$ *to* s*.*
- backward-reachable *if there exists a path from* s *to some state* $s' \in S_\mathrm{G}$.
- alive *if it is forward- and backward-reachable.*
- dead *if it is not alive.*

We write $S_\rightarrow \subseteq S$ *for the set of forward-reachable states,* $S_\leftarrow \subseteq S$ *for the set of backward-reachable states, and* $S_\leftrightarrow = S_\rightarrow \cap S_\leftarrow \subseteq S$ *for the set of alive states of* Θ*.*

In other work, forward-reachable states are sometimes called *reachable* and backward-reachable states are sometimes called *relevant*. Next, we introduce variants of the basic heuristic properties (Definition 5 on page 785) that only consider forward-reachable states.

Definition 35 (Forward-Perfect, Forward-Goal-Aware, Forward-Consistent, Forward-Admissible)**.** *Let* h_Θ *be a heuristic for a transition system* Θ *with states* S *and label cost function* c*. The heuristic* h_Θ *is called*

- forward-perfect *if* $h_\Theta(s) = h^*_\Theta(s)$ *for all states* $s \in S_\rightarrow$*.*
- forward-goal-aware *if* $h_\Theta(s) = 0$ *for all goal states* $s \in S_\rightarrow$*.*
- forward-consistent *if* $h_\Theta(s) \leq c(\ell) + h_\Theta(t)$ *for all transitions* $s \xrightarrow{\ell} t \in \Theta$ *with* $s \in S_\rightarrow$*. (Note that this implies* $t \in S_\rightarrow$*.)*
- forward-admissible *if* $h_\Theta(s) \leq h^*_\Theta(s)$ *for all states* $s \in S_\rightarrow$*.*

It is easy to see that for algorithms like A* which search in the forward direction, these "forward" properties suffice. For example, forward-admissibility is enough for A* to produce optimal solutions because states that are not forward-reachable are never encountered during search. It is also easy to see that heuristics that are forward-goal-aware and forward-consistent are also forward-admissible: we can think of forward-goal-awareness and forward-consistency as the regular goal-awareness and consistency properties on a transition system restricted to the states $S_\rightarrow$. Because goal-aware and consistent heuristics are admissible, it follows that the heuristic is admissible restricted to $S_\rightarrow$, which is the forward-admissibility property.

8.2.2 Additional Properties of Transformations

As a second step, we define additional properties that transformations can have. As we will show in Section 8.3, these are the transformation properties that imply the new forward-reachable heuristic properties of the induced heuristics.

Definition 36 (States Between). *Let Θ be a transition system with states S. For two states $s, s' \in S$, we write $s \rightsquigarrow s'$ if there exists a path from s to s' in Θ.*

For $S', S'' \subseteq S$, we define the states between *S' and S'' as the set Between$(S', S'') = \{s \in S \mid s' \rightsquigarrow s, s \rightsquigarrow s'', s' \in S', s'' \in S''\}$.*

Intuitively, *Between*(S', S'') is the set of all states that are visited by some path from some state in S' to some state in S''. For example, we can write $S_{\rightarrow}$ as *Between*(S_{I}, S), $S_{\leftarrow}$ as *Between*(S, S_{G}) and $S_{\leftrightarrow}$ as *Between*$(S_{\mathrm{I}}, S_{\mathrm{G}})$.

Definition 37 (Closure and Keep Properties). *Let $\tau = \langle \Theta', \sigma, \lambda \rangle$ be a transformation of a transition system $\Theta = \langle S, L, c, T, S_{\mathrm{I}}, S_{\mathrm{G}} \rangle$ into a transition system Θ'. The following list defines properties that τ may have, along with a short-hand name for each property (analogously to Definition 8).*

$\mathbf{CLOS}_{pred}$ *The domain of σ is* closed under predecessors*: Between$(S, dom(\sigma)) \subseteq dom(\sigma)$.*

$\mathbf{CLOS}_{pred}^{\rightarrow}$ *The domain of σ is* closed under forward-reachable predecessors*: Between$(S_{\mathrm{I}}, dom(\sigma)) \subseteq dom(\sigma)$.*

$\mathbf{KEEP_G}$ *τ* keeps goal states*: $S_{\mathrm{G}} \subseteq dom(\sigma)$.*

$\mathbf{KEEP_G^{\rightarrow}}$ *τ* keeps forward-reachable goal states*: Between$(S_{\mathrm{I}}, S_{\mathrm{G}}) \cap S_{\mathrm{G}} \subseteq dom(\sigma)$.*

In words, $\mathbf{CLOS}_{pred}$ requires that for all states s on which the transformation is defined, it is also defined on all states on paths leading to s. This is equivalent to saying that whenever the transformation is defined for a state, it is also defined for its predecessors. $\mathbf{CLOS}_{pred}^{\rightarrow}$ is a slightly weaker variant of the same property that allows discarding states that are not forward-reachable, even if they are predecessors of states for which the transformation is defined. Similarly, $\mathbf{KEEP_G}$ requires the transformation to be defined on all goal states and $\mathbf{KEEP_G^{\rightarrow}}$ only requires this for goal states reachable from an initial state.

State-conservative transformations ($\mathbf{CONS_S}$) clearly have all four properties. In particular, all transformations considered in previous sections satisfy $\mathbf{CONS_S}$ and therefore also satisfy the four new properties. Hence, these properties can be seen as weaker forms of state-conservativeness where some states may be discarded, but not arbitrarily. (We require some structural properties for $dom(\sigma)$.) The following relationships between these properties are obvious (without proof):

- $\mathbf{CLOS}_{pred} \Rightarrow \mathbf{CLOS}_{pred}^{\rightarrow}$
- $\mathbf{KEEP_G} \Rightarrow \mathbf{KEEP_G^{\rightarrow}}$

8.2.3 Composition of Transformations

The transformation properties we discussed in previous sections are closed under composition, i.e., if τ and τ' have a certain property, then $\tau' \circ \tau$ also has it (Theorems 1 and 6). This compositionality is a key characteristic of these transformation properties because it means that complex combinations of merge-and-shrink transformations can be understood compositionally. Unfortunately, the new properties are not directly compositional. However, they do compose under mild side conditions.

Theorem 21. *Let X be any of the properties of transformations from Definition 37. Let τ be a transformation of transition system Θ into transition system Θ' with property X, and let τ' be a transformation of Θ' into transition system Θ'' with property X. Then the composed transformation $\tau'' = \tau' \circ \tau$ also has the property X if τ additionally satisfies the following side conditions (depending on X):*

- $\mathbf{CLOS}_{pred}$*: side conditions* $\mathbf{CONS_{L+T}}$
- $\mathbf{CLOS}^{\rightarrow}_{pred}$*: side conditions* $\mathbf{CONS_{L+T+I}}$
- $\mathbf{KEEP_G}$*: side condition* $\mathbf{CONS_G}$
- $\mathbf{KEEP}^{\rightarrow}_{\mathbf{G}}$*: side conditions* $\mathbf{CONS_{L+T+I+G}} + \mathbf{CLOS}^{\rightarrow}_{pred}$

In all cases, all side conditions on τ are necessary in the sense that if we drop any one of them, the result no longer holds. For example, if we only require that τ and τ' are $\mathbf{CLOS}^{\rightarrow}_{pred}$ and τ is $\mathbf{CONS_{L+T}}$ or $\mathbf{CONS_{L+I}}$ or $\mathbf{CONS_{T+I}}$, then τ'' is not necessarily $\mathbf{CLOS}^{\rightarrow}_{pred}$.

8.2.4 EFFECT OF PROPERTIES OF TRANSFORMATIONS ON HEURISTICS

We are now ready to generalize the results on heuristic properties from Section 3.3.

Theorem 22. *Let τ be a transformation of a transition system Θ. The heuristic h^τ for Θ induced by τ is*

1. *goal-aware if τ satisfies* $\mathbf{CONS_G} + \mathbf{KEEP_G}$*,*
2. *forward-goal-aware if τ satisfies* $\mathbf{CONS_G} + \mathbf{KEEP}^{\rightarrow}_{\mathbf{G}}$*,*
3. *consistent if τ satisfies* $\mathbf{CONS_{L+C+T}} + \mathbf{CLOS}_{pred}$*,*
4. *forward-consistent if τ satisfies* $\mathbf{CONS_{L+C+T}} + \mathbf{CLOS}^{\rightarrow}_{pred}$*,*
5. *admissible if τ satisfies* $\mathbf{CONS_{L+C+T+G}} + \mathbf{KEEP_G} + \mathbf{CLOS}_{pred}$*, and*
6. *forward-admissible if τ satisfies* $\mathbf{CONS_{L+C+T+G}} + \mathbf{KEEP}^{\rightarrow}_{\mathbf{G}} + \mathbf{CLOS}^{\rightarrow}_{pred}$*.*

This result generalizes Theorem 3: the results for goal-awareness, consistency and admissibility are strictly stronger in the sense that they require weaker conditions ($\mathbf{CONS_S}$ is replaced by weaker properties), and the conditions are relaxed further if we only need the "forward" versions of these heuristic properties.

Theorem 23. *Let τ be a transformation of a transition system Θ. The heuristic h^τ for Θ induced by τ is*

1. *perfect if τ satisfies* $\mathbf{CONS_{L+C+T+G}} + \mathbf{KEEP_G} + \mathbf{CLOS}_{pred} + \mathbf{REF}$ *and*
2. *forward-perfect if τ satisfies* $\mathbf{CONS_{L+C+T+G}} + \mathbf{KEEP}^{\rightarrow}_{\mathbf{G}} + \mathbf{CLOS}^{\rightarrow}_{pred} + \mathbf{REF}$.

This result generalizes Theorem 5.2: again, we obtain a perfect heuristic under a weaker condition than $\mathbf{CONS_S}$, and the required condition is weaker again if we only need a forward-perfect heuristic.

8.2.5 RELATIONSHIP TO BÄCKSTRÖM AND JONSSON

In Section 3.3, we discussed how our results on heuristic properties like admissibility and exactness relate to the work of Bäckström and Jonsson (2013). In this section we extend our results to transformations with non-total state mappings and to weaker heuristic properties focusing on reachable states.

Bäckström and Jonsson do not consider distinguished initial and goal states and hence do not have a concept of "reachable states". However, they do discuss and prove some results for a variant of admissibility that only applies to finite heuristic values (roughly speaking, if the heuristic value is finite, it must be admissible, but inadmissible infinite estimates are permitted), which is at least similar in spirit to some of the generalizations we study in this section. Specifically, they describe conditional properties based on such infinite estimates as a way to model non-total state mappings. Despite these similarities, there are significant differences between their formalization and ours, and we do not see any further interesting connections beyond what was already discussed in Section 3.3.

8.3 Properties of Prune Transformations

We are now ready to study the properties of prune transformations, the last result in this section. Theorem 21 shows under which conditions the new transformation properties introduced in this section compose, and Theorems 22 and 23 show which transformation properties lead to desirable heuristic properties. As a last piece of the puzzle, we now show which restrictions must be placed on the set of pruned (or kept) states in order to achieve these desirable properties in a shrink transformation as well as the other transformation properties introduced earlier.

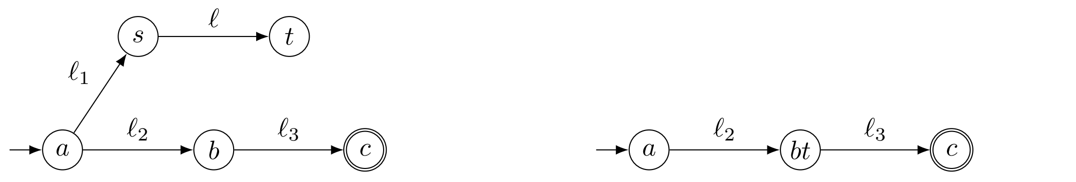
Theorem 24. *Let F be a factored transition system, let $\Theta_k \in F$, and let S^k be the states of Θ_k. The prune transformation τ_F of F for K and Θ_k has the following properties depending on K:*

1. *for all K: τ_F satisfies* $\mathbf{CONS_{L+C+T+I+G}} + \mathbf{IND} + \mathbf{REF} + \mathbf{LOC_{\geq}}$.
2. *for $K = S^k_{\leftarrow}$: τ_F additionally satisfies* $\mathbf{KEEP_G} + \mathbf{CLOS}_{pred} + \mathbf{LOC_{=}}$.
3. *for $K = S^k_{\rightarrow}$ or $K = S^k_{\leftrightarrow}$: τ_F additionally satisfies* $\mathbf{KEEP^{\rightarrow}_G} + \mathbf{CLOS}^{\rightarrow}_{pred}$.

The first result can be briefly summarized as: prune transformations always satisfy *all* desirable transformation properties from the earlier sections except $\mathbf{CONS_S}$. (If we can guarantee admissibility, $\mathbf{LOC_{\geq}}$ is generally a desirable property because it leads to stronger heuristics.) It almost goes without saying that prune transformation do *not* in general satisfy $\mathbf{CONS_S}$, as their only purpose is to prune states. Formally, the only prune transformations that satisfy $\mathbf{CONS_S}$ are those where K consists of all states of Θ_k, in which case the prune transformation does nothing.

The second and third result show under which conditions we obtain the weaker versions of $\mathbf{CONS_S}$ from Definition 37. If we keep exactly the set of backward-reachable states, we get the properties $\mathbf{KEEP_G}$ and $\mathbf{CLOS}_{pred}$. These properties are sufficient for obtaining admissible or even perfect heuristics in the merge-and-shrink framework. (Of course, admissibility or perfection can be lost if other involved transformations lack the required properties.) We also get all properties that are necessary for $\mathbf{KEEP_G}$ and $\mathbf{CLOS}_{pred}$ to compose.

If we keep exactly the set of forward-reachable or the set of alive states, the resulting heuristics will not generally be admissible, as pruned states lead to infinite heuristic values. However,



(a) Transition system Θ with backward-unreachable states s and t.

(b) Transformed transition system Θ' after applying transformation τ.

Figure 19: Example transformation τ of transition system Θ into transition system Θ' that only prunes the backward-unreachable state s, but not t, which is combined with the backward-reachable state b.

we still get the properties $\mathbf{KEEP}^{\rightarrow}_{\mathbf{G}}$ and $\mathbf{CLOS}^{\rightarrow}_{\mathit{pred}}$ required for forward-admissibility and forward-perfection, as well as all conditions required for these properties to compose. Therefore, the default pruning strategy of the merge-and-shrink implementations in the Fast Downward planner (Helmert, 2006) only keeps the alive states (i.e., prunes all dead states) with the rationale that smaller factors are preferable. The MIASM merge strategy (Fan et al., 2014) actively prefers merging two factors such that their product contains many dead states, which are then subsequently removed by a prune transformation.

The results of this section also help explain why these more aggressive forms of pruning are problematic in scenarios that go beyond a vanilla A* search, such as the orbit search algorithm mentioned in the introduction to this section (Domshlak et al., 2015): for orbit search, forward-admissibility is not sufficient to guarantee optimality, while full admissibility is. A similar comment applies to the certificates of unsolvability by Eriksson, Röger, and Helmert (2017, footnote 4), which require full consistency (not just forward-consistency) to form so-called inductive sets.

An important and perhaps surprising subtlety to Theorem 24.2 is that it is important that *all* states that are not backward-reachable are pruned (and analogously for Theorem 24.3). If we prune some but not all of the backward-unreachable states, the $\mathbf{CLOS}_{\mathit{pred}}$ property is no longer guaranteed, and as a consequence we can obtain an inconsistent heuristic when composing this prune transformation with other transformations.

Figure 19 shows an example transformation $\tau = \langle \Theta', \sigma, \lambda \rangle$ of a transition system Θ into a transition system Θ'. All labels have unit cost. Transformation τ is the composition of two transformations: first, we prune the backward-unreachable state s (but *not* the other backward-unreachable state t), and then we shrink, mapping b and t to the same abstract state bt. We have $h^\tau(s) = \infty$ (because $s \notin dom(\sigma)$) and $h^\tau(t) = h^*_{\Theta'}(\sigma(t)) = h^*_{\Theta'}(bt) = 1$, which violates the consistency condition $h^\tau(s) \leq c(\ell) + h^\tau(t)$ for the transition $s \xrightarrow{\ell} t$.

9. Discussion and Related Work

At this point, we have concluded our presentation of the transformation-based merge-and-shrink framework. As mentioned in the introduction, the purpose of this paper is to develop a solid and extensible theory of merge-and-shrink, both to consolidate and unify existing work and to provide a firm foundation for future developments. In the interest of maintaining focus, this paper is not concerned with experimental evidence of the utility of merge-and-shrink, which has already been established in the literature (e.g., Dräger et al., 2006; Helmert et al., 2007; Helmert, Haslum, &

Hoffmann, 2008; Nissim et al., 2011; Hoffmann et al., 2014; Sievers et al., 2014; Fan et al., 2014; Sievers et al., 2015; Torralba & Hoffmann, 2015; Torralba & Kissmann, 2015; Sievers et al., 2016; Eriksson et al., 2017; Fan, Müller, & Holte, 2017; Eriksson, Röger, & Helmert, 2018; Fan, Holte, & Müller, 2018; Sievers, Pommerening, Keller, & Helmert, 2020). Instead, this last section before the conclusions highlights applications of our theory, both in relation to previously published work and in a wider context including possible future research directions.

We structure the discussion into three parts: firstly, we discuss the most common application of merge-and-shrink as a basis of heuristics for factored state-space search. Secondly, we discuss applications of the framework and extensions that go beyond the use as heuristics, but stay within the wider context of AI planning and related factored state-space search problems. Finally, we discuss connections to other areas of AI and computer science, such as constraint programming and automata theory.

9.1 Merge-and-Shrink for Computing Heuristics

The merge-and-shrink framework (although not under this name, which was suggested by Carmel Domshlak in personal communications) and its use for computing heuristics were originated by Dräger et al. (2006, 2009) for the purpose of finding error traces in model-checking problems. They introduce composition (merge) and abstraction (shrink) operations and mention that heuristics derived from such transformations are admissible and consistent, which we show for our more general framework in Theorem 3 (page 792). Their (non-linear) merge strategy prefers merging factors that need to "synchronize" close to a goal state. They use a shrink strategy that prefers to combine bisimilar states and switches to more radical h-preserving shrinking when bisimulation cannot sufficiently reduce the size of the factor. We theoretically justify h-preserving shrinking in Theorem 8 (page 815) and bisimulation-based shrinking in Theorem 9 (page 817). They do not use label reduction or pruning.

9.1.1 Development of the Merge-and-Shrink Framework in Classical Planning

Helmert et al. (2007, 2008) adapt merge-and-shrink to classical planning. They show that merge-and-shrink abstractions are strict homomorphisms (i.e., conservative and induced). In our theoretical development, this is a consequence of the properties of shrinking (Theorem 7, page 815) and merging (Theorem 12, page 821) and the compositionality of transformation properties (Theorem 1 on page 791). They also emphasize the importance of suitable merge and shrink strategies and provide the first discussion of the representational power of merge-and-shrink abstractions, in particular showing that they generalize projections, the form of abstractions underlying pattern database heuristics (Culberson & Schaeffer, 1998). They consider a linear merge strategy based on the goals and causal graph of the underlying planning task representation and introduce f-preserving shrinking. As discussed in the previous two sections, their implementation also includes a limited form of label reduction (HHH label reduction, see Section 7.4) and pruning of dead states, but these two improvements are not discussed in the papers.

Nissim et al. (2011) study bisimulation-based shrinking in more depth. They provide the first description of HHH label reduction and show that bisimulation-based shrinking combined with HHH label reduction leads to perfect heuristics. They demonstrate that such perfect heuristics can be computed in polynomial time in several standard benchmark domains from classical planning. In our theory, Theorem 9 (page 817) shows that bisimulation-based shrinking is exact, and Theorem 20 (page 830) shows that Θ-combinable label reduction is exact. HHH label reduction is a special case of Θ-combinable label reduction. Of course, merging is also exact. Combining these theorems, our theory generalizes their result on perfect heuristics. Note that HHH label reduction is *not* compositional, and therefore Nissim et al. cannot argue in the same way: their theorems must explicitly discuss the interactions of merging, label reduction and bisimulation-based shrinking. Due to the limitations of HHH label reduction, their results only apply to linear merge strategies. Specifically, their merge strategy follows the variable order of the *causal graph heuristic* (Helmert, 2004). Nissim et al. only show that use of bisimulation is sufficient for shrinking to be exact; we show that shrinking is exact iff it is based on bisimulation (Theorem 11 on page 818). (Note also that the definition of "exact" in this paper differs from earlier definitions, which were either not fully formal or non-compositional.)

The next important development in the theory of merge-and-shrink abstractions is the journal paper by Helmert et al. (2014), which synthesizes and extends the papers by Helmert et al. (2007, 2008) and Nissim et al. (2011). As discussed in Section 7.4, one of the major novel contributions is the generalization of HHH label reduction to HHHN label reduction, which is not restricted to linear merge strategies but still lacks compositionality and can only be applied in limited ways. We refer to Section 7.4 for a more detailed discussion. The journal paper is also the first work on merge-and-shrink abstractions that attempts to capture the general class of systems for which the framework is applicable. Our use of the term *factored transition systems* (but not our exact definition) stems from this paper. Moreover, it is the first work on merge-and-shrink to provide a real discussion of the representation of the state mapping, which previous papers left implicit. Helmert et al.'s discussion of *merge-and-shrink trees* and *cascading tables* is a precursor to our discussion of factored mappings (Section 4.3). The paper also establishes that merge-and-shrink heuristics are incomparable to the h^2 critical path heuristics (Haslum & Geffner, 2000) within the heuristic compilation framework of Helmert and Domshlak (2009). Finally, the paper includes some expressiveness results, which we discuss in Section 9.3.

The last important step in the previous development of the merge-and-shrink framework is the conference paper by Sievers et al. (2014), which introduces the general form of label reduction we use in this work (Section 7). While label reduction is not one of the two eponymous transformations in the merge-and-shrink framework, the conference paper is a critical step in the evolution of merge-and-shrink because it first enabled the development of a clean compositional theory, without which the transformation-based view underlying this paper would not be possible. See Section 7.4 for a detailed discussion of the relationship between the results on label reduction in this paper and the results of the conference paper. Sievers et al. also give experimental evidence showing that label reduction is critical to the performance of planning algorithms based on merge-and-shrink abstractions. The new form of label reduction finally allowed fully combining label reduction with non-linear merge strategies. Sievers et al. introduce the non-linear *Dräger, Finkbeiner, Podelski (DFP)* merge strategy based on the early work by Dräger et al. (2006) and show that it outperforms the previous state of the art.

9.1.2 MERGE STRATEGIES, SHRINK STRATEGIES, GENERAL STRATEGIES

The generalized theory of label reduction prompted several follow-up papers studying different (non-linear) merge strategies. Fan et al. (2014) introduce the *undirected min-cut (UMC)* strategy based on minimum cuts in the (weighted) *causal graph* of the underlying planning task and the *maximum intermediate abstraction size minimizing (MIASM)* strategy, which searches in the space of possible merges trying to minimize the size of intermediate factors in the merge tree under exact shrinking. Sievers et al. (2015) devise merge strategies that augment an existing merge strategy by reasoning about symmetries of factored transition systems. Their core theoretical result is a connection between such symmetries and bisimulation, showing that symmetries can drive the merge strategy to obtain more opportunities for exact shrinking. Sievers et al. (2016) systematically study merge strategies by computing all merge strategies on small planning tasks and randomly sampling merge strategies on larger ones. The analysis shows that the merge strategies in the literature leave substantial room for improvement. They also suggest a new variant of the MIASM strategy, tie-breaking enhancements of merge strategies, and a new non-linear merge strategy based on the strongly connected components of the causal graph of a planning task.

Katz et al. (2012) introduce a shrink strategy based on a relaxed notion of bisimulation that restricts the bisimulation properties to a subset K of labels in order to obtain more shrinking while maintaining a perfect heuristic. Testing if a given label set K satisfies their requirements is PSPACE-complete, so they study approximations. Fan et al. (2018) make two contributions. Firstly, they present a fast to compute variant of merge-and-shrink with a random merge strategy, maximal h-preserving shrinking and no label reduction. Secondly, they introduce a further refinement of the MIASM strategy building on the MIASM variant by Sievers et al. (2016).

Sievers (2018) adds a time limit to the merge-and-shrink algorithm. In all previous work, the overall planning algorithm simply failed if the merge-and-shrink computation took too much time. If the algorithm terminates with multiple factors, the max-factor heuristic (Definition 22, page 810) is used. The paper otherwise focuses on how to implement the merge-and-shrink framework efficiently and gives a description of the *general strategy* (Section 4.4.4) used in the merge-and-shrink implementation of the Fast Downward planner.

9.2 Merge-and-Shrink Beyond Heuristics

We now discuss related work that uses the merge-and-shrink framework in the context of state-space search, but includes aspects other than the use as a heuristic. Specifically, we consider merge-and-shrink for proving unsolvability, integration with symbolic search, cost partitioning over merge-and-shrink heuristics, the computation of dominance relations, the use of merge-and-shrink as a task reformulation framework, and transformations and properties that go beyond the ones considered in this paper.

9.2.1 UNSOLVABILITY

Hoffmann et al. (2014) use the merge-and-shrink framework to prove the unsolvability of planning tasks (e.g., Bäckström, Jonsson, & Ståhlberg, 2013; Muise & Lipovetzky, 2016). Their approach is equivalent to running an A* search on a modified state space where all label costs are changed to 0, in which case an abstraction heuristic only assigns heuristic values of 0 or ∞. Building on the shrink strategies by Katz et al. (2012), they consider a relaxation of exact shrinking called "safe shrinking" that only aims to preserve the existence of plans for a given state, but not the plan cost. In the terminology of this paper, they identify Θ-combinable labels and add direct transitions for sequences of transitions induced by such labels. This introduces strongly connected components of transitions in the factors that lead to more opportunities for bisimulation-based shrinking. They also experimentally evaluate a wide variety of linear merge strategies based on the goals and causal graph of a planning task and on a notion of synchronization similar to the one used by Dräger et al. (2006).

Because Hoffmann et al. work with an older, non-compositional theory of merge-and-shrink abstractions, they have to explicitly consider how safe shrinking interacts with other merge-and-shrink transformations. In our framework, their results could be more easily established by introducing safety as a new compositional property and showing that the required transformations satisfy it. Because safety is a relaxation of exactness, nothing new needs to be shown for exact transformations.

Eriksson et al. (2017, 2018) study the use of merge-and-shrink abstractions (among other techniques) for *certified* unsolvability in classical planning, i.e., for a setting where the output of a planning system on an unsolvable task must include a computer-verifiable proof that no plan exists. The approach covers all conservative transformations we discuss except non-linear merging.

9.2.2 SYMBOLIC SEARCH AND SPM&S

There are several papers that connect the merge-and-shrink framework to data structures used in symbolic search, in particular binary decision diagrams (BDDs; Bryant, 1986) and algebraic decision diagrams (ADDs; Bahar, Frohm, Gaona, Hachtel, Macii, Pardo, & Somenzi, 1997).[11] Blai Bonet first pointed out the close relationship between the factored mappings used to represent merge-and-shrink heuristics with linear merge strategies and such symbolic data structures (personal communications, 2008).

This connection is formally discussed by Edelkamp, Kissmann, and Torralba (2012), who note that ADD/BDD reductions can further compress factored mappings representing linear merge-and-shrink heuristics. They also observe that BDD representations of linear merge-and-shrink heuristics have a suitable form for the symbolic BDDA* (Edelkamp & Reffel, 1998) algorithm, a variant of A* that can process sets of states at a time by representing them as BDDs. They explore this idea experimentally with several linear merge strategies, including novel ones based on the BDD variable ordering strategy of the GAMER planner (Kissmann, 2012).

Torralba, Linares López, and Borrajo (2013) introduce fully symbolic algorithms based on the merge-and-shrink framework, originally called *symbolic merge-and-shrink* and renamed to *symbolic perimeter merge-and-shrink* (SPM&S) in later papers. The key idea is to interleave a symbolic backward search over the state space, as performed by traditional symbolic search planners (e.g., Edelkamp & Helmert, 2001; Kissmann, 2012; Kissmann, Edelkamp, & Hoffmann, 2014), with merge-and-shrink abstraction. As in the regular merge-and-shrink framework, their algorithm maintains a factored transition system that originally consists of all atomic factors, representing the (unabstracted) original transition system. Merge and shrink transformations on this factored transition system are interleaved with symbolic search steps that explore the state space under the current abstraction. Roughly speaking, an additional merge step followed by a shrink step happens whenever the representation becomes too large to perform another symbolic search step.

An important aspect of SPM&S is that it only performs a single symbolic exploration of the state space. Whenever the merge-and-shrink abstraction needs to be abstracted in order to continue the symbolic exploration, the exploration process continues with the current sets of open and closed states adjusted to account for the modified abstraction. This effectively results in an abstraction hierarchy, where states close to the goal are not abstracted at all, and the fidelity of the abstraction decreases as goal distance increases.

11. Throughout this paper, by BDDs and ADDs, we refer to reduced ordered BDDs and reduced ordered ADDs.

When symbolic exploration finishes, SPM&S returns an admissible heuristic, represented as a set of BDDs in a form suitable for symbolic search algorithms like BDDA*. By imposing size limits on BDDs generated as intermediate results and tailoring the (necessarily linear) merge strategy to the variable ordering used by the symbolic data structures, SPM&S can guarantee upper bounds on the representation size of the resulting heuristic, similarly to the upper bounds on the size of factors imposed by most work using the regular (non-symbolic) merge-and-shrink framework. In cases where no abstraction is necessary within the given computational bounds, SPM&S is equivalent to a standard (non-heuristic) symbolic regression search.

More elaborate accounts of the same approach are given in Torralba's PhD thesis (2015) and in a journal paper (Torralba et al., 2018). Torralba, Linares López, and Borrajo (2016) generalize the approach to a bidirectional search setting. Experimental results show strong performance for SPM&S, which is one of the ingredients of the successful SymBA* planner (Torralba, Alcázar, Borrajo, Kissmann, & Edelkamp, 2014). However, the available evidence suggests that the heuristic search aspects of SymBA* only play a minor role in its overall performance, and its blind symbolic search configurations perform best in many common benchmarks (Torralba, 2015).

9.2.3 COST PARTITIONING

Fan et al. (2017) consider the use of merge-and-shrink abstractions within the framework of *cost partitioning* (Katz & Domshlak, 2010). Cost partitioning is a conceptual framework for the additive combination of admissible heuristics. The key idea is to assign shares (*cost partitionings*) of the label costs to different copies of the transition system in such a way that the sum of admissible heuristics derived for each copy is an admissible heuristic. In general, cost partitioning for merge-and-shrink abstractions requires reasoning over the internal structure of the abstractions in order to derive suitable cost partitionings, so this goes beyond the simple use of a merge-and-shrink abstraction as a (black-box) heuristic.

However, the approach of Fan et al. does not need to look "inside" merge-and-shrink abstractions, as its cost partitioning is determined entirely by the original cost function. Their motivating observation is that merge-and-shrink abstraction benefits from cost functions with as few different label costs as possible because this affords additional opportunities for exact label reduction (Theorem 15, page 827), which in turn affords additional opportunities for exact shrinking by bisimulation (Theorem 9, page 817). They identify the lowest non-zero label cost $c_{\min}$ and split off a label cost function that assigns cost 0 to all original zero-cost labels and cost $c_{\min}$ to all other labels. All remaining costs are then recursively partitioned in the same way until only zero-cost labels remain. They then compute a regular merge-and-shrink abstraction heuristic for each of the cost functions and use their sum as the final heuristic.

Sievers et al. (2020) describe a tighter integration of cost partitioning into the factored transformation framework. Their approach collects all factors that are computed as intermediate results by the transformations (Algorithm 1, page 809) and computes a cost partitioning over their factor heuristics. Cost partitioning also offers a better solution than the max-factor heuristics used by Sievers (2018) in cases where it is prohibitively expensive to apply transformations until only a single factor remains.

They also study the interaction of different kinds of transformations with the properties of cost partitioning, for example showing that an optimal cost partitioning (Katz & Domshlak, 2010) never needs to consider a factor obtained by shrinking the factor Θ if it already considers Θ. Their algorithm exploits these relationships to limit the set of factors to consider, even though for efficiency reasons they primarily use the non-optimal *saturated cost partitioning* algorithm (Seipp et al., 2020), which does not give the same guarantees. A useful property of saturated cost partitioning is that it can be performed online in the sense that it processes the factors for the cost partitioning one at a time, and after each factor is processed, its transition information can be discarded. This comes at a price of heuristic quality compared to offline cost partitioning, where full representations of all factors need to be in memory simultaneously. Sievers et al. (2020) consider both online and offline versions of their approach, as well as versions that maximize over multiple cost partitionings.

9.2.4 DOMINANCE RELATIONS

Torralba and Hoffmann (2015) use concepts of the merge-and-shrink framework to define notions of *dominance* between states of a factored transition system. Roughly speaking, state s (weakly) dominates state s' if everything useful that can be achieved from s' can also be achieved from state s. In particular, for every solution from state s' there is a corresponding solution of the same or lower cost from s. Weak dominance defines a weak partial order over the states. If a search algorithm reaches a state that is dominated by a previously considered state reached at the same or lower cost, the dominated state can be pruned.

Their approach is based on the concept of *simulation*, which is "one half" of bisimulation (Definition 25, page 816). In more detail, their notion of simulation is analogous of property **BISIM2**, and the additional requirement for *goal-respecting* simulations is analogous of property **BISIM1**. They then show how goal-respecting simulations for factors in a factored transition system induce goal-respecting simulations for their products. This can be viewed as a compositional approach for computing simulation relations for factored transition systems. In order to obtain more general dominance relations, the notion of dominance between states is augmented by a notion of dominance between labels, which essentially combines goal-respecting simulations with a form of label dominance analogous to our notion of locally subsuming labels (Definition 31, page 828).

In addition to informing the theoretical development of the paper, the merge-and-shrink framework is also used in the experimental evaluation. Torralba and Hoffmann do not use the given factored transition system as the input to their dominance detection algorithm, but first apply a sequence of exact merge-and-shrink transformations (merging using the DFP strategy, bisimulation-based shrinking and exact label reduction) because this leads to dominance relations affording more pruning.

Torralba and Kissmann (2015) extend this work by considering other uses of dominance relations within the merge-and-shrink framework. They introduce the notion of a *subsumed transition* of a factor. Roughly speaking, a transition is subsumed if it can be safely removed without affecting solution costs in the product. One of their contributions is to use the previously developed simulation relations and label dominance relations to determine subsumed transitions. Another contribution is *simulation-based shrinking*, which generalizes bisimulation-based shrinking (Theorem 9, page 817).

Because simulation-based shrinking combines some states that bisimulation-based shrinking does not, it cannot be exact (Theorem 10, page 818). However, pruning of subsumed transitions and simulation-based shrinking are both *globally h-preserving*, i.e., result in the perfect heuristic when only combined with exact transformations. This shows that there are weaker properties than exactness that may still be as useful as exactness in many contexts, similar to our relaxations of exactness that give rise to forward-perfect heuristics (Theorem 23, page 837). Being globally h-preserving is not presented as a compositional property, and therefore Torralba and Kissmann carefully consider the interaction of the two new transformations with other merge-and-shrink transformations. Indeed, combining pruning of subsumed transition with pruning of unreachable states (Section 8) can result in heuristics that are not even forward-admissible. However, the A* algorithm will still find optimal solutions with such heuristics.

Torralba and Kissmann consider two applications of the new transformations: firstly, they use them directly in a regular merge-and-shrink heuristic computation to obtain heuristics for A*. Secondly, they use a merge-and-shrink computation with their new transformations as a preprocessing step to determine irrelevant operators in a planning task: if a transformation results in dead labels (Definition 26, page 818) in any factor of the factored transition system, the corresponding operators can be safely pruned from the planning task. This pruning of operators can be used together with any planning algorithm, and Torralba and Kissmann show good results for several planning algorithms that are unrelated to the merge-and-shrink framework.

9.2.5 Task Reformulation

Using merge-and-shrink for operator pruning as in the work of Torralba and Kissmann can be viewed as a special case of the more general idea of using merge-and-shrink transformations for *task reformulation*. This idea is the topic of a paper by Torralba and Sievers (2019), who argue that the factored transition system representation used in the merge-and-shrink framework is useful as a planning task representation in general, not just for the purposes of deriving heuristics.

In this view, exact merge-and-shrink transformations correspond to task reformulations that preserve the existence and cost of solutions. Together with a mechanism for converting solutions of the transformed representation back into the original representation, such task reformulations can be combined with arbitrary planning algorithms. Torralba and Sievers also consider non-exact transformations that preserve the existence but not necessarily the cost of solutions. These can be used as task reformulations for satisficing planning. This idea builds a bridge to previous work on reducing the accidental complexity of planning tasks (Haslum, 2007), to the notion of safe abstraction (Wehrle & Helmert, 2009) and to planning task reformulation (Tožička, Jakubův, Svatoš, & Komenda, 2016). Additional connections exist to the safe shrinking techniques for unsolvability discussed in Section 9.2.1 and to a long line of research on refining abstract solutions (e.g., Knoblock, 1994; Bacchus & Yang, 1994; Bäckström & Jonsson, 2012b), and of course to the work by Bäckström and Jonsson (2013) discussed throughout this paper.

Besides using merge-and-shrink as a task transformation tool that can be wrapped around an arbitrary planning algorithm, Torralba and Sievers also modify existing search-based planning algorithms to work directly with arbitrary factored transition systems. Compared to the more commonly used SAS$^+$ representation, such representations can be more compact and have additional expressiveness that can be viewed as limited forms of conditional effects and angelic nondeterminism.

9.2.6 Other Transformations and Properties

We conclude our overview of related work in state-space search by mentioning transformations of factored transition systems (and their properties) not covered in this paper. Some of these have not been directly influenced by the merge-and-shrink literature, but could be recast in our transformation framework. We believe that developing these connections could improve our understanding of these techniques while also adding to the the merge-and-shrink toolbox.

We already discussed some such transformations and properties throughout this section. Examples include the safe shrinking of Hoffmann et al. (2014), the dominance pruning of Torralba and Hoffmann (2015), the simulation-based shrinking and label pruning of Torralba and Kissmann (2015) and the solution-preservation transformations of Torralba and Sievers (2019). As discussed above, some of these transformations and the underlying properties are closely related to the transformations we studied in depth. For example, the property of being safe in the sense of Hoffmann et al. (2014) or solution-preserving in the sense of Torralba and Sievers (2019) is a weaker form of the **REF** property we studied. Similar notions of refinability are a major focus of the work of Bäckström and Jonsson (2013). Label pruning is directly analogous to (state) pruning discussed in Section 8.

Wehrle, Sievers, and Helmert (2016) discuss an algorithm for splitting a state variable of a planning task into multiple state variables whose joint behavior is equivalent to the behavior of the original variable. In other words, they introduce a *factoring* transformation, which is in some sense the opposite of our merge transformation (Section 6). The relationship is not exact: their factoring applies to a so-called Cartesian product of transition systems, with different semantics from the synchronized product underlying merging. Despite this difference, their work could be easily recast within the merge-and-shrink framework. It appears that the notion of factoring by Wehrle et al. is “almost” exact, with some technicalities related to auxiliary labels introduced by factoring.

Röger, Pommerening, and Helmert (2014) introduce the idea of *context splitting* for classical planning tasks. Context splitting can be naturally viewed as a transformation in the merge-and-shrink framework and is essentially the converse of label reduction (Section 7). Like factoring, a complication is that a single original label can correspond to multiple transformed labels, which would require some extensions of our theory. Apart from this aspect, context splitting is again an exact transformation.

Fluent merging in classical planning is essentially the same idea as the merge transformation for factored transition systems. It was introduced by van den Briel et al. (2007a, 2007b) for a planning algorithm based on integer programming and revisited by Seipp and Helmert (2011) to improve the quality of heuristics for satisficing search, especially ones based on delete relaxation. Bonet and van den Briel (2014) use fluent merging to strengthen heuristics based on network flows, which are closely related to the integer programming approach of van den Briel, Vossen, and Kambhampati (2005). They emphasize two distinct variants of fluent merging: one where two state variables of a planning task are *replaced* by their product and one where the product is *added* to the existing state variables. The former directly corresponds to the merge transformation studied in this paper. The latter makes it possible to construct multiple overlapping products, for example merging factors Θ_1 and Θ_2 to obtain Θ_{12} while also merging Θ_1 and Θ_3 to obtain Θ_{13}. In the merge-and-shrink literature, such overlapping merging has been discussed hypothetically under the name of *non-orthogonal merging* (Helmert et al., 2007, 2014) but is not covered by the existing theory or implementations. A related technique somewhat more distant from the merge-and-shrink literature is the tracking of conjunctions of state variables in the h^m and h^C family of heuristics (e.g., Haslum & Geffner, 2000; Haslum, 2009, 2012; Keyder, Hoffmann, & Haslum, 2012; Fickert, Hoffmann, & Steinmetz, 2016).

Non-orthogonal merging could be accommodated in the merge-and-shrink framework in two different ways: either by adding a new additive merge transformation, or more minimalistically by adding a new *clone* transformation that adds a copy of a factor to the factored transition system. An additive merge transformation can then be expressed by cloning the two factors to be merged and then merging the clones. This view is attractive because of its minimalism: we already understand the properties of the existing merge transformation, and hence we only need to analyze the (very simple) clone transformation and appeal to the compositionality of the framework in order to understand non-orthogonal merging.

It is easy to see that such a clone transformation is not state-induced, i.e., its state mapping is not surjective. It is also easy to see that it does not have the properties $\mathbf{IND_T}$, $\mathbf{IND_I}$ or $\mathbf{IND_G}$ – all this directly follows from the observation that the transformed factored transition system can have more states (transitions, initial states, goal states) than the original system. However, cloning is conservative and refinable. To attempt an intuitive explanation, cloning only adds spurious states, as well as spurious transitions related to these spurious states. The spurious states and their transitions do not affect refinability precisely because the states are spurious and hence do not have a preimage.

Domshlak, Katz, and Lefler (2010, 2012) transform planning tasks by *enriching* the original set of state variables with additional variables that track the achievement of *landmarks* (Hoffmann, Porteous, & Sebastia, 2004). The hope is that heuristics computed on such landmark-enriched planning tasks may be more informative than the same heuristics computed on the original task. Seipp and Helmert (2014, 2018) present a technique for capturing landmark information within the framework of strictly homomorphic abstract transition systems. Landmark-enriching can be modeled in the merge-and-shrink framework by cloning factors that contribute to a given landmark, and then using the approach of Seipp and Helmert to capture the landmark information.

Finally, *cost partitioning* (Section 9.2.3) could conceivably be represented as a transformation within the merge-and-shrink framework. One perspective of cost partitioning is that it creates multiple clones of the factored transition system, replacing each factor Θ by clones $\Theta^1, \dots, \Theta^n$. These clones are then *decoupled* by giving each clone its own version of the labels. Essentially, label ℓ is replaced by labels $\ell_1, \dots, \ell_n$ such that ℓ_i behaves like ℓ in the i-th clone of each factor but does not affect the other clones (induces self-loops in all their states). This results in essentially independent copies of the factored transition system, connected only by the *cost partitioning constraints* $c(\ell_1) + \cdots + c(\ell_n) \leq c(\ell)$. While this view offers no immediate advantage over the current view of cost partitioning, it may be a useful starting point for ideas that fall in between the maximally loose coupling of cost partitioning and the maximally tight coupling of merge transformations. For example, one could conceive of variants of cost partitioning that loosely couple some labels but tightly couple others. More generally, cost partitioning for abstraction heuristics can be considered a special case of *operator-counting constraints* (Pommerening, Röger, Helmert, & Bonet, 2014; Pommerening, Helmert, Röger, & Seipp, 2015), and it would be interesting to explore the connections between the merge-and-shrink framework and operator counting more deeply.

9.3 Beyond Planning

As the last part of our literature discussion, we widen the scope again and discuss connections of the merge-and-shrink framework to other areas of computer science, both within and outside of artificial intelligence.

The merge-and-shrink framework is essentially automata-theoretic. Individual transition systems correspond to finite-state automata, and merge transformations correspond to computing product automata. Finite-state automata capture the regular languages, and in this view synchronized products correspond to language intersection. Because the semantics of a factored transition system is given by the product of its factors, finding plans is the same problems as finding words in the intersection of regular languages. Kozen (1977) proved that the problem of testing whether the language intersection of n finite automata is empty is PSPACE-complete. Taking into account the equivalence of the propositional STRIPS and SAS^+ planning formalisms, this result already showed that classical planning in such formalisms is PSPACE-complete, predating the seminal work of Bylander (1994) by almost two decades. Similar connections to automata theory can be drawn for other parts of the merge-and-shrink framework. In particular, there are close connections between bisimulation-based shrinking and automata minimization, which can be traced back to the work of Moore (1956).

Automata-based approaches are very common in the area of computer-aided verification, specifically for the problem of model checking (Clarke et al., 1999, Chapter 9). Systems under analysis are often modeled as communicating automata. Properties to be verified are typically expressed in a temporal logic such as LTL (Pnueli, 1977) or CTL (Clarke, Emerson, & Sistla, 1986), which can also be converted into automata. A system then satisfies a given property if all words corresponding to possible behaviors of the system satisfy the property. This is essentially a language entailment check that can be tested by a combination of automata complementation, intersection, and emptiness testing. Against this background, it perhaps comes as no surprise that the merge-and-shrink framework originates from model checking, as discussed before (Dräger et al., 2006, 2009). Clearly, fruitful interactions between merge-and-shrink and computer-aided verification are possible in both directions. For example, the concepts of bisimulation in this paper and earlier works on merge-and-shrink and of simulation in the work of Torralba and Hoffmann (2015) both originate from the area of computer-aided verification.

In the constraint programming community, Pesant (2004) introduced the `regular` global constraint, which encodes that a given sequence of symbols (represented as a sequence of variables of the constraint program) is accepted by a given finite automaton. In its basic form, the constraint applies to fixed-length sequences and deterministic automata, but generalizations have been described in the literature, including ones incorporating notions of cost (e.g., Demassey, Pesant, & Rousseau, 2006). Using a set of such global constraints, one per factor of a factored transition system, it is easy to express that an assignment to the variables $V_1, \dots, V_n$ must correspond to the operators in a length-n plan for the factored transition system. The connection to classical planning and hence factored state-space search in general was first explored in a workshop paper by Zanarini, Pesant, and Milano (2006). In recent years, these connections between the planning and constraint programming research communities have received increasing attention (e.g., Babaki, Pesant, & Quimper, 2020), partially inspired by the development of the merge-and-shrink framework in the classical planning community (Beck, Magazzeni, Röger, & van Hoeve, 2018, Section 4.3).

Finally, factored mappings[12] (Section 4.3) can be viewed as a general knowledge compilation formalism in the sense of Darwiche and Marquis (2002). We already partially discussed the relationship to BDDs and ADDs above. Helmert et al. (2014) showed a form of equivalence between BDDs and factored mappings with linear merge trees. They also showed that non-linear factored mappings can be converted to BDDs with a (super-polynomial, but subexponential) representational increase from $\|\sigma\|$ to $O(\|\sigma\|^{\Theta(\mathrm{HS}(\sigma))})$, where $\mathrm{HS}(\sigma)$ is the *Horton-Strahler number* (Horton, 1945) of the merge tree underlying σ, which is in turn within a multiplicative constant of the *pathwidth*

12. For brevity, in the following we silently assume that the factored mappings considered are orthogonal.

of the merge tree (Helmert et al., 2015, footnote 3). Helmert et al. (2015) later showed that this bound is tight, proving that factored mappings are strictly more compact than BDDs. This suggests that factored mappings might be useful as a knowledge representation formalism beyond the use in the merge-and-shrink framework. Intuitively, the increase in compactness can be explained by the ability to partition a mapping with respect to a variable *tree*, which generalizes the variable *orders* underlying BDDs. In the knowledge compilation literature, *Deterministic Decomposable Negation Normal Form* (d-DNNF, Darwiche, 2001) and *Sentential Decision Diagrams* (Darwiche, 2011) are also generalizations of BDDs that are based on variables trees rather than variable orders. So far, the relationship between factored mappings and these two formalisms has not been explored in depth. The same is true for the computational properties of factored mappings as a knowledge compilation formalism.

10. Conclusions

We reframed the merge-and-shrink framework, which dates back to work by Dräger et al. (2006) on deriving abstraction heuristics for model-checking problems, as a compositional theory of transformations of factored transition systems. Our work cleans up and generalizes earlier attempts to develop a theory of merge-and-shrink abstractions in the papers by Helmert et al. (2007, 2014) and Sievers et al. (2014). As a consequence of these generalizations, we no longer view merge-and-shrink primarily as a mechanism for deriving abstraction heuristics, but more generally as a toolbox and theory for transforming and reasoning about factored transition systems. Recent work in the literature, such as Torralba and Hoffmann's (2015) paper on dominance pruning and Torralba and Sievers's (2019) paper on task reformulation are examples that already embody this research direction. We believe that the theory developed in this paper will allow future work in this direction to be conducted more easily and more cleanly. Moreover, due to the compositionality of the approach, which allows different transformations and different properties to be studied fully independently from each other, we believe that our theory will make it significantly easier to integrate existing and future algorithms that operate on factored transition systems.

Acknowledgments

We thank the anonymous reviewers for their thorough reviews which helped improve the final version of this paper. We further thank Martin Wehrle, our co-author of the original conference paper which laid the foundation for this work. We have received funding for this work from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 817639). Moreover, this research was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215.

Appendix A. Proofs of Theorems in Section 3

Theorem 1. *Let X be any of the properties of transformations from Definition 8. Let τ be a transformation of transition system Θ into transition system Θ' with property X, and let τ' be a transformation of Θ' into transition system Θ'' with property X. Then the composed transformation $\tau'' = \tau' \circ \tau$ also has the property X.*

Proof. Let $\Theta = \langle S, L, c, T, S_\mathrm{I}, S_\mathrm{G}\rangle$, $\Theta' = \langle S', L', c', T', S'_\mathrm{I}, S'_\mathrm{G}\rangle$, and $\Theta'' = \langle S'', L'', c'', T'', S''_\mathrm{I}, S''_\mathrm{G}\rangle$. Let $\tau = \langle \Theta', \sigma, \lambda\rangle$ and $\tau' = \langle \Theta'', \sigma', \lambda'\rangle$. Then $\tau'' = \langle \Theta'', \sigma'', \lambda''\rangle$ with $\sigma'' = \sigma' \circ \sigma$ and $\lambda'' = \lambda' \circ \lambda$.

We remind the reader that $dom(\sigma'') = dom(\sigma) \cap \sigma^{-1}(dom(\sigma'))$ (cf. Definition 2). This means that $s \in dom(\sigma'')$ implies $s \in dom(\sigma)$ and $\sigma(s) \in dom(\sigma')$. Analogously, we have $dom(\lambda'') = dom(\lambda) \cap \lambda^{-1}(dom(\lambda'))$ and therefore $\ell \in dom(\lambda'')$ implies $\ell \in dom(\lambda)$ and $\lambda(\ell) \in dom(\lambda')$.

CONS$_\mathbf{S}$ The composition of total functions is total.

CONS$_\mathbf{L}$ The composition of total functions is total.

CONS$_\mathbf{C}$ Consider $\ell \in dom(\lambda'')$. This implies $\ell \in dom(\lambda)$, and because τ has property **CONS$_\mathbf{C}$**, $c'(\lambda(\ell)) \le c(\ell)$. It also implies $\lambda(\ell) \in dom(\lambda')$, and because τ' has property **CONS$_\mathbf{C}$**, $c''(\lambda'(\lambda(\ell)) \le c'(\lambda(\ell))$. Hence $c''(\lambda''(\ell)) = c''(\lambda'(\lambda(\ell)) \le c'(\lambda(\ell)) \le c(\ell)$, which shows that τ'' has property **CONS$_\mathbf{C}$**.

CONS$_\mathbf{T}$ Consider $s, t \in dom(\sigma'')$ and $\ell \in dom(\lambda'')$ with $s \xrightarrow{\ell} t \in T$. We get $s, t \in dom(\sigma)$, $\sigma(s), \sigma(t) \in dom(\sigma')$, $\ell \in dom(\lambda)$, and $\lambda(\ell) \in dom(\lambda')$. Because τ has property **CONS$_\mathbf{T}$**, we get $\sigma(s) \xrightarrow{\lambda(\ell)} \sigma(t) \in T'$. Because τ' has property **CONS$_\mathbf{T}$**, we get $\sigma'(\sigma(s)) \xrightarrow{\lambda'(\lambda(\ell))} \sigma'(\sigma(t)) \in T''$, which is $\sigma''(s) \xrightarrow{\lambda''(\ell)} \sigma''(t) \in T''$, showing that τ'' has property **CONS$_\mathbf{T}$**.

CONS$_\mathbf{I}$ Consider $s \in S_\mathrm{I}$ with $s \in dom(\sigma'')$. We get $s \in dom(\sigma)$ and $\sigma(s) \in dom(\sigma')$. Because τ has property **CONS$_\mathbf{I}$**, we get $\sigma(s) \in S'_\mathrm{I}$. Because τ' has property **CONS$_\mathbf{I}$**, we get $\sigma''(s) = \sigma'(\sigma(s)) \in S''_\mathrm{I}$, which shows that τ'' has property **CONS$_\mathbf{I}$**.

CONS$_\mathbf{G}$ Consider $s \in S_\mathrm{G}$ with $s \in dom(\sigma'')$. We get $s \in dom(\sigma)$ and $\sigma(s) \in dom(\sigma')$. Because τ has property **CONS$_\mathbf{G}$**, we get $\sigma(s) \in S'_\mathrm{G}$. Because τ' has property **CONS$_\mathbf{G}$**, we get $\sigma''(s) = \sigma'(\sigma(s)) \in S''_\mathrm{G}$, which shows that τ'' has property **CONS$_\mathbf{G}$**.

IND$_\mathbf{S}$ The composition of surjective functions is surjective.

IND$_\mathbf{L}$ The composition of surjective functions is surjective.

IND$_\mathbf{C}$ Consider $\ell'' \in L''$. Because τ' has property **IND$_\mathbf{C}$**, there exists $\ell' \in \lambda'^{-1}(\ell'')$ with $c'(\ell') = c''(\ell'')$. Because τ has property **IND$_\mathbf{C}$**, there exists $\ell \in \lambda^{-1}(\ell')$ with $c(\ell) = c'(\ell')$. Put together, there exists $\ell \in \lambda^{-1}(\lambda'^{-1}(\ell'')) = \lambda''^{-1}(\ell'')$ with $c(\ell) = c''(\ell'')$, which shows that τ'' has property **IND$_\mathbf{C}$**.

IND$_\mathbf{T}$ Consider $s'' \xrightarrow{\ell''} t'' \in T''$. Then, because τ' has property **IND$_\mathbf{T}$**, there exists $s' \xrightarrow{\ell'} t' \in T'$ with $s' \in \sigma'^{-1}(s'')$, $t' \in \sigma'^{-1}(t'')$ and $\ell' \in \lambda'^{-1}(\ell'')$. Because $s' \xrightarrow{\ell'} t' \in T'$ and τ has property **IND$_\mathbf{T}$**, there exists $s \xrightarrow{\ell} t \in T$ with $s \in \sigma^{-1}(s')$, $t \in \sigma^{-1}(t')$ and $\ell \in \lambda^{-1}(\ell')$. Putting everything together, we obtain that given $s'' \xrightarrow{\ell''} t'' \in T''$, there exists $s \xrightarrow{\ell} t \in T$ with $s \in \sigma^{-1}(\sigma'^{-1}(s'')) = \sigma''^{-1}(s'')$, $t \in \sigma^{-1}(\sigma'^{-1}(t'')) = \sigma''^{-1}(t'')$ and $\ell \in \lambda^{-1}(\lambda'^{-1}(\ell'')) = \lambda''^{-1}(\ell'')$, which shows that τ'' has property **IND$_\mathbf{T}$**.

IND$_\mathbf{I}$ Consider $s'' \in S''_\mathrm{I}$. Because τ' has property **IND$_\mathbf{I}$**, there exists $s' \in S'_\mathrm{I}$ with $s' \in \sigma'^{-1}(s'')$. Because $s' \in S'_\mathrm{I}$ and τ has property **IND$_\mathbf{I}$**, there exists $s \in S_\mathrm{I}$ with $s \in \sigma^{-1}(s')$. Put together, we obtain that there exists $s \in S_\mathrm{I}$ with $s \in \sigma^{-1}(\sigma'^{-1}(s'')) = \sigma''^{-1}(s'')$, which shows that τ'' has property **IND$_\mathbf{I}$**.

IND$_\mathbf{G}$ Consider $s'' \in S''_\mathrm{G}$. Because τ' has property **IND$_\mathbf{G}$**, there exists $s' \in S'_\mathrm{G}$ with $s' \in \sigma'^{-1}(s'')$. Because $s' \in S'_\mathrm{G}$ and τ has property **IND$_\mathbf{G}$**, there exists $s \in S_\mathrm{G}$ with $s \in \sigma^{-1}(s')$. Put together, we obtain that there exists $s \in S_\mathrm{G}$ with $s \in \sigma^{-1}(\sigma'^{-1}(s'')) = \sigma''^{-1}(s'')$, which shows that τ'' has property **IND$_\mathbf{G}$**.

REF$_\mathbf{C}$ Consider $\ell'' \in L''$. If $\lambda''^{-1}(\ell'') = \emptyset$, the property is vacuously true. (We universally quantify over an empty set.) Otherwise, consider $\ell' \in \lambda'^{-1}(\ell'')$ and $\ell \in \lambda^{-1}(\ell')$. Then, because τ' has property **REF$_\mathbf{C}$**, $c'(\ell') = c''(\ell'')$, and because τ has property **REF$_\mathbf{C}$**, $c(\ell) = c'(\ell')$. Put together, we obtain that given $\ell'' \in L''$, for all $\ell \in \lambda^{-1}(\lambda'^{-1}(\ell'')) = \lambda''^{-1}(\ell'')$, $c(\ell) = c''(\ell'')$, which shows that τ'' has property **REF$_\mathbf{C}$**.

REF$_\mathbf{T}$ Consider $s'' \xrightarrow{\ell''} t'' \in T''$. If $\sigma''^{-1}(s'') = \emptyset$, the property is vacuously true. (We universally quantify over an empty set.) Otherwise, consider $s' \in \sigma'^{-1}(s'')$ and $s \in \sigma^{-1}(s')$. Then, because τ' has property **REF$_\mathbf{T}$**, there exists $s' \xrightarrow{\ell'} t' \in T'$ with $t' \in \sigma'^{-1}(t'')$ and $\ell' \in \lambda'^{-1}(\ell'')$. Because $s' \xrightarrow{\ell'} t' \in T'$ and τ has property **REF$_\mathbf{T}$**, there exists $s \xrightarrow{\ell} t \in T$ with $t \in \sigma^{-1}(t')$ and $\ell \in \lambda^{-1}(\ell')$. Putting everything together, we obtain that given $s'' \xrightarrow{\ell''} t'' \in T''$, for all $s \in \sigma^{-1}(\sigma'^{-1}(s'')) = \sigma''^{-1}(s'')$, there exists $s \xrightarrow{\ell} t \in T$ with $t \in \sigma^{-1}(\sigma'^{-1}(t'')) = \sigma''^{-1}(t'')$ and $\ell \in \lambda^{-1}(\lambda'^{-1}(\ell'')) = \lambda''^{-1}(\ell'')$, which shows that τ'' has property **REF$_\mathbf{T}$**.

REF$_\mathbf{G}$ Consider $s'' \in S''_\mathrm{G}$. If $\sigma''^{-1}(s'') = \emptyset$, the property is vacuously true. (We universally quantify over an empty set.) Otherwise, consider $s' \in \sigma'^{-1}(s'')$ and $s \in \sigma^{-1}(s')$. Then, because τ' has property **REF$_\mathbf{G}$**, we have $s' \in S'_\mathrm{G}$, and because τ has property **REF$_\mathbf{G}$**, we have $s \in S_\mathrm{G}$. Put together, we obtain that given $s'' \in S''_\mathrm{G}$, for all $s \in \sigma^{-1}(\sigma'^{-1}(s'')) = \sigma''^{-1}(s'')$, we have $s \in S_\mathrm{G}$, which shows that τ'' has property **REF$_\mathbf{G}$**. □

Theorem 2. *Let Θ be a transition system with states S and label costs c, and let $\tau = \langle \Theta', \sigma, \lambda \rangle$ be a transformation of Θ into transition system Θ' with label costs c'. Let π be a path in Θ. Then:*

1. *If τ is state- and label-conservative (**CONS$_\mathbf{S+L}$**), then $\tau(\pi)$ is defined.*
2. *If τ is transition-conservative (**CONS$_\mathbf{T}$**) and $\tau(\pi)$ is defined, then $\tau(\pi)$ is a legal path in Θ'.*
3. *If τ is transition- and goal-state-conservative (**CONS$_\mathbf{T+G}$**), π is an s-plan and $\tau(\pi)$ is defined, then $\tau(\pi)$ is a $\sigma(s)$-plan for Θ'.*
4. *If τ is state-, label-, transition-, initial-state- and goal-state-conservative (**CONS$_\mathbf{S+L+T+I+G}$**) and π is a plan for Θ, then $\tau(\pi)$ is a plan for Θ'.*
5. *If τ is cost-conservative (**CONS$_\mathbf{C}$**) and $\tau(\pi)$ is defined, then $c'(\tau(\pi)) \leq c(\pi)$.*

Proof. 1. Obvious.

2. Let $\pi = \langle s_0 \xrightarrow{\ell_1} s_1, \ldots, s_{n-1} \xrightarrow{\ell_n} s_n \rangle$ be a path in Θ. Because $\tau(\pi)$ is defined, we have $s_i \in \mathit{dom}(\sigma)$ for all $0 \leq i \leq n$ and $\ell_i \in \mathit{dom}(\lambda)$ for all $1 \leq i \leq n$, and we get $\tau(\pi) = \langle \sigma(s_0) \xrightarrow{\lambda(\ell_1)} \sigma(s_1), \ldots, \sigma(s_{n-1}) \xrightarrow{\lambda(\ell_n)} \sigma(s_n) \rangle$. Because τ is transition-conservative, all elements of $\tau(\pi)$ are transitions of Θ', and hence $\tau(\pi)$ is a legal path in Θ'.

3. Let π be an s-plan for Θ such that $\tau(\pi)$ is defined. Let t be the goal state in which π ends. From the previous part, we obtain that $\tau(\pi)$ is a legal path in Θ' beginning in state $\sigma(s)$ and ending in state $\sigma(t)$. Because $\tau(\pi)$ is defined and π ends in state t, we get $t \in \mathit{dom}(\sigma)$. Because τ is goal-state-conservative and $t \in \mathit{dom}(\sigma)$, $\sigma(t)$ is a goal state in Θ', and hence $\tau(\pi)$ is a $\sigma(s)$-plan for Θ'.

4. Because π is a plan for Θ, it is an s-plan for some initial state $s \in S_\mathrm{I}$. Because of 1., $\tau(\pi)$ is defined, and because of 3., it is a $\sigma(s)$-plan. Because τ is initial-state-conservative and $s \in \mathit{dom}(\sigma)$, we get $\sigma(s) \in S'_\mathrm{I}$, showing that $\tau(\pi)$ is a plan for Θ'.

5. Let $\pi = \langle s_0 \xrightarrow{\ell_1} s_1, \dots, s_{n-1} \xrightarrow{\ell_n} s_n\rangle$ such that $\tau(\pi)$ is defined. Then $\tau(\pi) = \langle \sigma(s_0) \xrightarrow{\lambda(\ell_1)} \sigma(s_1), \dots, \sigma(s_{n-1}) \xrightarrow{\lambda(\ell_n)} \sigma(s_n)\rangle$, and we get $c'(\tau(\pi)) = \sum_{i=1}^{n} c'(\lambda(\ell_i)) \leq \sum_{i=1}^{n} c(\ell_i) = c(\pi)$, where the inequality holds because τ is cost-conservative. (In this step, we use that $\ell_i \in \mathit{dom}(\lambda)$ for all $1 \leq i \leq n$ because $\tau(\pi)$ is defined.) □

Theorem 3. *Let τ be a transformation of a transition system Θ. The heuristic h^τ for Θ induced by τ is*

1. *goal-aware if τ is state- and goal-state-conservative (**CONS$_\mathbf{S+G}$**),*

2. *consistent if τ is state-, label-, cost- and transition-conservative (**CONS$_\mathbf{S+L+C+T}$**), and*

3. *admissible if τ is state-, label-, cost-, transition- and goal-state-conservative (**CONS$_\mathbf{S+L+C+T+G}$**).*

Proof. Let $\Theta = \langle S, L, c, T, S_\mathrm{I}, S_\mathrm{G}\rangle$ and $\Theta' = \langle S', L', c', T', S'_\mathrm{I}, S'_\mathrm{G}\rangle$.

1. Let s be a goal state of Θ. Then $s \in \mathit{dom}(\sigma)$ due to **CONS$_\mathbf{S}$** and $\sigma(s)$ is a goal state of Θ' due to **CONS$_\mathbf{G}$**. Therefore, $h^\tau(s) = h^*_{\Theta'}(\sigma(s)) = 0$.

2. Let $s \xrightarrow{\ell} t$ be a transition of Θ. We have $s \in \mathit{dom}(\sigma)$ and $t \in \mathit{dom}(\sigma)$ due to **CONS$_\mathbf{S}$** and $\ell \in \mathit{dom}(\lambda)$ due to **CONS$_\mathbf{L}$**. We get $h^\tau(s) = h^*_{\Theta'}(\sigma(s)) \leq c'(\lambda(\ell)) + h^*_{\Theta'}(\sigma(t)) \leq c(\ell) + h^*_{\Theta'}(\sigma(t)) = c(\ell) + h^\tau(t)$, where the first inequality holds because $\sigma(s) \xrightarrow{\lambda(\ell)} \sigma(t)$ is a transition of Θ' (due to **CONS$_\mathbf{T}$**) and the perfect heuristic $h^*_{\Theta'}$ is consistent, and the second inequality holds because $c'(\lambda(\ell)) \leq c(\ell)$ (due to **CONS$_\mathbf{C}$**).

3. Follows from 1. and 2. because goal-awareness and consistency imply admissibility. □

Theorem 4. *Let Θ be a transition system with label costs c, and let $\tau = \langle \Theta', \sigma, \lambda\rangle$ be a transformation of Θ into transition system Θ' with label costs c'. Let π' be a path from state s' to state t' in Θ', and let $s \in \sigma^{-1}(s')$. Then:*

1. *If τ is transition-refinable (**REF$_\mathbf{T}$**), then there exists a legal path $\pi \in \tau^{-1}(\pi')$ from s to some state $t \in \sigma^{-1}(t')$ in Θ.*

2. *If τ is transition-refinable and goal-state-refinable (**REF$_\mathbf{T+G}$**) and π' is an s'-plan for Θ', then there exists an s-plan $\pi \in \tau^{-1}(\pi')$ for Θ.*

3. *If τ is cost-refinable (**REF$_\mathbf{C}$**), then $c(\pi) = c'(\pi')$ for all $\pi \in \tau^{-1}(\pi')$.*

Proof. 1. The proof is by induction over the length of π', denoted by $|\pi'|$. Base case: $|\pi'| = 0$, which implies $\pi' = \langle\rangle$ and $s' = t'$. Set $t = s$. Then $\pi = \langle\rangle \in \tau^{-1}(\pi')$ is a legal path from $s \in \sigma^{-1}(s')$ to $t = s \in \sigma^{-1}(s') = \sigma^{-1}(t')$. Inductive step: assume that the property holds for some s'-plan $\pi'_{u'}$ that ends in state u' and which has length $|\pi'_{u'}| = n$. Consider the plan $|\pi'| = \pi'_{u'} \circ \langle u' \xrightarrow{\ell'} t'\rangle$, i.e., π' consists of a length-n path from s' to some state u', followed by a transition from u' to t' with label ℓ'.[13] Clearly, π' has length $|\pi'| = n + 1$. By the induction hypothesis, there exists a legal path $\pi_u \in \tau^{-1}(\pi'_{u'})$ in Θ from s to some state $u \in \sigma^{-1}(u')$. Because of $\mathbf{REF_T}$, Θ has a transition $u \xrightarrow{\ell} t$ with $\ell \in \lambda^{-1}(\ell')$ and $t \in \sigma^{-1}(t')$. Let $\pi = \pi_u \circ \langle u \xrightarrow{\ell} t\rangle$. We observe that $\pi \in \tau^{-1}(\pi')$ and π is a path from s to $t \in \sigma^{-1}(t')$ in Θ, concluding the proof.

2. Because of part 1., there exists a legal path $\pi \in \tau^{-1}(\pi')$ from s to some state $t \in \sigma^{-1}(t')$ in Θ. Because π' is a plan, t' is a goal state of Θ'. With $\mathbf{REF_G}$ and $t \in \sigma^{-1}(t')$, we obtain that t is a goal state of Θ, and hence π is an s-plan.

3. Let $\langle \ell'_1, \dots, \ell'_n\rangle$ be the sequence of labels in π'. From the definition of refinements, the sequence of labels in π is of the form $\langle \ell_1, \dots, \ell_n\rangle$ with $\ell_i \in \lambda^{-1}(\ell'_i)$ for all $1 \le i \le n$. From $\mathbf{REF_C}$, we get $c(\ell_i) = c'(\ell'_i)$ for all $1 \le i \le n$. We obtain $c(\pi) = \sum_{i=1}^{n} c(\ell_i) = \sum_{i=1}^{n} c'(\ell'_i) = c'(\pi')$, concluding the proof. □

Theorem 5. *Let τ be a transformation of a transition system Θ. The heuristic h^τ for Θ induced by τ is*

1. *lower-bounded by h^* ($h^*(s) \le h^\tau(s)$ for all states s) if τ is refinable (**REF**) and*

2. *perfect if τ is exact (**CONS** + **REF**), or more generally if τ is $\mathbf{CONS_{S+L+C+T+G}}$ + **REF**.*

Proof. Let $\tau = \langle \Theta', \sigma, \lambda\rangle$. Part 1. follows directly from Theorem 4: for states s with $h^\tau(s) = \infty$, there is nothing to show. If $h^\tau(s) = k < \infty$, then Θ' has a $\sigma(s)$-plan π' of cost k. Because $s \in \sigma^{-1}(\sigma(s))$, Theorem 4.2 shows that Θ has an s-plan $\pi \in \tau^{-1}(\pi')$, and Theorem 4.3 shows that the cost of this plan is k. Therefore $h^*(s)$, the cost of an *optimal* s-plan, is at most k.

Part 2. follows because $h^*(s) \le h^\tau(s)$ (part 1.) and $h^\tau(s) \le h^*(s)$ (admissibility; Theorem 3.3) imply $h^\tau(s) = h^*(s)$. □

Appendix B. Proofs of Theorems in Section 4

Theorem 6. *Let X be any of the properties of transformations from Definition 23. Let τ_{F} be a factored transformation of F into F' with property X, and let τ'_{F} be a factored transformation of F' into F'' with property X. Then $\tau'_{\mathrm{F}} \circ \tau_{\mathrm{F}}$ has the property X.*

Proof. We first show the claim for property $\mathbf{LOC}_{\le}$. We must show $h_F^{\mathrm{loc},\tau''_{\mathrm{F}}}(s) \le h_F^{\mathrm{mf}}(s)$ for all $s \in \mathcal{A}(F)$. Using the definition of $\mathbf{LOC}_{\le}$ for τ_{F} and τ'_{F}, we have

$$h_F^{\mathrm{loc},\tau_{\mathrm{F}}}(s) \le h_F^{\mathrm{mf}}(s) \text{ for all } s \in \mathcal{A}(F) \tag{6}$$

$$h_{F'}^{\mathrm{loc},\tau'_{\mathrm{F}}}(s') \le h_{F'}^{\mathrm{mf}}(s') \text{ for all } s' \in \mathcal{A}(F') \tag{7}$$

13. We use the symbol "$\circ$" to denote concatenation of sequences.

We then get:

$$\begin{array}{ccccccc} h_F^{\mathrm{loc},\tau''_{\mathrm{F}}}(s) & \overset{\text{Def. 22}}{=} & h_{F''}^{\mathrm{mf}}([\![\Sigma'']\!](s)) & \overset{\Sigma''=\Sigma'\circ\Sigma}{=} & h_{F''}^{\mathrm{mf}}([\![\Sigma']\!]([\![\Sigma]\!](s))) & \overset{\text{Def. 22}}{=} & h_{F'}^{\mathrm{loc},\tau'_{\mathrm{F}}}([\![\Sigma]\!](s)) \\ & \overset{(7)}{\leq} & h_{F'}^{\mathrm{mf}}([\![\Sigma]\!](s)) & \overset{\text{Def. 22}}{=} & h_F^{\mathrm{loc},\tau_{\mathrm{F}}}(s) & \overset{(6)}{\leq} & h_F^{\mathrm{mf}}(s). \end{array}$$

For property $\mathbf{LOC}_{\geq}$, we can use the same proof with "$\leq$" replaced by "$\geq$" everywhere. The result for $\mathbf{LOC}_{=}$ follows because $\mathbf{LOC}_{=} = \mathbf{LOC}_{\leq} + \mathbf{LOC}_{\geq}$. □

Appendix C. Proofs of Theorems in Section 5

Theorem 7. *All shrink transformations are induced abstractions, i.e., they satisfy* **CONS** + **IND**. *They are also cost-refinable, i.e., satisfy* $\mathbf{REF_C}$*, and locally nonincreasing, i.e., satisfy* $\mathbf{LOC}_{\leq}$.

Proof. Let $F = \langle \Theta_1, \dots, \Theta_n \rangle$ and $F' = \langle \Theta'_1, \dots, \Theta'_n \rangle$ be factored transition systems. Let $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$ be a shrink transformation of F into F' based on a local abstraction α of Θ_k for some $1 \leq k \leq n$.

By the definition of shrink transformations, we have $\Theta'_i = \Theta_i$ for all $i \neq k$, $\Theta'_k = \Theta_k^\alpha$, $\Sigma = \langle \sigma_1, \dots, \sigma_n \rangle$ where $\sigma_i = \pi_i$ for all $i \neq k$, σ_k is an atomic FM with variable Θ_k and $\sigma_k^{\mathrm{tab}}(s_k) = \alpha(s_k)$ for all $s_k \in \Theta_k$, and $\lambda = \mathrm{id}$. Let $\tau = \langle \bigotimes F', [\![\Sigma]\!], \lambda \rangle$ be the transformation of $\bigotimes F$ into $\bigotimes F'$ induced by τ_{F}.

Since $\lambda = \mathrm{id}$, we simplify the notation by dropping the use of λ and λ^{-1} throughout. Furthermore, as a general remark, we observe that σ_k represents the function α applied to the local state of Θ_k in a given state of F, i.e., $[\![\sigma_k]\!](s) = \alpha(s[\Theta_k])$. Thus we can reason about σ_k by reasoning about α.

We must show that τ satisfies $\mathbf{CONS_S}$, $\mathbf{CONS_L}$, $\mathbf{CONS_C}$, $\mathbf{CONS_T}$, $\mathbf{CONS_I}$, $\mathbf{CONS_G}$, $\mathbf{IND_S}$, $\mathbf{IND_L}$, $\mathbf{IND_C}$, $\mathbf{IND_T}$, $\mathbf{IND_I}$, $\mathbf{IND_G}$, $\mathbf{REF_C}$, and that τ_{F} satisfies $\mathbf{LOC}_{\leq}$.

- $\mathbf{CONS_S}$: $[\![\Sigma]\!]$ is a total function.

- $\mathbf{CONS_L}$, $\mathbf{CONS_C}$, $\mathbf{IND_L}$, $\mathbf{IND_C}$, $\mathbf{REF_C}$: It is easy to see that τ_F satisfies these properties because $\lambda = \mathrm{id}$ and hence labels (and their costs) are not changed.

- $\mathbf{CONS_T}$: Consider a transition $s \xrightarrow{\ell} t \in \bigotimes F$. By the definition of products, $s[\Theta_i] \xrightarrow{\ell} t[\Theta_i] \in \Theta_i$ for all $1 \leq i \leq n$. Furthermore, for all $i \neq k$, we have $\Theta_i = \Theta'_i$ and thus $[\![\sigma_i]\!](s) \xrightarrow{\ell} [\![\sigma_i]\!](t) \in \Theta'_i$ because $[\![\sigma_i]\!]$ is the projection onto Θ_i (and thus the identity function on Θ_i). For $i = k$, we also have that $[\![\sigma_k]\!](s) \xrightarrow{\ell} [\![\sigma_k]\!](t) \in \Theta_k^\alpha$ because Θ_k^α is the transition system induced by Θ_k and α. Put together, we have that $[\![\sigma_i]\!](s) \xrightarrow{\ell} [\![\sigma_i]\!](t) \in \Theta'_i$ for all $1 \leq i \leq n$, and hence by the definition of products, $[\![\Sigma]\!](s) \xrightarrow{\ell} [\![\Sigma]\!](t) \in \bigotimes F'$, which shows that τ satisfies $\mathbf{CONS_T}$.

- $\mathbf{CONS_I}$: Consider an initial state $s \in \bigotimes F$. By the definition of products, $s[\Theta_i]$ is an initial state of Θ_i for all $1 \leq i \leq n$. Furthermore, for all $i \neq k$, we have $\Theta_i = \Theta'_i$ and thus $[\![\sigma_i]\!](s)$ is an initial state of Θ'_i because $[\![\sigma_i]\!]$ is the projection onto Θ_i (and thus the identity function on Θ_i). For $i = k$, we also have that $[\![\sigma_k]\!](s)$ is an initial state of Θ_k^α because Θ_k^α is the transition system induced by Θ_k and α. Put together, we have that $[\![\sigma_i]\!](s)$ is an initial state of Θ'_i for all $1 \leq i \leq n$, and hence by the definition of products, $[\![\Sigma]\!](s)$ is an initial state of $\bigotimes F'$, which shows that τ satisfies $\mathbf{CONS_I}$.

- $\mathbf{CONS_G}$: same as the proof for $\mathbf{CONS_I}$, replacing "initial" with "goal" everywhere.

- $\mathbf{IND_S}$: Consider a state $s' \in \bigotimes F'$. By the definition of products, $s'[\Theta'_i]$ is a state of Θ'_i for all $1 \le i \le n$. Furthermore, for all $i \neq k$, we have $\Theta_i = \Theta'_i$, and hence $s'[\Theta'_i]$ is a state of Θ_i. For $i = k$, there exists a state s_k of Θ_k with $\alpha(s_k) = s'[\Theta^\alpha_k]$ because α is surjective. Put together, by setting $s = \{\Theta_i \mapsto s'[\Theta'_i] \mid 1 \le i \le n, i \neq k\} \cup \{\Theta_k \mapsto s_k\}$ for some $s_k \in \alpha^{-1}(s'[\Theta^\alpha_k])$, we have that s is a state of $\bigotimes F$ with $[\![\Sigma]\!](s) = s'$, which shows that $[\![\Sigma]\!]$ is surjective and thus τ satisfies $\mathbf{IND_S}$.

- $\mathbf{IND_T}$: Consider a transition $s' \xrightarrow{\ell} t' \in \bigotimes F'$. By the definition of products, $s'[\Theta'_i] \xrightarrow{\ell} t'[\Theta'_i] \in \Theta'_i$ for all $1 \le i \le n$. Furthermore, for all $i \neq k$, we have $\Theta_i = \Theta'_i$ and thus $s'[\Theta'_i] \xrightarrow{\ell} t'[\Theta'_i] \in \Theta_i$. For $i = k$, there exist $s_k \in \alpha^{-1}(s'_k)$ and $t_k \in \alpha^{-1}(t'_k)$ with $s_k \xrightarrow{\ell} t_k \in \Theta_k$ because Θ^α_k is induced by Θ_k and α. Put together, by setting $s = \{\Theta_i \mapsto s'[\Theta'_i] \mid 1 \le i \le n, i \neq k\} \cup \{\Theta_k \mapsto s_k\}$ and $t = \{\Theta_i \mapsto t'[\Theta'_i] \mid 1 \le i \le n, i \neq k\} \cup \{\Theta_k \mapsto t_k\}$ for some $s_k \in \alpha^{-1}(s'[\Theta^\alpha_k])$ and $t_k \in \alpha^{-1}(t'[\Theta^\alpha_k])$ with $s_k \xrightarrow{\ell} t_k \in \Theta_k$, we have that $s \xrightarrow{\ell} t \in \bigotimes F$ with $[\![\Sigma]\!](s) = s'$ and $[\![\Sigma]\!](t) = t'$, which shows that τ satisfies $\mathbf{IND_T}$.

- $\mathbf{IND_I}$: Consider an initial state $s' \in \bigotimes F'$. By the definition of products, $s'[\Theta'_i]$ is an initial state of Θ'_i for all $1 \le i \le n$. Furthermore, for all $i \neq k$, we have $\Theta_i = \Theta'_i$, and hence $s'[\Theta'_i]$ is an initial state of Θ_i. For $i = k$, there exists an initial state s_k of Θ_k with $\alpha(s_k) = s'[\Theta^\alpha_k]$ because Θ^α_k is induced by Θ_k and α. Put together, by setting $s = \{\Theta_i \mapsto s'[\Theta'_i] \mid 1 \le i \le n, i \neq k\} \cup \{\Theta_k \mapsto s_k\}$ for some initial state $s_k \in \alpha^{-1}(s'[\Theta^\alpha_k])$, we have that s is an initial state of $\bigotimes F$ with $[\![\Sigma]\!](s) = s'$, which shows that τ satisfies $\mathbf{IND_I}$.

- $\mathbf{IND_G}$: same as the proof for $\mathbf{IND_I}$, replacing "initial" with "goal" everywhere.

- $\mathbf{LOC_\le}$: We need to show $h^{\mathrm{loc},\tau_F}_F(s) \le h^{\mathrm{mf}}_F(s)$ for all states $s \in \mathcal{A}(F)$, which translates to showing $h^{\mathrm{mf}}_{F'}([\![\Sigma]\!](s)) \le h^{\mathrm{mf}}_F(s)$ by the definition of local heuristics. By the definition of max-factor heuristics, we have $h^{\mathrm{mf}}_{F'} = \max_{1\le i\le n} h'_i$ where $h'_i(s') = h^*_{\Theta'_i}(s'[\Theta'_i])$ and $h^{\mathrm{mf}}_F = \max_{1\le i\le n} h_i$ where $h_i(s) = h^*_{\Theta_i}(s[\Theta_i])$. Thus, we need to show $\max_{1\le i\le n} h'_i([\![\Sigma]\!](s)) \le \max_{1\le i\le n} h_i(s)$. It suffices to show $h'_i([\![\Sigma]\!](s)) \le h_i(s)$ for all $1 \le i \le n$.

  For $i \neq k$, we have $h'_i([\![\Sigma]\!](s)) = h_i(s)$ because $\Theta'_i = \Theta_i$ and $[\![\Sigma]\!](s)[\Theta'_i] = s[\Theta_i]$. For $i = k$, we have $h'_i([\![\Sigma]\!](s)) = h^*_{\Theta^\alpha_k}(\alpha(s[\Theta_k])) \le h^*_{\Theta_k}(s[\Theta_k]) = h_i(s)$, where the inequality holds because Θ^α_k is the transition system induced by Θ_k and α and therefore the shortest path costs in Θ^α_k cannot exceed the corresponding shortest path costs in Θ_k (cf. properties $\mathbf{CONS_T}$ and $\mathbf{CONS_C}$ of induced abstractions). Together, this shows that that τ_F satisfies $\mathbf{LOC_\le}$. □

Theorem 8. *Consider a shrink transformation τ_F based on a local abstraction α of factor Θ such that $\alpha(s_1) = \alpha(s_2)$ implies $h^*_\Theta(s_1) = h^*_\Theta(s_2)$ for all states $s_1, s_2 \in \Theta$. Then τ_F is locally nondecreasing (*$\mathbf{LOC_\ge}$*) and hence locally exact (*$\mathbf{LOC_=}$*).*

Proof. We prove $\mathbf{LOC_=}$, which implies $\mathbf{LOC_\ge}$. The proof follows the same structure as the proof for $\mathbf{LOC_\le}$ in Theorem 7. Let $F = \langle \Theta_1, \dots, \Theta_n \rangle$ be a factored transition system, and let $\tau_F = \langle F', \Sigma, \lambda \rangle$ be the shrink transformation from F into $F' = \langle \Theta'_1, \dots, \Theta'_n \rangle$, where $\Theta'_k = \Theta^\alpha_k$ is induced by Θ_k and α and $\Theta'_i = \Theta_i$ for all $i \neq k$.

We need to show $h^{\mathrm{loc},\tau_F}_F(s) = h^{\mathrm{mf}}_F(s)$ for all states $s \in \mathcal{A}(F)$, which translates to showing $h^{\mathrm{mf}}_{F'}([\![\Sigma]\!](s)) = h^{\mathrm{mf}}_F(s)$ by the definition of local heuristics. By the definition of max-factor heuristics, we have $h^{\mathrm{mf}}_{F'} = \max_{1\le i\le n} h'_i$ where $h'_i(s') = h^*_{\Theta'_i}(s'[\Theta'_i])$ and $h^{\mathrm{mf}}_F = \max_{1\le i\le n} h_i$ where

$h_i(s) = h^*_{\Theta_i}(s[\Theta_i])$. Thus, we need to show $\max_{1 \le i \le n} h'_i([\![\Sigma]\!](s)) = \max_{1 \le i \le n} h_i(s)$. It suffices to show $h'_i([\![\Sigma]\!](s)) = h_i(s)$ for all $1 \le i \le n$.

For $i \neq k$, we have $h'_i([\![\Sigma]\!](s)) = h_i(s)$ because $\Theta'_i = \Theta_i$ and $[\![\Sigma]\!](s)[\Theta'_i] = s[\Theta_i]$. For $i = k$, we have $h'_i([\![\Sigma]\!](s)) = h^*_{\Theta^\alpha_k}(\alpha(s[\Theta_k])) \overset{(*)}{=} h^*_{\Theta_k}(s[\Theta_k]) = h_i(s)$, where (*) holds because Θ^α_k is the transition system induced by Θ_k and α, where α only combines states with the same goal distance. Under such an abstraction, the goal distance of a state in Θ_k and of its image in Θ^α_k must be the same, which is easy to see by contradiction. Together, this shows that that τ_{F} satisfies **LOC**$_=$. □

Theorem 9. *Let τ_{F} be a shrink transformation based on the local abstraction α of factor Θ where α is induced by the equivalence relation $\sim$ on the states of Θ. Then:*

1. *If $\sim$ has the property* **BISIM1***, then τ_{F} has the property* **REF$_\mathbf{G}$**.
2. *If $\sim$ has the property* **BISIM2***, then τ_{F} has the property* **REF$_\mathbf{T}$**.
3. *If $\sim$ is a bisimulation, then τ_{F} is exact induced (***CONS** + **IND** + **REF***) and locally equal (***LOC**$_=$*).*

Proof. Let $F = \langle \Theta_1, \dots, \Theta_n \rangle$ and $F' = \langle \Theta'_1, \dots, \Theta'_n \rangle$ be factored transition systems. Let $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$ be a shrink transformation of F into F' based on a local abstraction α of factor Θ_k for some $1 \le k \le n$, where α is induced by the bisimulation relation $\sim$ on the states of Θ_k.

By the definition of shrink transformations, we have $\Theta'_i = \Theta_i$ for all $i \neq k$, $\Theta'_k = \Theta^\alpha_k$, $\Sigma = \langle \sigma_1, \dots, \sigma_n \rangle$ where $\sigma_i = \pi_i$ for all $i \neq k$, σ_k is an atomic FM with variable Θ_k and $\sigma^{\text{tab}}_k(s_k) = \alpha(s_k)$ for all $s_k \in \Theta_k$, and $\lambda = \text{id}$. Let $\tau = \langle \bigotimes F', [\![\Sigma]\!], \lambda \rangle$ be the transformation of $\bigotimes F$ into $\bigotimes F'$ induced by τ_{F}.

Since $\lambda = \text{id}$, we simplify the notation by dropping the use of λ and λ^{-1} throughout. Furthermore, as a general remark, we observe that σ_k represents the function α applied to the local state of Θ_k in a given state of F, i.e., $[\![\sigma_k]\!](s) = \alpha(s[\Theta_k])$. Thus we can reason about σ_k by reasoning about α.

1. **REF$_\mathbf{G}$** *given* **BISIM1***:* Consider a goal state $s' \in \bigotimes F'$. By the definition of products, $s'[\Theta'_i]$ is a goal state of Θ'_i for all $1 \le i \le n$. Furthermore, for all $i \neq k$, we have $\Theta_i = \Theta'_i$, and hence $s'[\Theta'_i]$ is a goal state of Θ_i. Because Θ^α_k is induced by Θ_k and α, at least one $s_k \in \alpha^{-1}(s'[\Theta^\alpha_k])$ is a goal state of Θ_k. Because $\sim$ (on which α is based) has property **BISIM1**, this implies that *all* such s_k are goal states of Θ_k.

   The preimages of s' under $[\![\Sigma]\!]$ are exactly the states of the form $s = \{\Theta_i \mapsto s'[\Theta'_i] \mid 1 \le i \le n, i \neq k\} \cup \{\Theta_k \mapsto s_k\}$, where $s_k \in \alpha^{-1}(s'[\Theta^\alpha_k])$. By the above reasoning, all component states $s[\Theta_i]$ of these states are goal states of Θ_i, and therefore all states s are goal states of $\bigotimes F$. Therefore τ satisfies **REF$_\mathbf{G}$**.

2. **REF$_\mathbf{T}$** *given* **BISIM2***:* Consider a transition $s' \xrightarrow{\ell} t' \in \bigotimes F'$ and a state $s \in [\![\Sigma]\!]^{-1}(s')$. We must show that there exists a state $t \in [\![\Sigma]\!]^{-1}(t')$ with $s \xrightarrow{\ell} t \in \bigotimes F$.

   By the definition of products, we have $s'[\Theta'_i] \xrightarrow{\ell} t'[\Theta'_i] \in \Theta'_i$ for all $1 \le i \le n$. Furthermore, for all $i \neq k$, we have $\Theta_i = \Theta'_i$ and thus $s'[\Theta'_i] \xrightarrow{\ell} t'[\Theta'_i] \in \Theta_i$. Because Θ^α_k is induced by Θ_k and α, there exist states $s_k \in \alpha^{-1}(s'[\Theta^\alpha_k])$ and $t_k \in \alpha^{-1}(t'[\Theta^\alpha_k])$ such that $s_k \xrightarrow{\ell} t_k \in \Theta_k$. Consider any state $\tilde{s}_k \in \alpha^{-1}(s'[\Theta^\alpha_k])$. Because α is based on $\sim$, we have $\tilde{s}_k \sim s_k$. Because

$\sim$ has property **BISIM2**, there exists a state $\tilde{t}_k \sim t_k$ with $\tilde{s}_k \xrightarrow{\ell} \tilde{t}_k \in \Theta_k$. Because α is based on $\sim$, we have $\tilde{t}_k \in \alpha^{-1}(t'[\Theta_k^\alpha])$. Therefore, the transition $s'_k \xrightarrow{\ell} t'_k$ is induced by some transition of Θ_k for *all* preimages of s'_k.

The preimages of $s' \xrightarrow{\ell} t'$ under $[\![\Sigma]\!]$ are exactly the transitions $s \xrightarrow{\ell} t$ where $s_i \xrightarrow{\ell} t_i \in \Theta_i$ for all $1 \leq i \leq n$. By the above reasoning, such component transitions exist for *all* preimages s_i of s'_i, and hence $s' \xrightarrow{\ell} t'$ has a preimage transition for all $s \in [\![\Sigma]\!]^{-1}(s')$. Therefore τ satisfies $\mathbf{REF_T}$.

3. *Exact induced and* $\mathbf{LOC_=}$ *given bisimulation:* Due to Theorem 7, τ_F satisfies **CONS**+**IND**+$\mathbf{REF_C}$. The previous two parts of the theorem show that τ_F also satisfies $\mathbf{REF_T}$ and $\mathbf{REF_G}$ and is therefore exact induced. For $\mathbf{LOC_=}$, we show that shrink transformations based on bisimulation are h-preserving and apply Theorem 8.

   We say that a *solution* for a state $s_k \in \Theta_k$ is the sequence of labels along some path from s_k to a goal state of Θ_k. We show that bisimilar states admit the same solutions: if $s_k \sim \tilde{s}_k$ and π is a solution for s_k, then π is a solution for $\tilde{s}_k$. From this it follows that $h^*_{\Theta_k}(s_k) = h^*_{\Theta_k}(\tilde{s}_k)$ whenever $s_k \sim \tilde{s}_k$, i.e., whenever α maps s_k and $\tilde{s}_k$ to the same state.

   The proof is by induction over the length of π. For the base case, let π be a solution for s_k with $|\pi| = 0$. Then s_k is a goal state. With $s_k \sim \tilde{s}_k$ and **BISIM1**, it follows that $\tilde{s}_k$ is a goal state and hence π is a solution for $\tilde{s}_k$.

   For the induction step, let π be a solution for s_k with $|\pi| = n+1$. Then there exists a transition $s_k \xrightarrow{\ell} t_k \in \Theta_k$ such that $\pi = \langle \ell \rangle + \pi'$, where π' is a solution for t_k with $|\pi'| = n$. Using $s_k \sim \tilde{s}_k$, $s_k \xrightarrow{\ell} t_k \in \Theta_k$ and **BISIM2**, there exists a state $\tilde{t}_k$ with $t_k \sim \tilde{t}_k$ and $\tilde{s}_k \xrightarrow{\ell} \tilde{t}_k \in \Theta_k$. Because $t_k \sim \tilde{t}_k$ and π' is a solution for t_k of length n, the induction hypothesis shows that π' is a solution for $\tilde{t}_k$. Therefore, π is a solution for $\tilde{s}_k$. □

Theorem 10. *Let τ_F be a shrink transformation of a factored transition system F based on the local abstraction α of factor Θ, where α is induced by the equivalence relation $\sim$ on the states of Θ. Then:*

1. *If τ_F has the property* $\mathbf{REF_G}$ *and F is not trivially unsolvable, then $\sim$ has the property* **BISIM1**.

2. *If τ_F has the property* $\mathbf{REF_T}$ *and F has no dead labels, then $\sim$ has the property* **BISIM2**.

3. *If τ_F is exact, F is not trivially unsolvable and F has no dead labels, then $\sim$ is a bisimulation.*

Proof. Let $F = \langle \Theta_1, \dots, \Theta_n \rangle$ and $F' = \langle \Theta'_1, \dots, \Theta'_n \rangle$ be factored transition systems. Let $\tau_F = \langle F', \Sigma, \lambda \rangle$ be a shrink transformation of F into F' based on a local abstraction α of factor Θ_k for some $1 \leq k \leq n$, where α is induced by the equivalence relation $\sim$ on the states of Θ_k. Since $\lambda = \text{id}$, we simplify the notation by dropping the use of λ and λ^{-1} throughout.

1. **BISIM1** *given* $\mathbf{REF_G}$*:* Consider two states $s_k, \tilde{s}_k \in \Theta_k$ with $s_k \sim \tilde{s}_k$. We show: if s_k is a goal state of Θ_k, then $\tilde{s}_k$ is a goal state of Θ_k. This implies that $\sim$ has the property **BISIM1**. So let s_k be a goal state of Θ_k.

   Because $\bigotimes F$ is not trivially unsolvable, it has a goal state. The goal states of $\bigotimes F$ are exactly the states where every component is a goal state of the corresponding factor, all factors of F

have at least one goal state. For all $i \neq k$, let s_i be a goal state of Θ_i. Then $s = \langle s_1, \dots, s_n \rangle$ is a goal state.

Let $s' = [\![\Sigma]\!](s)$. This state is a goal state of $\bigotimes F'$ because shrink transformations have the property $\mathbf{CONS_G}$. Let $\tilde{s} = \langle s_1, \dots, s_{k-1}, \tilde{s}_k, s_{k+1}, \dots, s_n \rangle$. Because τ_{F} is based on α, α is based on $\sim$ and $s_k \sim \tilde{s}_k$, Σ maps s and $\tilde{s}$ to the same abstract state. Therefore, we also have $s' = [\![\Sigma]\!](\tilde{s})$ and hence $\tilde{s} \in [\![\Sigma]\!]^{-1}(s')$. Using $\mathbf{REF_G}$, we get that $\tilde{s}$ is a goal state, from which it follows that $\tilde{s}_k$ is a goal state of Θ_k.

2. **BISIM2** *given* $\mathbf{REF_T}$*:* Consider a label ℓ and states $s_k, \tilde{s}_k, t_k \in \Theta_k$ with $s_k \sim \tilde{s}_k$ and $s_k \xrightarrow{\ell} t_k \in \Theta_k$. We must show that there exists $\tilde{t}_k \in \Theta_k$ with $\tilde{s}_k \xrightarrow{\ell} \tilde{t}_k \in \Theta_k$ and $t_k \sim \tilde{t}_k$.

   Because ℓ is not dead in $\bigotimes F$, ℓ is not dead in any of the factors of F, and therefore there exist transitions with label ℓ in every factor of F. For all $i \neq k$, let $s_i \xrightarrow{\ell} t_i \in \Theta_i$. Let $s = \langle s_1, \dots, s_n \rangle$ and $t = \langle t_1, \dots, t_n \rangle$. Then $s \xrightarrow{\ell} t \in \bigotimes F$.

   Let $\tilde{s} = \langle s_1, \dots, s_{k-1}, \tilde{s}_k, s_{k+1}, \dots, s_n \rangle$. As in the previous part of the proof, Σ maps s and $\tilde{s}$ to the same abstract state $s' = [\![\Sigma]\!](s) = [\![\Sigma]\!](\tilde{s})$. Let $t' = [\![\Sigma]\!](t)$. Because of $\mathbf{REF_T}$, there exists a state $\tilde{t}$ such that $\tilde{s} \xrightarrow{\ell} \tilde{t} \in \bigotimes F$ and $\tilde{t} \in [\![\Sigma]\!]^{-1}(t')$. Let $\tilde{t}_k$ be the k-th component of $\tilde{t}$.

   From the definition of transitions in products and $\tilde{s} \xrightarrow{\ell} \tilde{t} \in \bigotimes F$, we get $\tilde{s}_k \xrightarrow{\ell} \tilde{t}_k \in \Theta_k$. From $\tilde{t} \in [\![\Sigma]\!]^{-1}(t')$ and the fact that Σ is based on $\sim$, we get $\tilde{t}_k \sim t_k$.

3. *Bisimulation given exact:* If τ_{F} is exact, it satisfies $\mathbf{REF_G}$ and $\mathbf{REF_T}$. Together with the previous two parts of the theorem, this shows that $\sim$ is a bisimulation. □

Theorem 11. *Let τ_{F} be a shrink transformation of a factored transition system that is not trivially unsolvable and has no dead labels. Let $\sim$ be the equivalence relation on which the local abstraction of τ_{F} is based. Then:*

- τ_{F} *satisfies* $\mathbf{CONS} + \mathbf{IND} + \mathbf{REF_C} + \mathbf{LOC_{\leq}}$.
- *If τ_{F} is h-preserving, then it additionally satisfies* $\mathbf{LOC_{\geq}}$ *(and hence also* $\mathbf{LOC_{=}}$*).*
- τ_{F} *satisfies* $\mathbf{REF_G}$ *iff $\sim$ satisfies* **BISIM1**.
- τ_{F} *satisfies* $\mathbf{REF_T}$ *iff $\sim$ satisfies* **BISIM2**.
- τ_{F} *is exact iff $\sim$ is a bisimulation.*
- *If $\sim$ is a bisimulation, then τ_{F} is h-preserving.*

Proof. Follows from Theorems 7–10. □

Appendix D. Proofs of Theorems in Section 6

Theorem 12. *All merge transformations are exact induced, i.e., they satisfy* $\mathbf{CONS} + \mathbf{IND} + \mathbf{REF}$.

Proof. Let F and F' be factored transition systems with the same label sets and label costs. Let τ_{F} be a merge transformation of F into F'.

From $\bigotimes F$ being defined as the product over all transition systems in F and $\otimes$ being a commutative and associative operator, it is clear that replacing two factors from F by their product does not change $\bigotimes F$, i.e., $\bigotimes F$ and $\bigotimes F'$ are equivalent. (The underlying graphs are isomorphic in the sense that the only differences are the names of states.) With $\lambda = \text{id}$ and Σ mapping corresponding states of $\bigotimes F$ and $\bigotimes F'$, we immediately have that τ_F is exact induced. $\square$

Theorem 13. *All merge transformations are locally nondecreasing, i.e., they satisfy* **LOC$_{\geq}$**.

Proof. Consider a merge transformation τ_F of F into F' that merges factors Θ_j and Θ_k, and let $s \in \mathcal{A}(F)$ be a state. We must show $h_F^{\text{loc},\tau_F}(s) \geq h_F^{\text{mf}}(s)$. Let $\Theta_\otimes = \Theta_j \otimes \Theta_k$. By the definition of local and max-factor heuristics and of merge transformations, this holds trivially if the factor maximizing the factor heuristic in $h_F^{\text{mf}}(s)$ is different from Θ_j and Θ_k because these factors are preserved unchanged and with the same state mapping by τ_F. Therefore, it remains to show $h^*_{\Theta_\otimes}(\langle s[\Theta_j], s[\Theta_k]\rangle) \geq \max\{h^*_{\Theta_j}(s[\Theta_j]), h^*_{\Theta_k}(s[\Theta_k])\}$.

Let $\pi = \langle \ell_1, \dots, \ell_n \rangle$ be a minimum-cost path in $\Theta_\otimes$ from $\langle s[\Theta_j], s[\Theta_k]\rangle$ to some goal state $s^{\text{G}}_\otimes = \langle s^{\text{G}}_j, s^{\text{G}}_k \rangle$ of $\Theta_\otimes$. By the definition of products, π is also a path in Θ_j from $s[\Theta_j]$ to s^{G}_j, which is a goal state of Θ_j, and a path in Θ_k from $s[\Theta_k]$ to s^{G}_k, which is a goal state of Θ_k. Therefore, the minimum-cost paths of Θ_j and Θ_k cannot have larger cost than π. This immediately gives us $h^*_{\Theta_\otimes}(\langle s[\Theta_j], s[\Theta_k]\rangle) \geq \max\{h^*_{\Theta_j}(s[\Theta_j]), h^*_{\Theta_k}(s[\Theta_k])\}$ for all states $s \in \mathcal{A}(F)$ as desired. $\square$

Appendix E. Proofs of Theorems in Section 7

Theorem 14. *All label reduction transformations are abstractions, i.e., they satisfy* **CONS**. *They also satisfy* **IND$_{\text{S+L+C+I+G}}$**, *are goal-refinable, i.e., satisfy* **REF$_\text{G}$**, *and are locally nonincreasing, i.e., satisfy* **LOC$_{\leq}$**.

Proof. Let $F = \langle \Theta_1, \dots, \Theta_n \rangle$ and $F' = \langle \Theta'_1, \dots, \Theta'_n \rangle$ be factored transition systems, and let L be the label set and c be the label cost function of F. Let $\tau_F = \langle F', \Sigma, \lambda \rangle$ be a label reduction of F into F'.

By the definition of label reductions, we have $\Theta'_i = \Theta_i^{\lambda,c'}$ for all $1 \leq i \leq n$, where $c'(\ell') = \min_{\ell \in \lambda^{-1}(\ell')} c(\ell)$ for all $\ell' \in \lambda(L)$. Furthermore, $\Sigma = \langle \pi_1, \dots, \pi_n \rangle$ is the identity mapping. We simplify the notation by dropping the use of Σ and Σ^{-1} throughout. Let further $\bigotimes F = \langle S, L, c, T, S_\text{I}, S_\text{G} \rangle$ and $\bigotimes F' = \langle S', L', c', T', S'_\text{I}, S'_\text{G} \rangle$.

Let $\tau = \langle \bigotimes F', [\![\Sigma]\!], \lambda \rangle$ be the transformation of $\bigotimes F$ into $\bigotimes F'$ induced by τ_F. We must show that τ satisfies **CONS$_\text{S}$**, **CONS$_\text{L}$**, **CONS$_\text{C}$**, **CONS$_\text{T}$**, **CONS$_\text{I}$**, **CONS$_\text{G}$**, **IND$_\text{S}$**, **IND$_\text{L}$**, **IND$_\text{C}$**, **IND$_\text{T}$**, **IND$_\text{I}$**, **IND$_\text{G}$** and **REF$_\text{G}$**, and that τ_F satisfies **LOC$_{\leq}$**.

CONS$_\text{S}$ $[\![\Sigma]\!]$ is the identity function.

CONS$_\text{L}$ λ is a total function.

CONS$_\text{C}$ Consider $\ell \in L$. By the definition of c', $c'(\lambda(\ell)) = \min_{\tilde{\ell} \in \lambda^{-1}(\lambda(\ell))} c(\tilde{\ell})$. Because $\ell \in \lambda^{-1}(\lambda(\ell))$, we get $c'(\lambda(\ell)) \leq c(\ell)$, which shows that τ satisfies **CONS$_\text{C}$**.

CONS$_\text{T}$ Consider a transition $\langle s_1, \dots, s_n \rangle \xrightarrow{\ell} \langle t_1, \dots, t_n \rangle \in \bigotimes F$. By the definition of products, we have $s_i \xrightarrow{\ell} t_i \in \Theta_i$ for all $1 \leq i \leq n$. From $\Theta'_i = \Theta_i^{\lambda,c'}$ and the definition of $\Theta_i^{\lambda,c'}$, we get $s_i \xrightarrow{\lambda(\ell)} t_i \in \Theta'_i$ for all $1 \leq i \leq n$. Finally, again by the definition of products, we have $\langle s_1, \dots, s_n \rangle \xrightarrow{\lambda(\ell)} \langle t_1, \dots, t_n \rangle \in \bigotimes F'$, which shows that τ satisfies **CONS$_\text{T}$**.

CONS$_\text{I}$ $[\![\Sigma]\!]$ is the identity function and $S'_\text{I} = S_\text{I}$.

CONS$_\mathbf{G}$ $[\![\Sigma]\!]$ is the identity function and $S'_\mathrm{G} = S_\mathrm{G}$.

IND$_\mathbf{S}$ $[\![\Sigma]\!]$ is the identity function.

IND$_\mathbf{L}$ λ is surjective ($L' = \lambda(L)$).

IND$_\mathbf{C}$ Consider $\ell' \in L'$. By the definition of label reductions, $c'(\ell') = \min_{\ell \in \lambda^{-1}(\ell')} c(\ell)$, and hence $c'(\ell') = c(\ell)$ for some label $\ell \in \lambda^{-1}(\ell')$, which shows that τ satisfies **IND$_\mathbf{C}$**.

IND$_\mathbf{I}$ $[\![\Sigma]\!]$ is the identity function and $S'_\mathrm{I} = S_\mathrm{I}$.

IND$_\mathbf{G}$ $[\![\Sigma]\!]$ is the identity function and $S'_\mathrm{G} = S_\mathrm{G}$.

REF$_\mathbf{G}$ $[\![\Sigma]\!]$ is the identity function and $S'_\mathrm{G} = S_\mathrm{G}$.

LOC$_\leq$ We need to show $h_F^{\mathrm{loc},\tau_\mathrm{F}}(s) \leq h_F^{\mathrm{mf}}(s)$ for all states $s \in \mathcal{A}(F)$, which translates to showing $h_{F'}^{\mathrm{mf}}([\![\Sigma]\!](s)) \leq h_F^{\mathrm{mf}}(s)$ by the definition of local heuristics. Again note that $[\![\Sigma]\!]$ is the identity function. By the definition of max-factor heuristics, we have $h_{F'}^{\mathrm{mf}} = \max_{1 \leq i \leq n} h'_i$ where $h'_i(s) = h^*_{\Theta'_i}(s[\Theta'_i])$ and $h_F^{\mathrm{mf}} = \max_{1 \leq i \leq n} h_i$ where $h_i(s) = h^*_{\Theta_i}(s[\Theta_i])$. Thus, we need to show $\max_{1 \leq i \leq n} h'_i(s) \leq \max_{1 \leq i \leq n} h_i(s)$. It suffices to show $h'_i(s) \leq h_i(s)$ for all $1 \leq i \leq n$.

It is easy to see that this holds: every s-plan π with label sequence $\langle \ell_1, \dots, \ell_k \rangle$ in Θ_i has a corresponding s-plan π' with label sequence $\langle \lambda(\ell_1), \dots, \lambda(\ell_k) \rangle$ in $\Theta_i^{\lambda,c'}$. With $c'(\lambda(\ell)) \leq c(\ell)$ (because of **CONS$_\mathbf{C}$**, shown above), we get $c'(\pi') \leq c(\pi)$, from which $h'_i(s) \leq h_i(s)$ follows. □

Theorem 15. *Consider a label reduction transformation τ_F of a factored transition system with cost function c for label mapping λ. τ_F is cost-refinable (**REF$_\mathbf{C}$**) iff λ only combines labels of the same cost, i.e., $\lambda(\ell) = \lambda(\ell')$ implies $c(\ell) = c(\ell')$ for all ℓ, ℓ'.*

Proof. Immediate from the definitions of label reduction transformations and **REF$_\mathbf{C}$**. □

Theorem 16. *Consider a label reduction transformation τ_F of a factored transition system with cost function c for label mapping λ. If λ only combines labels of the same cost, then τ_F is locally non-decreasing (**LOC$_\geq$**).*

Proof. As discussed in the proof for **LOC$_\leq$** (Theorem 14), the only differences between the local heuristics before and after label reduction are due to the changed label costs. If only labels of the same cost are combined, we have $h'_i(s) = h_i(s)$ in that proof, which shows **LOC$_\geq$** (and of course also **LOC$_=$**). □

Theorem 17. *Let $F = \langle \Theta_1, \dots, \Theta_n \rangle$ be a factored transition system with labels L. Consider the label reduction transformation τ_F of F for label mapping λ. Then τ_F satisfies **IND$_\mathbf{T}$** iff τ_F satisfies **REF$_\mathbf{T}$**, and it satisfies these properties iff for all transformed labels $\ell' \in \lambda(L)$:*

$$\bigcup_{\ell \in \lambda^{-1}(\ell')} \prod_{i=1}^{n} \mathcal{T}(\Theta_i, \ell) = \prod_{i=1}^{n} \bigcup_{\ell \in \lambda^{-1}(\ell')} \mathcal{T}(\Theta_i, \ell).$$

Proof. Recall $F = \langle \Theta_1, \dots, \Theta_n \rangle$, and let $F' = \langle \Theta'_1, \dots, \Theta'_n \rangle$. By the definition of label reductions, we have $\Theta'_i = \Theta_i^{\lambda,c'}$ for all $1 \leq i \leq n$ for a cost function c' that is not relevant for this proof. Let $\Theta = \bigotimes F = \langle S, L, c, T, S_\mathrm{I}, S_\mathrm{G} \rangle$ and $\Theta' = \bigotimes F' = \langle S', L', c', T', S'_\mathrm{I}, S'_\mathrm{G} \rangle$. The transformation induced by τ_F is $\tau = \langle \Theta', \lambda, \sigma \rangle$, where $\sigma = \mathrm{id}$ from the definition of label reductions.

From Definition 8, we recall:

(A) τ satisfies $\mathbf{IND_T}$ iff $\forall s' \xrightarrow{\ell'} t' \in T' \; \exists s \xrightarrow{\ell} t \in T$: $s \in \sigma^{-1}(s') \wedge t \in \sigma^{-1}(t') \wedge \ell \in \lambda^{-1}(\ell')$.

(B) τ satisfies $\mathbf{REF_T}$ iff $\forall s' \xrightarrow{\ell'} t' \in T' \; \forall s \in \sigma^{-1}(s') \; \exists s \xrightarrow{\ell} t \in T$: $t \in \sigma^{-1}(t') \wedge \ell \in \lambda^{-1}(\ell')$.

Because $\sigma = \mathrm{id}$, we can simplify these definitions: we must have $s = s'$ and $t = t'$ in both properties and can therefore drop $s \in \sigma^{-1}(s')$ ($t \in \sigma^{-1}(t')$) and replace s by s' (t by t'). We obtain the same simplified condition in both cases, namely $\forall s' \xrightarrow{\ell'} t' \in T' \; \exists s' \xrightarrow{\ell} t' \in T$: $\ell \in \lambda^{-1}(\ell')$. In particular, this shows that a label reduction is transition-induced iff it is transition-refinable. We rewrite the condition into the following more idiomatic form:

(C) τ satisfies $\mathbf{IND_T}$ (= $\mathbf{REF_T}$) iff $\forall s' \xrightarrow{\ell'} t' \in T' \; \exists \ell \in \lambda^{-1}(\ell')$: $s' \xrightarrow{\ell} t' \in T$.

We still need to show that (C) is equivalent to the characterization in the theorem, i.e., (C) holds iff for all $\ell' \in \lambda(L)$, we have *LHS*(ℓ') = *RHS*(ℓ'), where *LHS*$(\ell') = \bigcup_{\ell \in \lambda^{-1}(\ell')} \prod_{i=1}^{n} \mathcal{T}(\Theta_i, \ell)$ and *RHS*$(\ell') = \prod_{i=1}^{n} \bigcup_{\ell \in \lambda^{-1}(\ell')} \mathcal{T}(\Theta_i, \ell)$. One of these inclusions is trivial: *LHS*$(\ell') \subseteq$ *RHS*(ℓ') always holds by basic set theory. (This is analogous to the logical entailment $\exists x \forall y P(x,y) \models \forall y \exists x P(x,y)$.) Therefore, it remains to show that condition (C) holds iff $\forall \ell' \in \lambda(L)$: *RHS*$(\ell') \subseteq$ *LHS*(ℓ'). We show this in two parts.

1. Proof that if (C) holds, then $\forall \ell' \in \lambda(L)$: *RHS*$(\ell') \subseteq$ *LHS*(ℓ'):

   Consider $\ell' \in \lambda(L)$ and $p \in$ *RHS*(ℓ'). We must show $p \in$ *LHS*(ℓ').

   From the definition of *RHS*(ℓ'), we have $p = \langle \langle s_1, t_1 \rangle, \dots, \langle s_n, t_n \rangle \rangle$ with $\langle s_i, t_i \rangle \in \bigcup_{\ell \in \lambda^{-1}(\ell')} \mathcal{T}(\Theta_i, \ell)$ for all $1 \leq i \leq n$. From the definition of label reduction, we have $\langle s_i, t_i \rangle \in \mathcal{T}(\Theta'_i, \ell')$ for all $1 \leq i \leq n$. From the definition of synchronized products, this implies $\langle s', t' \rangle \in \mathcal{T}(\Theta', \ell')$, where $s' = \langle s_1, \dots, s_n \rangle$ and $t' = \langle t_1, \dots, t_n \rangle$. In other words, we get $s' \xrightarrow{\ell'} t' \in T'$. Applying (C), we get $\exists \ell \in \lambda^{-1}(\ell')$: $s' \xrightarrow{\ell} t' \in T$. Let $\ell \in \lambda^{-1}(\ell')$ be a label such that $s' \xrightarrow{\ell} t' \in T$. From the definition of synchronized products, we obtain $\langle s_i, t_i \rangle \in \mathcal{T}(\Theta_i, \ell)$ for all $1 \leq i \leq n$. By definition of Cartesian products, this means $p \in \prod_{i=1}^{n} \mathcal{T}(\Theta_i, \ell)$ (recall that $p = \langle \langle s_1, t_1 \rangle, \dots, \langle s_n, t_n \rangle \rangle$). Because $\ell \in \lambda^{-1}(\ell')$, this implies $p = \bigcup_{\ell \in \lambda^{-1}(\ell')} \prod_{i=1}^{n} \mathcal{T}(\Theta_i, \ell) =$ *LHS*(ℓ').

2. Proof that if (C) does not hold, then it is not the case that $\forall \ell' \in \lambda(L)$: *RHS*$(\ell') \subseteq$ *LHS*(ℓ'):

   We can use that a counterexample to (C) exists and must show that there exists $\ell' \in \lambda(L)$ such that *RHS*$(\ell') \not\subseteq$ *LHS*(ℓ')), i.e., we must find an element $p \in$ *RHS*(ℓ') with $p \notin$ *LHS*(ℓ').

   Because (C) does not hold, there exists a transition $s' \xrightarrow{\ell'} t' \in T'$ such that for all $\ell \in \lambda^{-1}(\ell')$, $s' \xrightarrow{\ell} t' \notin T$ (*). Let $s' \xrightarrow{\ell'} t' \in T'$ be such a transition. Let $s' = \langle s_1, \dots, s_n \rangle$ and $t' = \langle t_1, \dots, t_n \rangle$. Let $p = \langle \langle s_1, t_1 \rangle, \dots, \langle s_n, t_n \rangle \rangle$. We will show $p \in$ *RHS*(ℓ') and $p \notin$ *LHS*(ℓ').

   (a) Proof that $p \in$ *RHS*(ℓ'):

Because $s' \xrightarrow{\ell'} t' \in T'$, we have $\langle s', t' \rangle \in \mathcal{T}(\Theta', \ell')$. From the definition of synchronized products, this means $\langle s_i, t_i \rangle \in \mathcal{T}(\Theta'_i, \ell')$ for all $1 \leq i \leq n$. From the definition of label reduction, this means that for all $1 \leq i \leq n$, there exists $\ell_i \in \lambda^{-1}(\ell')$ with $\langle s_i, t_i \rangle \in \mathcal{T}(\Theta_i, \ell_i)$. Hence, for all $1 \leq i \leq n$, we have $\langle s_i, t_i \rangle \in \bigcup_{\ell \in \lambda^{-1}(\ell')} \mathcal{T}(\Theta_i, \ell)$. From the definition of Cartesian products and recalling that $p = \langle \langle s_1, t_1 \rangle, \dots, \langle s_n, t_n \rangle \rangle$, we get $p \in \prod_{i=1}^{n} \bigcup_{\ell \in \lambda^{-1}(\ell')} \mathcal{T}(\Theta_i, \ell) = \mathit{RHS}(\ell')$.

(b) Proof that $p \notin \mathit{LHS}(\ell')$:
Consider any $\ell \in \lambda^{-1}(\ell')$. From (*) above, we know that $s' \xrightarrow{\ell} t' \notin T$, and therefore $\langle s', t' \rangle \notin \mathcal{T}(\Theta, \ell)$. From the definition of synchronized products, there must exist a factor Θ_i such that $\langle s_i, t_i \rangle \notin \mathcal{T}(\Theta_i, \ell)$. From the definition of Cartesian products, we obtain $p = \langle \langle s_1, t_1 \rangle, \dots, \langle s_n, t_n \rangle \rangle \notin \prod_{i=1}^{n} \mathcal{T}(\Theta_i, \ell)$. Because this is true for any $\ell \in \lambda^{-1}(\ell')$, we get $p \notin \bigcup_{\ell \in \lambda^{-1}(\ell')} \prod_{i=1}^{n} \mathcal{T}(\Theta_i, \ell) = \mathit{LHS}(\ell')$. □

Theorem 18. LABELREDUCTION-$\mathbf{IND_T}$ *is* **coNP**-*complete.*

Proof. LABELREDUCTION-$\mathbf{IND_T} \in$ **coNP***:*

Let F be the given factored transition system, let λ be the given label mapping, and let F' be the factored transition system of the label reduction of F for λ. We provide a polynomial guess-and-check algorithm for counterexamples to inducedness (in other words, a guess-and-check algorithm for finding spurious transitions). This shows membership in **coNP**. We guess states s and s' and a label ℓ of F and verify that $s \xrightarrow{\ell} s' \notin \bigotimes F$ and $s \xrightarrow{\lambda(\ell)} s' \in \bigotimes F'$. Note that verifying if a given transition is or is not present in the product ($\bigotimes F$ or $\bigotimes F'$) does not require computing the product: it is sufficient to check if the corresponding component transitions are present in the individual factors or not.

LABELREDUCTION-$\mathbf{IND_T}$ *is* **coNP**-*hard:*

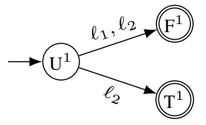
Let SAT denote the propositional satisfiability problem for formulas in CNF. SAT is known to be **NP**-hard (Cook, 1971; Karp, 1972), and therefore its complement is **coNP**-hard. The complement of SAT is the problem TAUT of deciding whether a given DNF formula is a tautology: φ is unsatisfiable iff $\neg\varphi$ is a tautology, and if φ is in CNF, then $\neg\varphi$ can easily be transformed to DNF. To prove that LABELREDUCTION-$\mathbf{IND_T}$ is **coNP**-hard, we provide the polynomial reduction TAUT $\leq_\mathrm{p}$ LABELREDUCTION-$\mathbf{IND_T}$.

We assume that DNF formulas φ are given by a set of propositional variables $X = \{X_1, \dots, X_n\}$ and a set of conjunctions $C = \{C_1, \dots, C_m\}$, where each conjunction C_j is a set of literals over X, i.e., a set of elements of the form X_i or $\neg X_i$. For example, $\{X_3, \neg X_5, \neg X_7\}$ represents the conjunction $X_3 \wedge \neg X_5 \wedge \neg X_7$.

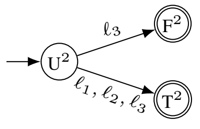
Without loss of generality, we require that each literal is *absent* in at least one conjunction C_j. For example, it is not the case that $\neg X_3 \in C_j$ for all $1 \leq j \leq m$. If $\neg X_3$ were contained in all conjunctions, then φ could not be a tautology because all satisfying assignments must map X_3 to $\mathbf{F}$. Such cases can be easily detected and mapped to any input to LABELREDUCTION-$\mathbf{IND_T}$ for which the answer is false.

Given such a DNF formula φ, we construct the input $\langle F, \lambda \rangle$ for LABELREDUCTION-$\mathbf{IND_T}$ as follows:

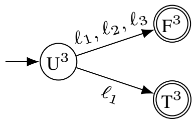
- $F = \langle \Theta^1, \dots, \Theta^n \rangle$, where $\Theta^i = \langle S^i, L, c, T^i, S^i_\mathrm{I}, S^i_\mathrm{G} \rangle$ with
  - $S^i = \{\mathrm{U}^i, \mathrm{F}^i, \mathrm{T}^i\}$



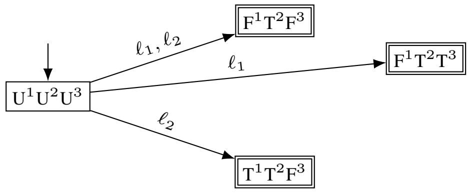
(a) Factor Θ_1.



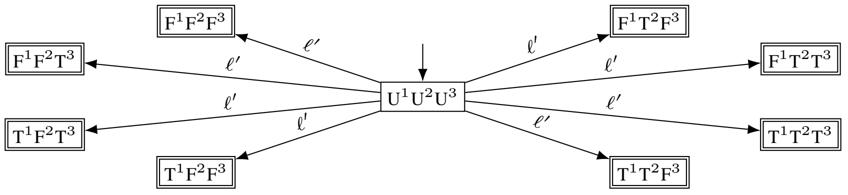
(b) Factor Θ_2.



(c) Factor Θ_3.



(d) $\bigotimes F$ (without states with no transitions).



(e) $\bigotimes F'$ (without states with no transitions).

Figure 20: Illustration of the reduction in Theorem 18 for $\varphi = (\neg X_1 \land X_2) \lor (X_2 \land \neg X_3) \lor (X_1 \land \neg X_1 \land X_3)$. Top: factored transition system F with three factors Θ^1, Θ^2, and Θ^3. Middle: $\bigotimes F$. Bottom: $\bigotimes F'$, where F' is obtained from F by the label reduction mapping all labels to ℓ'. For better readability, the products only show states with incident transitions and use rectangles instead of the usual circles for states.

  - $L = \{\ell_1, \dots, \ell_m\}$
  - $c = \{\ell_j \mapsto 1 \mid 1 \leq j \leq m\}$
  - $T^i = \{\mathrm{U}^i \xrightarrow{\ell_j} \mathrm{F}^i \mid 1 \leq j \leq m, X_i \notin C_j\} \cup \{\mathrm{U}^i \xrightarrow{\ell_j} \mathrm{T}^i \mid 1 \leq j \leq m, \neg X_i \notin C_j\}$
  - $S^i_\mathrm{I} = \{\mathrm{U}^i\}$
  - $S^i_\mathrm{G} = \{\mathrm{F}^i, \mathrm{T}^i\}$

- $\lambda = \{\ell_j \mapsto \ell' \mid 1 \leq j \leq m\}$, i.e., all labels are mapped to the same (new) label ℓ'.

Clearly, F and λ can be computed from φ in polynomial time. Let $\tau_\mathrm{F} = \langle F', \Sigma, \lambda\rangle$ be the label reduction of F for λ.

Many of the details of the construction do not matter. For example, the initial and goal states are not important, and also the local states of the factors are not really relevant. The only property of the factors that is important for the reduction is that each factor can have two kinds of transitions (in our case, $\mathrm{U}^i \to \mathrm{F}^i$ and $\mathrm{U}^i \to \mathrm{T}^i$), and the key aspect of the reduction is defining with which labels these possible transitions occur. Specifically, the reduction is designed in such a way that the local transition $\mathrm{U}^i \xrightarrow{\ell_j} \mathrm{F}^i$ exists iff conjunction C_j is consistent with X_i being false (i.e., X_i occurs negatively in C_j or does not occur at all in C_j). Similarly, the local transition $\mathrm{U}^i \xrightarrow{\ell_j} \mathrm{T}^i$ exists iff conjunction C_j is consistent with X_i being true.

To illustrate the construction, consider the following example. Let $X = \{X_1, X_2, X_3\}$ and let $C = \{C_1, C_2, C_3\}$ with $C_1 = \{\neg X_1, X_2\}$, $C_2 = \{X_2, \neg X_3\}$, and $C_3 = \{X_1, \neg X_1, \neg X_3\}$. In other words, $\varphi = (\neg X_1 \wedge X_2) \vee (X_2 \wedge \neg X_3) \vee (X_1 \wedge \neg X_1 \wedge X_3)$. Figure 20 shows the factored transition system F (top), the product $\bigotimes F$ (middle) and the product $\bigotimes F'$ after label reduction (bottom).

We write truth assignments like $\{X_1 \mapsto \mathbf{T}, X_2 \mapsto \mathbf{T}, X_3 \mapsto \mathbf{F}\}$ with the short-hand notation **TTF**. Each truth assignment corresponds to a product state in an obvious way. For example, **TTF** corresponds to $\mathrm{T}^1\mathrm{T}^2\mathrm{F}^3$. Figure 20(d) shows that $\bigotimes F$ has transitions from the central state $\mathrm{U}^1\mathrm{U}^2\mathrm{U}^3$ to the three states $\mathrm{F}^1\mathrm{T}^2\mathrm{F}^3$, $\mathrm{F}^1\mathrm{T}^2\mathrm{T}^3$, and $\mathrm{T}^1\mathrm{T}^2\mathrm{F}^3$. These correspond to the truth assignments **FTF**, **FTT**, and **TTF**, which are exactly the satisfying assignments of φ. Moreover, the transition labels in Figure 20(d) show *why* the given states correspond to satisfying assignments. For example, the incoming transitions of $\mathrm{F}^1\mathrm{T}^2\mathrm{F}^3$ have the labels ℓ_1 and ℓ_2 because **FTF** satisfies the two conjunctions C_1 and C_2 in φ.

We see that Figure 20(e) has additional transitions not corresponding to ones in $\bigotimes F$. All eight states that correspond to truth assignments have an incoming transition, so there are five spurious transitions. Consequently, this label reduction is *not* transition-induced, which is as desired by the construction because φ is not a tautology. More generally, every spurious transition (counterexample to $\mathbf{IND_T}$) in $\bigotimes F'$ corresponds to a non-satisfying assignment of φ (counterexample to tautology).

It remains to formally prove this relationship. Specifically, we show the reduction property: φ is a tautology iff τ_{F} is transition-induced. Recalling Definition 8 and taking into account that the state mapping of label reductions is the identity, τ_{F} is transition-induced iff $\forall s' \xrightarrow{\ell'} t' \in \bigotimes F'\ \exists \ell \in \lambda^{-1}(\ell')$: $s' \xrightarrow{\ell} t' \in \bigotimes F$.

- $(\Rightarrow)$: *If φ is a tautology, then τ_{F} is transition-induced.*

  Consider $s' \xrightarrow{\ell'} t' \in \bigotimes F'$. We must show that the transition is not spurious, i.e., that $\exists \ell \in \lambda^{-1}(\ell')$: $s' \xrightarrow{\ell} t' \in \bigotimes F$.

  From the definition of the factors, it is easy to see that all transitions of $\bigotimes F'$ are of the form $\mathrm{U}^1 \dots \mathrm{U}^n \xrightarrow{\ell'} A^1 \dots A^n$, where each A^i is either T^i or F^i. For a given transition of this form, we define the truth assignment α as follows: $\alpha(X_i) = \mathbf{T}$ if $A^i = \mathrm{T}^i$ and $\alpha(X_i) = \mathbf{F}$ if $A^i = \mathrm{F}^i$.

  Because φ is a tautology, α satisfies φ, which means that α satisfies at least one conjunction of φ. Let C_j be such a conjunction, and consider the label ℓ_j. To complete this part of the proof, we show that $U^i \xrightarrow{\ell_j} A^i \in \Theta^i$ for all $1 \leq i \leq n$, and therefore $\mathrm{U}^1 \dots \mathrm{U}^n \xrightarrow{\ell'} A^1 \dots A^n \in \bigotimes F$.

  Consider any factor Θ^i. If $A^i = \mathrm{T}^i$, then $\alpha(X_i) = \mathbf{T}$. Because α satisfies C_j, this means that the literal $\neg X_i$ *does not* occur in C_j. From the definition of T^i, we get $\mathrm{U}^i \xrightarrow{\ell_j} \mathrm{T}^i \in T^i$, as desired. The case $A^i = \mathrm{F}^i$ is symmetric.

- $(\Leftarrow)$ by contraposition: *If φ is not a tautology, then τ_{F} is not transition-induced.*

  Because φ is not a tautology, there exist truth assignments that fail to satisfy φ. Let α be such a truth assignment. For $1 \leq i \leq n$, set $A^i = \mathrm{T}^i$ if $\alpha(X_i) = \mathbf{T}$ and $A^i = \mathrm{F}^i$ if $\alpha(X_i) = \mathbf{F}$. We prove that τ_{F} is not transition-induced by showing that (A) $\mathrm{U}^1 \dots \mathrm{U}^n \xrightarrow{\ell'} A^1 \dots A^n \in \bigotimes F'$, but (B) $\mathrm{U}^1 \dots \mathrm{U}^n \xrightarrow{\ell_j} A^1 \dots A^n \notin \bigotimes F$ for all $1 \leq j \leq m$.

  For (A), we must show $\mathrm{U}^i \xrightarrow{\ell'} A^i \in \Theta'^i$ for all $1 \leq i \leq n$, where Θ'^i is the transition system induced by Θ^i and λ. Let L_i be the literal over X_i that is inconsistent with α on X_i, i.e.,

$L_i = X_i$ if $\alpha(X_i) = \mathbf{F}$ and $L_i = \neg X_i$ if $\alpha(X_i) = \mathbf{T}$. Because of our restriction to φ, no literal is contained in every conjunction of φ. Let C_j be a conjunction that does not contain L_i. By the definition of T^i, we obtain $\mathrm{U}^i \xrightarrow{\ell_j} A^i \in \Theta^i$ and therefore $\mathrm{U}^i \xrightarrow{\ell'} A^i \in \Theta'^i$.

For (B), for any $j \in \{1, \dots, m\}$, consider the conjunction C_j. Because α does not satisfy φ, it does not satisfy C_j. Therefore, there exists a literal $L \in C_j$ not satisfied by α. In the case where L is a negative literal $\neg X_i$, it follows that $\alpha(X_i) = \mathbf{T}$ (because L is not satisfied by α) and therefore $A^i = \mathrm{T}^i$. Consider the factor Θ_i. Because $\neg X_i \in C_j$, we have $\mathrm{U}^i \xrightarrow{\ell_j} \mathrm{T}^i \notin \Theta^i$ from the definition of T_i. With $A^i = \mathrm{T}^i$, this implies $\mathrm{U}^1 \dots \mathrm{U}^n \xrightarrow{\ell_j} A^1 \dots A^n \notin \bigotimes F$, concluding the proof. The case where L is a positive literal is symmetric. □

Theorem 19. *Let F be a factored transition system with label set L. Let λ be a label mapping defined on L with $\lambda(\ell_1) = \lambda(\ell_2) = \ell_{12}$ for $\ell_1, \ell_2 \in L$, $\ell_{12} \notin L$, and $\lambda(\ell) = \ell$ for all $\ell \in L \setminus \{\ell_1, \ell_2\}$. The (atomic) label reduction of F for λ satisfies* $\mathbf{IND_T}$ *(equivalently,* $\mathbf{REF_T}$*) iff*

1. *ℓ_1 globally subsumes ℓ_2, or*
2. *ℓ_2 globally subsumes ℓ_1, or*
3. *ℓ_1 and ℓ_2 are Θ-combinable for some $\Theta \in F$, or*
4. *there exists $\Theta \in F$ such that ℓ_1 and ℓ_2 are dead in Θ.*

Proof. Let $F = \langle \Theta_1, \dots, \Theta_n \rangle$ and $F' = \langle \Theta'_1, \dots, \Theta'_n \rangle$. Let c be the label cost function of F. Let $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$ be the given label reduction. By the definition of label reductions, we have $\Theta'_i = \Theta_i^{\lambda, c'}$ for all $1 \le i \le n$, where $c'(\ell') = \min_{\ell \in \lambda^{-1}(\ell')}(c(\ell))$ for all $\ell' \in \lambda(L)$. Furthermore, $\Sigma = \langle \pi_1, \dots, \pi_n \rangle$ is the identity mapping. Let $\tau = \langle \bigotimes F', \mathrm{id}, \lambda \rangle$ be the (nonfactored) transformation induced by τ_{F}. We distinguish four cases:

(A) If neither 1. nor 2. nor 3. nor 4. holds, then τ does not satisfy $\mathbf{IND_T}$.

(B) If 1. or 2. holds, then τ satisfies $\mathbf{IND_T}$.

(C) If 3. holds, then τ satisfies $\mathbf{IND_T}$.

(D) If 4. holds, then τ satisfies $\mathbf{IND_T}$.

Using Theorem 17, τ satisfies $\mathbf{IND_T}$ iff $\bigcup_{\ell \in \lambda^{-1}(\ell')} \prod_{i=1}^n \mathcal{T}(\Theta_i, \ell) = \prod_{i=1}^n \bigcup_{\ell \in \lambda^{-1}(\ell')} \mathcal{T}(\Theta_i, \ell)$ for all transformed labels $\ell' \in \lambda(L)$. It is easy to see that the only label ℓ' for which this condition could possibly be violated is the fresh label ℓ_{12} because the set union trivializes for the other, unchanged labels. Substituting ℓ_{12} into the condition means that $\mathbf{IND_T}$ is equivalent to

$$\mathrm{LHS} = \prod_{i=1}^n \mathcal{T}(\Theta_i, \ell_1) \cup \prod_{i=1}^n \mathcal{T}(\Theta_i, \ell_2) = \prod_{i=1}^n (\mathcal{T}(\Theta_i, \ell_1) \cup \mathcal{T}(\Theta_i, \ell_2)) = \mathrm{RHS}.$$

On (A): Because 3. does not hold, we have $\mathcal{T}(\Theta, \ell_1) \neq \mathcal{T}(\Theta, \ell_2)$ for at least two factors Θ. Because 1. and 2. do not hold, among the factors with $\mathcal{T}(\Theta, \ell_1) \neq \mathcal{T}(\Theta, \ell_2)$, there must be at least one factor Θ_u with $\mathcal{T}(\Theta_u, \ell_1) \not\subseteq \mathcal{T}(\Theta_u, \ell_2)$ and a different factor Θ_v with $\mathcal{T}(\Theta_v, \ell_2) \not\subseteq \mathcal{T}(\Theta_v, \ell_1)$. Choose any local transition (in the sense of pair of local states) $t_u \in \mathcal{T}(\Theta_u, \ell_1) \setminus \mathcal{T}(\Theta_u, \ell_2)$ and any local transition $t_v \in \mathcal{T}(\Theta_v, \ell_2) \setminus \mathcal{T}(\Theta_v, \ell_1)$. Because 4. does not hold, for all factors Θ_i other that Θ_u and Θ_v, we can choose a local transition $t_i \in \mathcal{T}(\Theta_i, \ell_1) \cup \mathcal{T}(\Theta_i, \ell_2)$ (the union cannot be empty).

Consider the composite transition $t = \langle t_1, \dots, t_n \rangle$. By construction, for each $1 \le i \le n$, we have $t_i \in \mathcal{T}(\Theta_i, \ell_1)$ or $t_i \in \mathcal{T}(\Theta_i, \ell_2)$. This implies $t \in$ RHS. Because $t_u \notin \mathcal{T}(\Theta_u, \ell_2)$, we have $t \notin \prod_{i=1}^n \mathcal{T}(\Theta_i, \ell_2)$. Because $t_v \notin \mathcal{T}(\Theta_v, \ell_1)$, we have $t \notin \prod_{i=1}^n \mathcal{T}(\Theta_i, \ell_1)$. This implies $t \notin$ LHS, concluding this part of the proof.

On (B): As discussed in the proof of Theorem 17, we always have LHS $\subseteq$ RHS, so we must show RHS $\subseteq$ LHS. In Case 1., where ℓ_1 globally subsumes ℓ_2, we have:

$$\begin{aligned}
\text{RHS} &= \prod_{i=1}^n (\mathcal{T}(\Theta_i, \ell_1) \cup \mathcal{T}(\Theta_i, \ell_2)) \\
&\overset{(*)}{=} \prod_{i=1}^n \mathcal{T}(\Theta_i, \ell_1) \\
&\subseteq \prod_{i=1}^n \mathcal{T}(\Theta_i, \ell_1) \cup \prod_{i=1}^n \mathcal{T}(\Theta_i, \ell_2) \\
&= \text{LHS},
\end{aligned}$$

where (*) uses $\mathcal{T}(\Theta_i, \ell_2) \subseteq \mathcal{T}(\Theta_i, \ell_1)$. Case 2. is symmetric.

On (C): Choose the factor Θ_j such that ℓ_1 and ℓ_2 are Θ_j-combinable, which means $\mathcal{T}(\Theta_i, \ell_1) = \mathcal{T}(\Theta_i, \ell_2)$ for all other factors Θ_i. We show RHS $=$ LHS:

$$\begin{aligned}
\text{RHS} =& \prod_{i=1}^n (\mathcal{T}(\Theta_i, \ell_1) \cup \mathcal{T}(\Theta_i, \ell_2)) \\
=& \prod_{i=1}^{j-1} (\mathcal{T}(\Theta_i, \ell_1) \cup \mathcal{T}(\Theta_i, \ell_2)) \times (\mathcal{T}(\Theta_j, \ell_1) \cup \mathcal{T}(\Theta_j, \ell_2)) \times \prod_{i=j+1}^{n} (\mathcal{T}(\Theta_i, \ell_1) \cup \mathcal{T}(\Theta_i, \ell_2)) \\
\overset{(*)}{=}& \prod_{i=1}^{j-1} \mathcal{T}(\Theta_i, \ell_1) \times (\mathcal{T}(\Theta_j, \ell_1) \cup \mathcal{T}(\Theta_j, \ell_2)) \times \prod_{i=j+1}^{n} \mathcal{T}(\Theta_i, \ell_1) \\
\overset{(**)}{=}& \prod_{i=1}^{j-1} \mathcal{T}(\Theta_i, \ell_1) \times \mathcal{T}(\Theta_j, \ell_1) \times \prod_{i=j+1}^{n} \mathcal{T}(\Theta_i, \ell_1) \cup \\
& \prod_{i=1}^{j-1} \mathcal{T}(\Theta_i, \ell_1) \times \mathcal{T}(\Theta_j, \ell_2) \times \prod_{i=j+1}^{n} \mathcal{T}(\Theta_i, \ell_1) \\
\overset{(***)}{=}& \prod_{i=1}^{j-1} \mathcal{T}(\Theta_i, \ell_1) \times \mathcal{T}(\Theta_j, \ell_1) \times \prod_{i=j+1}^{n} \mathcal{T}(\Theta_i, \ell_1) \cup \\
& \prod_{i=1}^{j-1} \mathcal{T}(\Theta_i, \ell_2) \times \mathcal{T}(\Theta_j, \ell_2) \times \prod_{i=j+1}^{n} \mathcal{T}(\Theta_i, \ell_2) \\
=& \prod_{i=1}^n \mathcal{T}(\Theta_i, \ell_1) \cup \prod_{i=1}^n \mathcal{T}(\Theta_i, \ell_2)
\end{aligned}$$

=LHS,

where (*) and (***) use $\mathcal{T}(\Theta_i, \ell_1) = \mathcal{T}(\Theta_i, \ell_2)$ for all $\Theta_i \neq \Theta_j$ and (**) is basic set algebra.

On (D): Let Θ be a factor in which ℓ_1 and ℓ_2 are dead. It is easy to see that LHS $= \emptyset =$ RHS because Cartesian products involving empty sets are empty and the union of two empty sets are empty. □

Theorem 20. *A label reduction transformation of a factored transition system F with labels L and label costs c that only combines two labels $\ell_1, \ell_2 \in L$ is an exact induced transformation, i.e., satisfies* **CONS** + **IND** + **REF**, *iff $c(\ell_1) = c(\ell_2)$ and*

1. *ℓ_1 globally subsumes ℓ_2, or*
2. *ℓ_2 globally subsumes ℓ_1, or*
3. *ℓ_1 and ℓ_2 are Θ-combinable for some $\Theta \in F$, or*
4. *there exists $\Theta \in F$ such that ℓ_1 and ℓ_2 are dead in Θ.*

General label reductions always satisfy **CONS** + $\mathbf{IND_{S+L+C+I+G}}$ + $\mathbf{REF_G}$ *and* $\mathbf{LOC_{\leq}}$. *They satisfy* $\mathbf{REF_C}$ *iff they only combine labels of the same cost. Cost-refinable label reductions additionally satisfy* $\mathbf{LOC_{=}}$. *Testing* $\mathbf{IND_T}$ *and* $\mathbf{REF_T}$ *is* **coNP***-complete for general label reductions.*

Proof. This follows from Theorems 14, 15, 16, 18, and 19. □

Appendix F. Proofs of Theorems in Section 8

Theorem 21. *Let X be any of the properties of transformations from Definition 37. Let τ be a transformation of transition system Θ into transition system Θ' with property X, and let τ' be a transformation of Θ' into transition system Θ'' with property X. Then the composed transformation $\tau'' = \tau' \circ \tau$ also has the property X if τ additionally satisfies the following side conditions (depending on X):*

- $\mathbf{CLOS}_{\mathit{pred}}$: *side conditions* $\mathbf{CONS_{L+T}}$
- $\mathbf{CLOS}^{\rightarrow}_{\mathit{pred}}$: *side conditions* $\mathbf{CONS_{L+T+I}}$
- $\mathbf{KEEP_G}$: *side condition* $\mathbf{CONS_G}$
- $\mathbf{KEEP}^{\rightarrow}_{\mathbf{G}}$: *side conditions* $\mathbf{CONS_{L+T+I+G}}$ + $\mathbf{CLOS}^{\rightarrow}_{\mathit{pred}}$

In all cases, all side conditions on τ are necessary in the sense that if we drop any one of them, the result no longer holds. For example, if we only require that τ and τ' are $\mathbf{CLOS}^{\rightarrow}_{\mathit{pred}}$ *and τ is* $\mathbf{CONS_{L+T}}$ *or* $\mathbf{CONS_{L+I}}$ *or* $\mathbf{CONS_{T+I}}$, *then τ'' is not necessarily* $\mathbf{CLOS}^{\rightarrow}_{\mathit{pred}}$.

Proof. Let $\Theta = \langle S, L, c, T, S_\mathrm{I}, S_\mathrm{G}\rangle$, $\Theta' = \langle S', L', c', T', S'_\mathrm{I}, S'_\mathrm{G}\rangle$, and $\Theta'' = \langle S'', L'', c'', T'', S''_\mathrm{I}, S''_\mathrm{G}\rangle$. Let $\tau = \langle \Theta', \sigma, \lambda\rangle$ and $\tau' = \langle \Theta'', \sigma', \lambda'\rangle$. Then $\tau'' = \langle \Theta'', \sigma'', \lambda''\rangle$ with $\sigma'' = \sigma' \circ \sigma$ and $\lambda'' = \lambda' \circ \lambda$.

We remind the reader that $\mathit{dom}(\sigma'') = \mathit{dom}(\sigma) \cap \sigma^{-1}(\mathit{dom}(\sigma'))$ (cf. Definition 2). This means that $s \in \mathit{dom}(\sigma'')$ iff $s \in \mathit{dom}(\sigma)$ and $\sigma(s) \in \mathit{dom}(\sigma')$. Analogously, we have $\mathit{dom}(\lambda'') = \mathit{dom}(\lambda) \cap \lambda^{-1}(\mathit{dom}(\lambda'))$ and therefore $\ell \in \mathit{dom}(\lambda'')$ iff $\ell \in \mathit{dom}(\lambda)$ and $\lambda(\ell) \in \mathit{dom}(\lambda')$.

CLOS$_{\textit{pred}}$ We have to show *Between*$(S, dom(\sigma''))\subseteq dom(\sigma'')$. We will show that for every state $t \in dom(\sigma'')$ and all transitions $s \xrightarrow{\ell} t \in T$, we also have $s \in dom(\sigma'')$. The result then follows by repeating the argument.

So consider any $t \in dom(\sigma'')$ and $s \xrightarrow{\ell} t \in T$. From $t \in dom(\sigma'')$, we have $t \in dom(\sigma)$. Because τ satisfies **CLOS**$_{\textit{pred}}$ and $s \xrightarrow{\ell} t \in T$, we also have $s \in dom(\sigma)$. From **CONS**$_{\mathbf{L}}$ we get $\ell \in dom(\lambda)$, and with **CONS**$_{\mathbf{T}}$ we then get **(*)** $\sigma(s) \xrightarrow{\lambda(\ell)} \sigma(t) \in T'$. From $t \in dom(\sigma'')$, we also have $\sigma(t) \in dom(\sigma')$. Because τ' satisfies **CLOS**$_{\textit{pred}}$, from **(*)** we get $\sigma(s) \in dom(\sigma')$. Finally, from $s \in dom(\sigma)$ and $\sigma(s) \in dom(\sigma')$, we conclude $s \in dom(\sigma'')$.

CLOS$^{\rightarrow}_{\textit{pred}}$ We have to show *Between*$(S_{\mathrm{I}}, dom(\sigma''))\subseteq dom(\sigma'')$. All states in *Between*$(S_{\mathrm{I}}, dom(\sigma''))$ are forward-reachable, so we consider a forward-reachable state $s \in dom(\sigma'')$ and show that *Between*$(S_{\mathrm{I}}, \{s\}) \subseteq dom(\sigma'')$.

Let $\tilde{s} \in$ *Between*$(S_{\mathrm{I}}, \{s\})$. Then there exists a path $\langle s_0 \xrightarrow{\ell_1} s_1, \dots, s_{n-1} \xrightarrow{\ell_n} s_n\rangle$ with $s_0 \in S_{\mathrm{I}}$, $s_n = s$ and $s_i = \tilde{s}$ for some $0 \le i \le n$. From $s_n \in dom(\sigma'')$, we get $s_n \in dom(\sigma)$. Because τ is **CLOS**$^{\rightarrow}_{\textit{pred}}$, we get $s_i \in dom(\sigma)$ for all $0 \le i \le n$. Since τ is **CONS**$_{\mathbf{I}}$, we have **(A)** $\sigma(s_0) \in S'_{\mathrm{I}}$. Since τ is **CONS**$_{\mathbf{L}}$, we have $\ell_i \in dom(\lambda)$. Since τ is **CONS**$_{\mathbf{T}}$, we then have **(B)** $\sigma(s_{i-1}) \xrightarrow{\lambda(\ell_i)} \sigma(s_i) \in T'$ for all $1 \le i \le n$.

From $s_n \in dom(\sigma'')$, we get $\sigma(s_n) \in dom(\sigma')$. Together with **(A)**, **(B)** and τ' satisfying **CLOS**$^{\rightarrow}_{\textit{pred}}$, we get $\sigma(s_i) \in dom(\sigma')$ for all $0 \le i \le n$. With the definition of $dom(\sigma'')$, we get $s_i \in dom(\sigma'')$ for all $0 \le i \le n$ and therefore $\tilde{s} \in dom(\sigma'')$ as desired.

KEEP$_{\mathbf{G}}$ We have to show $S_{\mathrm{G}} \subseteq dom(\sigma'')$. Consider a state $s \in S_{\mathrm{G}}$. We need to show $s \in dom(\sigma'')$. Because τ satisfies **KEEP**$_{\mathbf{G}}$, we get **(A)** $s \in dom(\sigma)$, and because τ satisfies **CONS**$_{\mathbf{G}}$, we have $\sigma(s) \in S'_{\mathrm{G}}$. Because τ' satisfies **KEEP**$_{\mathbf{G}}$, we get **(B)** $\sigma(s) \in dom(\sigma')$, and hence $s \in dom(\sigma'')$ from **(A)** and **(B)**.

KEEP$^{\rightarrow}_{\mathbf{G}}$ We have to show *Between*$(S_{\mathrm{I}}, S_{\mathrm{G}}) \cap S_{\mathrm{G}} \subseteq dom(\sigma'')$, i.e., that all forward-reachable goal states $s \in S_{\mathrm{G}}$ are included in $dom(\sigma'')$. Let s be such a state.

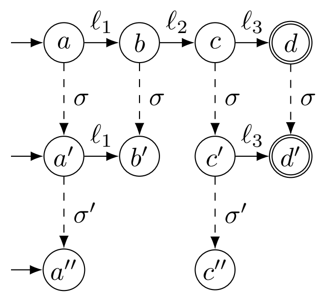
Because s is forward-reachable, there exists a path $\pi = \langle s_0 \xrightarrow{\ell_1} s_1, \dots, s_{n-1} \xrightarrow{\ell_n} s_n\rangle$ in Θ with $s_0 \in S_{\mathrm{I}}$ and $s_n = s$. Because τ satisfies **KEEP**$^{\rightarrow}_{\mathbf{G}}$ and s is a goal state, we get $s \in dom(\sigma)$. Using the same argument as in the proof for **CLOS**$^{\rightarrow}_{\textit{pred}}$, the path π is preserved by τ (including the fact that it begins in an initial state), i.e., we have $\sigma(s_0) \in S'_{\mathrm{I}}$ and $\sigma(s_{i-1}) \xrightarrow{\lambda(\ell_i)} \sigma(s_i) \in T'$ for all $1 \le i \le n$.

Because τ satisfies **CONS**$_{\mathbf{G}}$, we also have $\sigma(s) \in S'_{\mathrm{G}}$. Hence $\sigma(s)$ is a forward-reachable goal state in Θ'. Because τ' is **KEEP**$^{\rightarrow}_{\mathbf{G}}$, we get $\sigma(s) \in dom(\sigma')$.

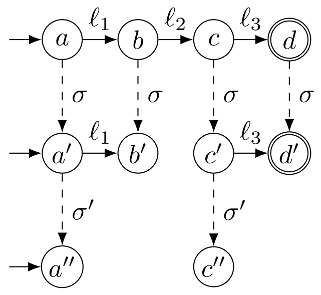
Finally, from $s \in dom(\sigma)$ and $\sigma(s) \in dom(\sigma')$, we conclude $s \in dom(\sigma'')$.

We now show that the side conditions on properties that τ must have are necessary. To do so, we provide examples of τ and τ' as in the above proofs where we remove one of the side conditions from τ such that τ'' does not have the claimed property.

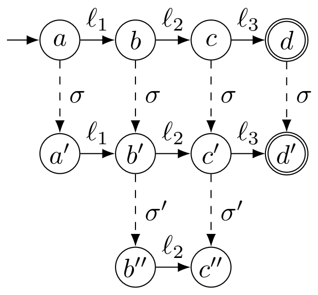
Figure 21 shows the examples. Each part illustrates the three transition systems Θ (top), Θ' (middle), and Θ'' (bottom). The transformations τ and τ' are represented through their state mappings σ and σ' which are indicated in the figures by dotted lines. The label mappings λ and λ' are



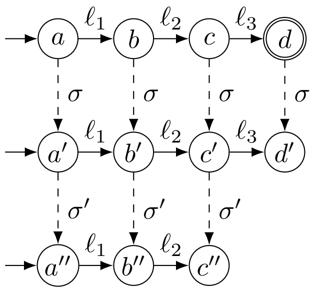
(a) $\lambda = \lambda' = \{\ell_1 \mapsto \ell_1, \ell_3 \mapsto \ell_3\}$. (Note $\ell_2 \notin dom(\lambda)$.)



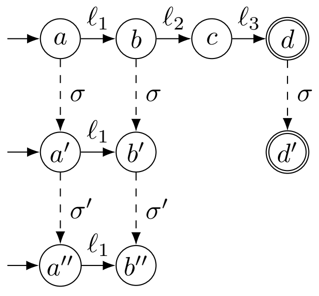
(b) $\lambda = \lambda' = \text{id}$.



(c) $\lambda = \lambda' = \text{id}$.



(d) $\lambda = \lambda' = \text{id}$.



(e) $\lambda = \lambda' = \text{id}$.

Figure 21: Each part shows example transition systems $\Theta, \Theta', \Theta''$ (top, middle, bottom) and two transformations $\tau = \langle \Theta', \sigma, \lambda \rangle$ and $\tau' = \langle \Theta'', \sigma', \lambda' \rangle$, represented by dotted lines between the transition systems indicating the state mapping. For example, in part (a), Θ has the states $\{a, b, c, d\}$, Θ' has the states $\{a', b', c', d'\}$, and $\sigma(a) = a'$. The caption of each part provides the label mapping.

defined in the caption of each part. It is worth pointing out that in all examples except Figure 21(e) τ trivially has all four properties of Definition 37 because σ is total. The composition "goes wrong" due to missing side conditions that allow the second transformation, τ', to discard important states.

CLOS$_{pred}$ *Side condition* **CONS**$_{\mathbf{L}}$ *is required:*

In Figure 21(a), τ satisfies **CONS**$_{\mathbf{T}}$ but not **CONS**$_{\mathbf{L}}$ because $\ell_2 \notin dom(\lambda)$. Because $\ell_2 \notin dom(\lambda)$, the transition $b \xrightarrow{\ell_2} c \in \Theta$ does not need to have a counterpart in Θ'.

Transformation τ has the property **CLOS**$_{pred}$ because σ is total. Transformation τ' has the property **CLOS**$_{pred}$ because $Between(S', dom(\sigma')) = Between(\{a', b', c', d'\}, \{a', c'\}) = \{a', c'\} \subseteq \{a', c'\} = dom(\sigma')$. However, $\tau'' = \tau' \circ \tau$ does not have the property **CLOS**$_{pred}$ because $Between(S, dom(\sigma'')) = Between(\{a, b, c, d\}, \{a, c\}) = \{a, b, c\} \not\subseteq \{a, c\} = dom(\sigma'')$.

Side condition **CONS**$_{\mathbf{T}}$ *is required:*

In Figure 21(b), τ satisfies **CONS**$_{\mathbf{L}}$, but not **CONS**$_{\mathbf{T}}$ because $\sigma(b) \xrightarrow{\lambda(\ell_2)} \sigma(c) \notin \Theta'$.

Transformation τ has the property **CLOS**$_{pred}$ because σ is total. Transformation τ' has the property **CLOS**$_{pred}$ because $Between(S', dom(\sigma')) = Between(\{a', b', c', d'\}, \{a', c'\}) = \{a', c'\} \subseteq \{a', c'\} = dom(\sigma')$. However, $\tau'' = \tau' \circ \tau$ does not have the property **CLOS**$_{pred}$ because $Between(S, dom(\sigma'')) = Between(\{a, b, c, d\}, \{a, c\}) = \{a, b, c\} \not\subseteq \{a, c\} = dom(\sigma'')$.

CLOS$^{\rightarrow}_{pred}$ *Side condition* **CONS**$_{\mathbf{L}}$ *is required:*

In Figure 21(a), τ satisfies **CONS**$_{\mathbf{T+I}}$ but not **CONS**$_{\mathbf{L}}$ because $\ell_2 \notin dom(\lambda)$. Because $\ell_2 \notin dom(\lambda)$, the transition $b \xrightarrow{\ell_2} c \in \Theta$ does not need to have a counterpart in Θ'.

Transformation τ has the property **CLOS**$^{\rightarrow}_{pred}$ because σ is total. Transformation τ' has the property **CLOS**$^{\rightarrow}_{pred}$ because $Between(S'_{\mathrm{I}}, dom(\sigma')) = Between(\{a'\}, \{a', c'\}) = \{a'\} \subseteq \{a', c'\} = dom(\sigma')$. However, $\tau'' = \tau' \circ \tau$ does not have the property **CLOS**$^{\rightarrow}_{pred}$ because $Between(S_{\mathrm{I}}, dom(\sigma'')) = Between(\{a\}, \{a, c\}) = \{a, b, c\} \not\subseteq \{a, c\} = dom(\sigma'')$.

Side condition **CONS**$_{\mathbf{T}}$ *is required:*

In Figure 21(b), τ satisfies **CONS**$_{\mathbf{L+I}}$, but not **CONS**$_{\mathbf{T}}$ because $\sigma(b) \xrightarrow{\lambda(\ell_2)} \sigma(c) \notin \Theta'$.

Transformation τ has the property **CLOS**$^{\rightarrow}_{pred}$ because σ is total. Transformation τ' has the property **CLOS**$^{\rightarrow}_{pred}$ because $Between(S'_{\mathrm{I}}, dom(\sigma')) = Between(\{a'\}, \{a', c'\}) = \{a'\} \subseteq \{a', c'\} = dom(\sigma')$. However, $\tau'' = \tau' \circ \tau$ does not have the property **CLOS**$^{\rightarrow}_{pred}$ because $Between(S_{\mathrm{I}}, dom(\sigma'')) = Between(\{a\}, \{a, c\}) = \{a, b, c\} \not\subseteq \{a, c\} = dom(\sigma'')$.

Side condition **CONS**$_{\mathbf{I}}$ *is required:*

In Figure 21(c), τ satisfies **CONS**$_{\mathbf{L+T}}$, but not **CONS**$_{\mathbf{I}}$ because $\sigma(a) \notin S'_{\mathrm{I}}$.

Transformation τ has the property **CLOS**$^{\rightarrow}_{pred}$ because σ is total. Transformation τ' has the property **CLOS**$^{\rightarrow}_{pred}$ because $Between(S'_{\mathrm{I}}, dom(\sigma')) = Between(\emptyset, \{b', c'\}) = \emptyset \subseteq \{b', c'\} = dom(\sigma')$. However, $\tau'' = \tau' \circ \tau$ does not have the property **CLOS**$^{\rightarrow}_{pred}$ because $Between(S_{\mathrm{I}}, dom(\sigma'')) = Between(\{a\}, \{b, c\}) = \{a, b, c\} \not\subseteq \{b, c\} = dom(\sigma'')$.

KEEP$_\mathbf{G}$ *Side condition* **CONS$_\mathbf{G}$** *is required:*

In Figure 21(d), τ does not satisfy **CONS$_\mathbf{G}$** because $\sigma(d) \notin S'_\mathrm{G}$.

Transformation τ has the property **KEEP$_\mathbf{G}$** because σ is total. Transformation τ' has the property **KEEP$_\mathbf{G}$** because $S'_\mathrm{G} = \emptyset \subseteq \{a', b', c'\} = dom(\sigma')$. However, $\tau'' = \tau' \circ \tau$ does not have the property **KEEP$_\mathbf{G}$** because $S_\mathrm{G} = \{d\} \nsubseteq \{a, b, c\} = dom(\sigma'')$.

KEEP$_\mathbf{G}^{\rightarrow}$ *Side condition* **CONS$_\mathbf{L}$** *is required:*

In Figure 21(a), τ satisfies **CONS$_\mathbf{T+I+G}$** and **CLOS$_{\mathit{pred}}^{\rightarrow}$** but not **CONS$_\mathbf{L}$** because $\ell_2 \notin dom(\lambda)$.

Transformation τ has the property **KEEP$_\mathbf{G}^{\rightarrow}$** because σ is total. Transformation τ' has the property **KEEP$_\mathbf{G}^{\rightarrow}$** because $\mathit{Between}(S'_\mathrm{I}, S'_\mathrm{G}) \cap S'_\mathrm{G} = \mathit{Between}(\{a'\}, \{d'\}) \cap \{d'\} = \emptyset \cap \{d'\} = \emptyset \subseteq \{a', c'\} = dom(\sigma')$. However, $\tau'' = \tau' \circ \tau$ does not have the property **KEEP$_\mathbf{G}^{\rightarrow}$** because $\mathit{Between}(S_\mathrm{I}, S_\mathrm{G}) \cap S_\mathrm{G} = \mathit{Between}(\{a\}, \{d\}) \cap \{d\} = \{a, b, c, d\} \cap \{d\} = \{d\} \nsubseteq \{a, c\} = dom(\sigma'')$.

Side condition **CONS$_\mathbf{T}$** *is required:*

In Figure 21(b), τ satisfies **CONS$_\mathbf{L+I+G}$** and **CLOS$_{\mathit{pred}}^{\rightarrow}$** but not **CONS$_\mathbf{T}$** because $\sigma(b) \xrightarrow{\lambda(\ell_2)} \sigma(c) \notin \Theta'$.

Transformation τ has the property **KEEP$_\mathbf{G}^{\rightarrow}$** because σ is total. Transformation τ' has the property **KEEP$_\mathbf{G}^{\rightarrow}$** because $\mathit{Between}(S'_\mathrm{I}, S'_\mathrm{G}) \cap S'_\mathrm{G} = \mathit{Between}(\{a'\}, \{d'\}) \cap \{d'\} = \emptyset \cap \{d'\} = \emptyset \subseteq \{a', c'\} = dom(\sigma')$. However, $\tau'' = \tau' \circ \tau$ does not have the property **KEEP$_\mathbf{G}^{\rightarrow}$** because $\mathit{Between}(S_\mathrm{I}, S_\mathrm{G}) \cap S_\mathrm{G} = \mathit{Between}(\{a\}, \{d\}) \cap \{d\} = \{a, b, c, d\} \cap \{d\} = \{d\} \nsubseteq \{a, c\} = dom(\sigma'')$.

Side condition **CONS$_\mathbf{I}$** *is required:*

In Figure 21(c), τ satisfies **CONS$_\mathbf{L+T+G}$** and **CLOS$_{\mathit{pred}}^{\rightarrow}$** but not **CONS$_\mathbf{I}$** because $\sigma(a) \notin S'_\mathrm{I}$.

Transformation τ has the property **KEEP$_\mathbf{G}^{\rightarrow}$** because σ is total. Transformation τ' has the property **KEEP$_\mathbf{G}^{\rightarrow}$** because $\mathit{Between}(S'_\mathrm{I}, S'_\mathrm{G}) \cap S'_\mathrm{G} = \mathit{Between}(\emptyset, \{d'\}) \cap \{d'\} = \emptyset \cap \{d'\} = \emptyset \subseteq \{b', c'\} = dom(\sigma')$. However, $\tau'' = \tau' \circ \tau$ does not have the property **KEEP$_\mathbf{G}^{\rightarrow}$** because $\mathit{Between}(S_\mathrm{I}, S_\mathrm{G}) \cap S_\mathrm{G} = \mathit{Between}(\{a\}, \{d\}) \cap \{d\} = \{a, b, c, d\} \cap \{d\} = \{d\} \nsubseteq \{b, c\} = dom(\sigma'')$.

Side condition **CONS$_\mathbf{G}$** *is required:*

In Figure 21(d), τ satisfies **CONS$_\mathbf{L+T+I}$** and **CLOS$_{\mathit{pred}}^{\rightarrow}$** but not **CONS$_\mathbf{G}$** because because $\sigma(d) \notin S'_\mathrm{G}$.

Transformation τ has the property **KEEP$_\mathbf{G}^{\rightarrow}$** because σ is total. Transformation τ' has the property **KEEP$_\mathbf{G}^{\rightarrow}$** because $\mathit{Between}(S'_\mathrm{I}, S'_\mathrm{G}) \cap S'_\mathrm{G} = \mathit{Between}(\{a'\}, \emptyset) \cap \emptyset = \emptyset \cap \emptyset = \emptyset \subseteq \{a', b', c'\} = dom(\sigma')$. However, $\tau'' = \tau' \circ \tau$ does not have the property **KEEP$_\mathbf{G}^{\rightarrow}$** because $\mathit{Between}(S_\mathrm{I}, S_\mathrm{G}) \cap S_\mathrm{G} = \mathit{Between}(\{a\}, \{d\}) \cap \{d\} = \{a, b, c, d\} \cap \{d\} = \{d\} \nsubseteq \{a, b, c\} = dom(\sigma'')$.

Side condition **CLOS$_{\mathit{pred}}^{\rightarrow}$** *is required:*

In Figure 21(e), τ satisfies **CONS$_\mathbf{L+T+I+G}$** but not **CLOS$_{\mathit{pred}}^{\rightarrow}$** because $c \notin dom(\sigma)$.

Transformation τ has the property $\mathbf{KEEP}^{\rightarrow}_{\mathbf{G}}$ because $\mathit{Between}(S_{\mathrm{I}}, S_{\mathrm{G}}) \cap S_{\mathrm{G}} = \mathit{Between}(\{a\}, \{d\}) \cap \{d\} = \{a, b, c, d\} \cap \{d\} = \{d\} \subseteq \{a, b, d\} = \mathit{dom}(\sigma)$. Transformation τ' has the property $\mathbf{KEEP}^{\rightarrow}_{\mathbf{G}}$ because $\mathit{Between}(S'_{\mathrm{I}}, S'_{\mathrm{G}}) \cap S'_{\mathrm{G}} = \mathit{Between}(\{a'\}, \{d'\}) \cap \{d'\} = \emptyset \cap \{d'\} = \emptyset \subseteq \{a', b'\} = \mathit{dom}(\sigma')$. However, $\tau'' = \tau' \circ \tau$ does not have the property $\mathbf{KEEP}^{\rightarrow}_{\mathbf{G}}$ because $\mathit{Between}(S_{\mathrm{I}}, S_{\mathrm{G}}) \cap S_{\mathrm{G}} = \mathit{Between}(\{a\}, \{d\}) \cap \{d\} = \{a, b, c, d\} \cap \{d\} = \{d\} \not\subseteq \{a, b\} = \mathit{dom}(\sigma'')$. □

Theorem 22. *Let τ be a transformation of a transition system Θ. The heuristic h^{τ} for Θ induced by τ is*

1. *goal-aware if τ satisfies* $\mathbf{CONS_G} + \mathbf{KEEP_G}$,
2. *forward-goal-aware if τ satisfies* $\mathbf{CONS_G} + \mathbf{KEEP}^{\rightarrow}_{\mathbf{G}}$,
3. *consistent if τ satisfies* $\mathbf{CONS_{L+C+T}} + \mathbf{CLOS}_{\mathit{pred}}$,
4. *forward-consistent if τ satisfies* $\mathbf{CONS_{L+C+T}} + \mathbf{CLOS}^{\rightarrow}_{\mathit{pred}}$,
5. *admissible if τ satisfies* $\mathbf{CONS_{L+C+T+G}} + \mathbf{KEEP_G} + \mathbf{CLOS}_{\mathit{pred}}$, *and*
6. *forward-admissible if τ satisfies* $\mathbf{CONS_{L+C+T+G}} + \mathbf{KEEP}^{\rightarrow}_{\mathbf{G}} + \mathbf{CLOS}^{\rightarrow}_{\mathit{pred}}$.

Proof. Let $\Theta = \langle S, L, c, T, S_{\mathrm{I}}, S_{\mathrm{G}} \rangle$ and $\Theta' = \langle S', L', c', T', S'_{\mathrm{I}}, S'_{\mathrm{G}} \rangle$.

1. Consider $s \in S_{\mathrm{G}}$. We have to show that $h^{\tau}(s) = 0$. With $\mathbf{KEEP_G}$, we get $s \in \mathit{dom}(\sigma)$. Together with $\mathbf{CONS_G}$, we have $\sigma(s) \in S'_{\mathrm{G}}$. Therefore, $h^{\tau}(s) = h^{*}_{\Theta'}(\sigma(s)) = 0$.

2. Consider $s \in S_{\mathrm{G}} \cap S_{\rightarrow}$. We have to show that $h^{\tau}(s) = 0$. Since $s \in S_{\rightarrow}$ and $s \in S_{\mathrm{G}}$, we have $s \in \mathit{Between}(S_{\mathrm{I}}, S_{\mathrm{G}})$. Thus, with $\mathbf{KEEP}^{\rightarrow}_{\mathbf{G}}$ ($\mathit{Between}(S_{\mathrm{I}}, S_{\mathrm{G}}) \cap S_{\mathrm{G}} \subseteq \mathit{dom}(\sigma)$), we get $s \in \mathit{dom}(\sigma)$. Then, with $\mathbf{CONS_G}$, we get $\sigma(s) \in S'_{\mathrm{G}}$. Therefore, $h^{\tau}(s) = h^{*}_{\Theta'}(\sigma(s)) = 0$.

3. Consider $s \xrightarrow{\ell} t \in T$. We have to show that $h^{\tau}(s) \leq c(\ell) + h^{\tau}(t)$. We distinguish two cases.

   If $t \notin \mathit{dom}(\sigma)$, then $h^{\tau}(t) = \infty$ by definition, and with that, the inequality trivially holds independently of $h^{\tau}(s)$.

   Otherwise, $t \in \mathit{dom}(\sigma)$. From $s \xrightarrow{\ell} t \in T$ and $\mathbf{CLOS}_{\mathit{pred}}$, we get that $s \in \mathit{Between}(\{s\}, \{t\}) \subseteq \mathit{Between}(S, \mathit{dom}(\sigma)) \subseteq \mathit{dom}(\sigma)$ and hence $s \in \mathit{dom}(\sigma)$. Together with $\mathbf{CONS_{L+T}}$, we have that $\sigma(s) \xrightarrow{\lambda(\ell)} \sigma(t) \in T'$. Hence $h^{*}_{\Theta'}(\sigma(s)) \leq c'(\lambda(\ell)) + h^{*}_{\Theta'}(\sigma(t))$ because the perfect heuristic is consistent. We further have $c'(\lambda(\ell)) \leq c(\lambda)$ due to $\mathbf{CONS_C}$. Put together, we get $h^{\tau}(s) = h^{*}_{\Theta'}(\sigma(s)) \leq c(\ell) + h^{*}_{\Theta'}(\sigma(t)) = c(\ell) + h^{\tau}(t)$ as desired.

4. Consider $s \xrightarrow{\ell} t \in T$ with $s, t \in S_{\rightarrow}$. We have to show that $h^{\tau}(s) \leq c(\ell) + h^{\tau}(t)$. We distinguish two cases.

   If $t \notin \mathit{dom}(\sigma)$, then $h^{\tau}(t) = \infty$ by definition, and with that, the inequality trivially holds independently of $h^{\tau}(s)$.

   Otherwise, $t \in \mathit{dom}(\sigma)$. From $\mathbf{CLOS}^{\rightarrow}_{\mathit{pred}}$, we get that $\mathit{Between}(S_{\mathrm{I}}, \{t\}) \subseteq \mathit{Between}(S_{\mathrm{I}}, \mathit{dom}(\sigma)) \subseteq \mathit{dom}(\sigma)$. Because $s \in S_{\rightarrow}$ and $s \xrightarrow{\ell} t \in T$, we have $s \in \mathit{Between}(S_{\mathrm{I}}, \{t\})$ and hence $s \in \mathit{dom}(\sigma)$. Together with $\mathbf{CONS_{L+T}}$, we have that $\sigma(s) \xrightarrow{\lambda(\ell)} \sigma(t) \in T'$. Hence $h^{*}_{\Theta'}(\sigma(s)) \leq c'(\lambda(\ell)) + h^{*}_{\Theta'}(\sigma(t))$ because the perfect heuristic is consistent. We further have $c'(\lambda(\ell)) \leq c(\lambda)$ due to $\mathbf{CONS_C}$. Put together, we get $h^{\tau}(s) = h^{*}_{\Theta'}(\sigma(s)) \leq c(\ell) + h^{*}_{\Theta'}(\sigma(t)) = c(\ell) + h^{\tau}(t)$ as desired.

5. Follows from 1. and 3. because goal-awareness and consistency implies admissibility.

6. Follows from 2. and 4. because forward-goal-awareness and forward-consistency implies forward-admissibility. $\square$

Theorem 23. *Let τ be a transformation of a transition system Θ. The heuristic h^τ for Θ induced by τ is*

1. *perfect if τ satisfies* $\mathbf{CONS_{L+C+T+G}} + \mathbf{KEEP_G} + \mathbf{CLOS}_{\mathit{pred}} + \mathbf{REF}$ *and*

2. *forward-perfect if τ satisfies* $\mathbf{CONS_{L+C+T+G}} + \mathbf{KEEP}^{\rightarrow}_{\mathbf{G}} + \mathbf{CLOS}^{\rightarrow}_{\mathit{pred}} + \mathbf{REF}$.

Proof. Let S be the states of Θ. From Theorem 5.1 we get $h^*(s) \leq h^\tau(s)$ for all states $s \in S$ because τ satisfies **REF**. Part 1. follows because $h^\tau(s) \leq h^*(s)$ for all $s \in S$ (admissibility; Theorem 22.5). Part 2. follows analogously with Theorem 22.6. $\square$

Theorem 24. *Let F be a factored transition system, let $\Theta_k \in F$, and let S^k be the states of Θ_k. The prune transformation τ_{F} of F for K and Θ_k has the following properties depending on K:*

1. *for all K: τ_{F} satisfies* $\mathbf{CONS_{L+C+T+I+G}} + \mathbf{IND} + \mathbf{REF} + \mathbf{LOC}_{\geq}$.

2. *for $K = S^k_{\leftarrow}$: τ_{F} additionally satisfies* $\mathbf{KEEP_G} + \mathbf{CLOS}_{\mathit{pred}} + \mathbf{LOC}_{=}$.

3. *for $K = S^k_{\rightarrow}$ or $K = S^k_{\leftrightarrow}$: τ_{F} additionally satisfies* $\mathbf{KEEP}^{\rightarrow}_{\mathbf{G}} + \mathbf{CLOS}^{\rightarrow}_{\mathit{pred}}$.

Proof. Let $F = \langle \Theta_1, \dots, \Theta_n \rangle$, $F' = \langle \Theta'_1, \dots, \Theta'_n \rangle$ and $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$. Let $\Theta = \bigotimes F$ and $\Theta' = \bigotimes F'$, and let τ be the transformation of Θ into Θ' induced by τ_{F}. Let S be the states of Θ, and let S' be the states of Θ'. Note that $S' \subseteq S$ and that $s \in S$ belongs to S' iff $s[\Theta_k] \in K$ (Definition 33). Also from Definition 33, we get that $\tau = \langle \Theta', \sigma, \lambda \rangle$ where $\Theta' = \Theta^{S'}$ (Definition 32), $\sigma = [\![\Sigma]\!]$ is the identity function on S restricted to S', and λ is the identity function.

Since $\lambda = \mathrm{id}$, we simplify the notation by dropping the use of λ and λ^{-1} throughout. We now show the three parts of the theorem.

1. All properties in $\mathbf{CONS_{L+C+T+I+G}} + \mathbf{IND} + \mathbf{REF}$ are trivial to verify because τ behaves like the identity transformation except for the fact that it discards some states.

   For $\mathbf{LOC}_{\geq}$, we need to show $h^{\mathrm{loc},\tau_{\mathrm{F}}}_F(s) \geq h^{\mathrm{mf}}_F(s)$ for all states $s \in \mathcal{A}(F)$. We have:

$$\begin{aligned} h^{\mathrm{loc},\tau_{\mathrm{F}}}_F(s) &= h^{\mathrm{mf}}_{F'}(\sigma(s)) && \text{(Definition 22)} \\ &= \max_{1 \leq i \leq n} h^*_{\Theta'_i}(\sigma(s)[\Theta'_i]) && \text{(Definition 22)} \\ &\geq \max_{1 \leq i \leq n} h^*_{\Theta_i}(s[\Theta_i]) && \text{(*)} \\ &= h^{\mathrm{mf}}_F(s) && \text{(Definition 22)} \end{aligned}$$

   where we show (*) by proving $h^*_{\Theta'_i}(\sigma(s)[\Theta'_i]) \geq h^*_{\Theta_i}(s[\Theta_i])$ for all $1 \leq i \leq n$. For $i \neq k$, this is an equality because, for all components other than k, Σ is the identity mapping and $\Theta'_i = \Theta_i$.

   For $i = k$, there are two cases. If $s[\Theta_k] \notin K$, the local state $s[\Theta_k]$ is pruned by τ_{F}. Then $\sigma(s)$ is undefined and we get $h^*_{\Theta'_k}(\sigma(s)[\Theta'_k]) = \infty$ by definition, so that the inequality holds trivially. If $s[\Theta_k] \in K$, we get $h^*_{\Theta'_k}(\sigma(s)[\Theta'_k]) \overset{(**)}{=} h^*_{\Theta_k^K}(s[\Theta_k]) \overset{(***)}{\geq} h^*_{\Theta_k}(s[\Theta_k])$, where the equality (**) holds because the transformed transition system Θ'_k is Θ_k^K (the original transition system restricted to the states in K) and Σ is the identity on states that are not pruned. Inequality (***) holds because shortest paths in Θ_k^K can only be longer than in Θ_k because the transition systems are identical except that Θ_k^K misses some states and incident transitions.

2. **KEEP$_\mathbf{G}$** We have to show $S_\mathrm{G} \subseteq dom(\sigma)$, where S_G are the goal states of Θ. Because the other factors are unaffected by the transformation, it suffices to consider Θ_k. All goal states of Θ_k are in the preserved set $K = S^k_{\leftarrow}$ because every goal state is trivially backward-reachable from some goal state (itself).

CLOS$_{\textit{pred}}$ We have to show *Between*$(S, dom(\sigma)) \subseteq dom(\sigma)$. Consider a state $s \in$ *Between*$(S, dom(\sigma))$. From the definition of *Between*, there exists a path from s to some state $s' \in dom(\sigma)$. We must show $s \in dom(\sigma)$, which holds iff $s[\Theta_k] \in K = S^k_{\leftarrow}$. From $s' \in dom(\sigma)$, we get $s'[\Theta_k] \in S^k_{\leftarrow}$, so that $s'[\Theta_k]$ is backward-reachable in Θ_k. Because s has a path to s', $s[\Theta_k]$ has a path to $s'[\Theta_k]$, so $s[\Theta_k]$ is backward-reachable in Θ_k, showing $s[\Theta_k] \in S^k_{\leftarrow}$ and hence $s \in dom(\sigma)$ as desired.

LOC$_=$ We need to show $h_F^{\mathrm{loc},\tau_\mathrm{F}}(s) = h_F^{\mathrm{mf}}(s)$ for all states $s \in \mathcal{A}(F)$. We follow the same arguments as in the proof of **LOC$_\geq$** in part 1., except that we must now show $h^*_{\Theta'_k}(\sigma(s)[\Theta'_k]) = h^*_{\Theta_k}(s[\Theta_k])$.

If $s[\Theta_k] \notin K$, then $h^*_{\Theta'_k}(\sigma(s)[\Theta'_k]) = \infty$ by definition. Moreover, by definition of K, $s[\Theta_k]$ is not backward-reachable in Θ_k, and therefore $h^*_{\Theta_k}(s[\Theta_k]) = \infty$, so the equality holds.

If $s[\Theta_k] \in K$, we get $h^*_{\Theta'_k}(\sigma(s)[\Theta'_k]) \overset{(**)}{=} h^*_{\Theta_k^K}(s[\Theta_k]) \overset{(***)}{=} h^*_{\Theta_k}(s[\Theta_k])$, for the same reasons as in part 1., with the difference that (***) is now an equality: every path to the goal in Θ_k also exists in Θ_k^K because the only pruned states are ones from which the goal cannot be reached.

3. First, we consider $K = S^k_{\rightarrow}$:

KEEP$^{\rightarrow}_\mathbf{G}$ We have to show *Between*$(S_\mathrm{I}, S_\mathrm{G}) \cap S_\mathrm{G} \subseteq dom(\sigma)$, where S_I are the initial states and S_G are the goal states of Θ. Consider a goal state s of Θ such that $s \in$ *Between*$(S_\mathrm{I}, S_\mathrm{G})$. Then there exists a path from some initial state s_0 via s to some goal state $s_\star$. This implies that Θ_k has a path from $s_0[\Theta_k]$ to $s[\Theta_k]$, where $s_0[\Theta_k]$ is an initial state of Θ_k. It follows that $s[\Theta_k] \in K = S^k_{\rightarrow}$ and therefore $s \in dom(\sigma)$.

CLOS$^{\rightarrow}_{\textit{pred}}$ We have to show *Between*$(S_\mathrm{I}, dom(\sigma)) \subseteq dom(\sigma)$, where S_I are the initial states of Θ. Consider a state $s \in$ *Between*$(S_\mathrm{I}, dom(\sigma))$. From the definition of *Between*, there exists a path from some initial state s_0 via s to some state $s' \in dom(\sigma)$. This implies that Θ_k has a path from $s_0[\Theta_k]$ to $s[\Theta_k]$, where $s_0[\Theta_k]$ is an initial state of Θ_k. It follows that $s[\Theta_k] \in K = S^k_{\rightarrow}$ and therefore $s \in dom(\sigma)$.

Finally, we consider $K = S^k_{\leftrightarrow}$. For any transition system with states S, the alive states $S_{\leftrightarrow}$ are the states that are forward-reachable after restricting to the backward-reachable states (or vice versa), i.e., $S_{\leftrightarrow} = (S_{\leftarrow})_{\rightarrow}$. Therefore, pruning to $S^k_{\leftrightarrow}$ can be expressed as the composition $\tau'' = \tau' \circ \tau$, where τ restricts Θ_k to its backward-reachable states and τ' restricts to the forward-reachable states. From part 2. of this proof, τ has the properties $\mathbf{KEEP_G} + \mathbf{CLOS}_{pred}$, which imply the weaker properties $\mathbf{KEEP}^{\rightarrow}_{\mathbf{G}} + \mathbf{CLOS}^{\rightarrow}_{pred}$ as discussed earlier. From part 1. of this proof, it also has the properties $\mathbf{CONS_{L+T+I+G}}$. Moreover, τ' has the properties $\mathbf{KEEP}^{\rightarrow}_{\mathbf{G}} + \mathbf{CLOS}^{\rightarrow}_{pred}$ as just shown. From Theorem 21, it follows that the composition τ'' of τ and τ' has the properties $\mathbf{KEEP}^{\rightarrow}_{\mathbf{G}} + \mathbf{CLOS}^{\rightarrow}_{pred}$. □

References

Babaki, B., Pesant, G., & Quimper, C. (2020). Solving classical AI planning problems using planning-independent CP modeling and search. In Harabor, D., & Vallati, M. (Eds.), *Proceedings of the 13th Annual Symposium on Combinatorial Search (SoCS 2020)*, pp. 2–10. AAAI Press.

Bacchus, F., & Yang, Q. (1994). Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, *71*(1), 43–100.

Bäckström, C., & Jonsson, P. (2012a). Abstracting abstraction in search with applications in planning. In Brewka, G., Eiter, T., & McIlraith, S. A. (Eds.), *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, pp. 446–456. AAAI Press.

Bäckström, C., & Jonsson, P. (2012b). Algorithms and limits for compact plan representations. *Journal of Artificial Intelligence Research*, *44*, 141–177.

Bäckström, C., & Jonsson, P. (2013). Bridging the gap between refinement and heuristics in abstraction. In Rossi, F. (Ed.), *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pp. 2261–2267. AAAI Press.

Bäckström, C., Jonsson, P., & Ståhlberg, S. (2013). Fast detection of unsolvable planning instances using local consistency. In Helmert, M., & Röger, G. (Eds.), *Proceedings of the Sixth Annual Symposium on Combinatorial Search (SoCS 2013)*, pp. 29–37. AAAI Press.

Bäckström, C., & Nebel, B. (1995). Complexity results for SAS$^+$ planning. *Computational Intelligence*, *11*(4), 625–655.

Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G. D., Macii, E., Pardo, A., & Somenzi, F. (1997). Algebraic decision diagrams and their applications. *Formal Methods in System Design*, *10*(2–3), 171–206.

Beck, J. C., Magazzeni, D., Röger, G., & van Hoeve, W. (2018). Planning and operations research (Dagstuhl seminar 18071). *Dagstuhl Reports*, *8*(2), 26–63.

Bonet, B., & van den Briel, M. (2014). Flow-based heuristics for optimal planning: Landmarks and merges. In Chien, S., Fern, A., Ruml, W., & Do, M. (Eds.), *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, pp. 47–55. AAAI Press.

Bryant, R. E. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, *35*(8), 677–691.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, *69*(1–2), 165–204.

Clarke, E. M., Emerson, E. A., & Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, *8*(2), 244–263.

Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model Checking*. The MIT Press.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In Harrison, M. A., Banerji, R. B., & Ullman, J. D. (Eds.), *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing (STOC 1971)*, pp. 151–158. ACM.

Culberson, J. C., & Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, *14*(3), 318–334.

Darwiche, A. (2001). Decomposable negation normal form. *Journal of the ACM*, *48*(4), 608–647.

Darwiche, A. (2011). SDD: A new canonical representation of propositional knowledge bases. In Walsh, T. (Ed.), *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pp. 819–826. AAAI Press.

Darwiche, A., & Marquis, P. (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research*, *17*, 229–264.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

Dechter, R., & Pearl, J. (1985). Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, *32*(3), 505–536.

Demassey, S., Pesant, G., & Rousseau, L. (2006). A cost-regular based hybrid column generation approach. *Constraints*, *11*(4), 315–333.

Domshlak, C., Katz, M., & Lefler, S. (2010). When abstractions met landmarks. In Brafman, R., Geffner, H., Hoffmann, J., & Kautz, H. (Eds.), *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, pp. 50–56. AAAI Press.

Domshlak, C., Katz, M., & Lefler, S. (2012). Landmark-enhanced abstraction heuristics. *Artificial Intelligence*, *189*, 48–68.

Domshlak, C., Katz, M., & Shleyfman, A. (2015). Symmetry breaking in deterministic planning as forward search: Orbit space search algorithm. Tech. rep. IS/IE-2015-03, Technion.

Dräger, K., Finkbeiner, B., & Podelski, A. (2006). Directed model checking with distance-preserving abstractions. In Valmari, A. (Ed.), *Proceedings of the 13th International SPIN Workshop (SPIN 2006)*, Vol. 3925 of *Lecture Notes in Computer Science*, pp. 19–34. Springer-Verlag.

Dräger, K., Finkbeiner, B., & Podelski, A. (2009). Directed model checking with distance-preserving abstractions. *International Journal on Software Tools for Technology Transfer*, *11*(1), 27–37.

Edelkamp, S. (2001). Planning with pattern databases. In Cesta, A., & Borrajo, D. (Eds.), *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, pp. 84–90. AAAI Press.

Edelkamp, S., & Helmert, M. (2001). The model checking integrated planning system (MIPS). *AI Magazine*, *22*(3), 67–71.

Edelkamp, S., Kissmann, P., & Torralba, Á. (2012). Symbolic A* search with pattern databases and the merge-and-shrink abstraction. In De Raedt, L., Bessiere, C., Dubois, D., Doherty, P., Frasconi, P., Heintz, F., & Lucas, P. (Eds.), *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pp. 306–311. IOS Press.

Edelkamp, S., & Reffel, F. (1998). OBDDs in heuristic search. In Herzog, O., & Günter, A. (Eds.), *Proceedings of the 22nd Annual German Conference on Artificial Intelligence (KI 1998)*, Vol. 1504 of *Lecture Notes in Computer Science*, pp. 81–92. Springer-Verlag.

Eriksson, S., Röger, G., & Helmert, M. (2017). Unsolvability certificates for classical planning. In Barbulescu, L., Frank, J., Mausam, & Smith, S. F. (Eds.), *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, pp. 88–97. AAAI Press.

Eriksson, S., Röger, G., & Helmert, M. (2018). A proof system for unsolvable planning tasks. In de Weerdt, M., Koenig, S., Röger, G., & Spaan, M. (Eds.), *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*, pp. 65–73. AAAI Press.

Fan, G., Holte, R., & Müller, M. (2018). MS-lite: A lightweight, complementary merge-and-shrink method. In de Weerdt, M., Koenig, S., Röger, G., & Spaan, M. (Eds.), *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*, pp. 74–82. AAAI Press.

Fan, G., Müller, M., & Holte, R. (2014). Non-linear merging strategies for merge-and-shrink based on variable interactions. In Edelkamp, S., & Barták, R. (Eds.), *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014)*, pp. 53–61. AAAI Press.

Fan, G., Müller, M., & Holte, R. (2017). Additive merge-and-shrink heuristics for diverse action costs. In Sierra, C. (Ed.), *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, pp. 4287–4293. IJCAI.

Fickert, M., Hoffmann, J., & Steinmetz, M. (2016). Combining the delete relaxation with critical-path heuristics: A direct characterization. *Journal of Artificial Intelligence Research*, *56*, 269–327.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Haslum, P. (2007). Reducing accidental complexity in planning problems. In Veloso, M. M. (Ed.), *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pp. 1898–1903.

Haslum, P. (2009). $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m. In Gerevini, A., Howe, A., Cesta, A., & Refanidis, I. (Eds.), *Proceedings of the*

Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009), pp. 354–357. AAAI Press.

Haslum, P. (2012). Incremental lower bounds for additive cost planning problems. In McCluskey, L., Williams, B., Silva, J. R., & Bonet, B. (Eds.), *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, pp. 74–82. AAAI Press.

Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In Chien, S., Kambhampati, S., & Knoblock, C. A. (Eds.), *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)*, pp. 140–149. AAAI Press.

Helmert, M. (2004). A planning heuristic based on causal graph analysis. In Zilberstein, S., Koehler, J., & Koenig, S. (Eds.), *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pp. 161–170. AAAI Press.

Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, *26*, 191–246.

Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway?. In Gerevini, A., Howe, A., Cesta, A., & Refanidis, I. (Eds.), *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pp. 162–169. AAAI Press.

Helmert, M., Haslum, P., & Hoffmann, J. (2007). Flexible abstraction heuristics for optimal sequential planning. In Boddy, M., Fox, M., & Thiébaux, S. (Eds.), *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, pp. 176–183. AAAI Press.

Helmert, M., Haslum, P., & Hoffmann, J. (2008). Explicit-state abstraction: A new method for generating heuristic functions. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, pp. 1547–1550. AAAI Press.

Helmert, M., Haslum, P., Hoffmann, J., & Nissim, R. (2014). Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM*, *61*(3), 16:1–63.

Helmert, M., Röger, G., & Sievers, S. (2015). On the expressive power of non-linear merge-and-shrink representations. In Brafman, R., Domshlak, C., Haslum, P., & Zilberstein, S. (Eds.), *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, pp. 106–114. AAAI Press.

Hoffmann, J., Kissmann, P., & Torralba, Á. (2014). "Distance"? Who cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In Schaub, T., Friedrich, G., & O'Sullivan, B. (Eds.), *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, pp. 441–446. IOS Press.

Hoffmann, J., & Kupferschmid, S. (2005). A covering problem for hypercubes. In Kaelbling, L. P., & Saffiotti, A. (Eds.), *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pp. 1523–1524. Professional Book Center.

Hoffmann, J., Porteous, J., & Sebastia, L. (2004). Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, *22*, 215–278.

Holte, R. C. (2010). Common misconceptions concerning heuristic search. In Felner, A., & Sturtevant, N. (Eds.), *Proceedings of the Third Annual Symposium on Combinatorial Search (SoCS 2010)*, pp. 46–51. AAAI Press.

Horton, R. E. (1945). Erosional development of streams and their drainage basins; hydrophysical approach to quantitative morphology. *Bulletin of the Geological Society of America*, *56*, 275–370.

Jonsson, P., & Bäckström, C. (1998). State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence*, *100*(1–2), 125–176.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E., & Thatcher, J. W. (Eds.), *Complexity of Computer Computations*, pp. 85–103. Plenum Press.

Katz, M., & Domshlak, C. (2010). Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, *174*(12–13), 767–798.

Katz, M., Hoffmann, J., & Helmert, M. (2012). How to relax a bisimulation?. In McCluskey, L., Williams, B., Silva, J. R., & Bonet, B. (Eds.), *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, pp. 101–109. AAAI Press.

Keyder, E., Hoffmann, J., & Haslum, P. (2012). Semi-relaxed plan heuristics. In McCluskey, L., Williams, B., Silva, J. R., & Bonet, B. (Eds.), *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, pp. 128–136. AAAI Press.

Kissmann, P. (2012). *Symbolic Search in Planning and General Game Playing*. Ph.D. thesis, University of Bremen.

Kissmann, P., Edelkamp, S., & Hoffmann, J. (2014). Gamer and Dynamic-Gamer – symbolic search at IPC 2014. In *Eighth International Planning Competition (IPC-8): planner abstracts*, pp. 77–84.

Knoblock, C. A. (1994). Automatically generating abstractions for planning. *Artificial Intelligence*, *68*(2), 243–302.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, *27*(1), 97–109.

Kozen, D. (1977). Lower bounds for natural proof systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pp. 254–266.

Milner, R. (1980). *A Calculus of Communicating Systems*, Vol. 92 of *Lecture Notes in Computer Science*. Springer-Verlag.

Moore, E. F. (1956). Gedanken-experiments on sequential machines. *Automata Studies*, *34*, 129–153.

Muise, C., & Lipovetzky, N. (Eds.). (2016). *Unsolvability International Planning Competition: Planner Abstracts*.

Nissim, R., Hoffmann, J., & Helmert, M. (2011). Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Walsh, T. (Ed.), *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pp. 1983–1990. AAAI Press.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pesant, G. (2004). A regular language membership constraint for finite sequences of variables. In Wallace, M. (Ed.), *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)*, Vol. 3258 of *Lecture Notes in Computer Science*, pp. 482–495. Springer-Verlag.

Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pp. 46–57.

Pommerening, F., Helmert, M., Röger, G., & Seipp, J. (2015). From non-negative to general operator cost partitioning. In Bonet, B., & Koenig, S. (Eds.), *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, pp. 3335–3341. AAAI Press.

Pommerening, F., Röger, G., Helmert, M., & Bonet, B. (2014). LP-based heuristics for cost-optimal planning. In Chien, S., Fern, A., Ruml, W., & Do, M. (Eds.), *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, pp. 226–234. AAAI Press.

Röger, G., Pommerening, F., & Helmert, M. (2014). Optimal planning in the presence of conditional effects: Extending LM-Cut with context splitting. In Schaub, T., Friedrich, G., & O'Sullivan, B. (Eds.), *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, pp. 765–770. IOS Press.

Seipp, J., & Helmert, M. (2011). Fluent merging for classical planning problems. In *ICAPS 2011 Workshop on Knowledge Engineering for Planning and Scheduling*, pp. 47–53.

Seipp, J., & Helmert, M. (2014). Diverse and additive Cartesian abstraction heuristics. In Chien, S., Fern, A., Ruml, W., & Do, M. (Eds.), *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, pp. 289–297. AAAI Press.

Seipp, J., & Helmert, M. (2018). Counterexample-guided Cartesian abstraction refinement for classical planning. *Journal of Artificial Intelligence Research*, *62*, 535–577.

Seipp, J., Keller, T., & Helmert, M. (2020). Saturated cost partitioning for optimal classical planning. *Journal of Artificial Intelligence Research*, *67*, 129–167.

Sievers, S. (2017). *Merge-and-shrink Abstractions for Classical Planning: Theory, Strategies, and Implementation*. Ph.D. thesis, University of Basel.

Sievers, S. (2018). Merge-and-shrink heuristics for classical planning: Efficient implementation and partial abstractions. In Bulitko, V., & Storandt, S. (Eds.), *Proceedings of the 11th Annual Symposium on Combinatorial Search (SoCS 2018)*, pp. 90–98. AAAI Press.

Sievers, S., Ortlieb, M., & Helmert, M. (2012). Efficient implementation of pattern database heuristics for classical planning. In Borrajo, D., Felner, A., Korf, R., Likhachev, M., Linares López, C., Ruml, W., & Sturtevant, N. (Eds.), *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SoCS 2012)*, pp. 105–111. AAAI Press.

Sievers, S., Pommerening, F., Keller, T., & Helmert, M. (2020). Cost-partitioned merge-and-shrink heuristics for optimal classical planning. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*, pp. 4152–4160. IJCAI.

Sievers, S., Wehrle, M., & Helmert, M. (2014). Generalized label reduction for merge-and-shrink heuristics. In Brodley, C. E., & Stone, P. (Eds.), *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, pp. 2358–2366. AAAI Press.

Sievers, S., Wehrle, M., & Helmert, M. (2016). An analysis of merge strategies for merge-and-shrink heuristics. In Coles, A., Coles, A., Edelkamp, S., Magazzeni, D., & Sanner, S. (Eds.), *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, pp. 294–298. AAAI Press.

Sievers, S., Wehrle, M., Helmert, M., Shleyfman, A., & Katz, M. (2015). Factored symmetries for merge-and-shrink abstractions. In Bonet, B., & Koenig, S. (Eds.), *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, pp. 3378–3385. AAAI Press.

Torralba, Á. (2015). *Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning*. Ph.D. thesis, Universidad Carlos III de Madrid.

Torralba, Á., Alcázar, V., Borrajo, D., Kissmann, P., & Edelkamp, S. (2014). SymBA*: A symbolic bidirectional A* planner. In *Eighth International Planning Competition (IPC-8): planner abstracts*, pp. 105–109.

Torralba, Á., & Hoffmann, J. (2015). Simulation-based admissible dominance pruning. In Yang, Q., & Wooldridge, M. (Eds.), *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pp. 1689–1695. AAAI Press.

Torralba, Á., & Kissmann, P. (2015). Focusing on what really matters: Irrelevance pruning in merge-and-shrink. In Lelis, L., & Stern, R. (Eds.), *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SoCS 2015)*, pp. 122–130. AAAI Press.

Torralba, Á., Linares López, C., & Borrajo, D. (2013). Symbolic merge-and-shrink for cost-optimal planning. In Rossi, F. (Ed.), *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pp. 2394–2400. AAAI Press.

Torralba, Á., Linares López, C., & Borrajo, D. (2016). Abstraction heuristics for symbolic bidirectional search. In Kambhampati, S. (Ed.), *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pp. 3272–3278. AAAI Press.

Torralba, Á., Linares López, C., & Borrajo, D. (2018). Symbolic perimeter abstraction heuristics for cost-optimal planning. *Artificial Intelligence*, *259*, 1–31.

Torralba, Á., & Sievers, S. (2019). Merge-and-shrink task reformulation for classical planning. In Kraus, S. (Ed.), *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pp. 5644–5652. IJCAI.

Tožička, J., Jakubův, J., Svatoš, M., & Komenda, A. (2016). Recursive polynomial reductions for classical planning. In Coles, A., Coles, A., Edelkamp, S., Magazzeni, D., & Sanner, S. (Eds.), *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, pp. 317–325. AAAI Press.

van den Briel, M., Benton, J., Kambhampati, S., & Vossen, T. (2007a). An LP-based heuristic for optimal planning. In Bessiere, C. (Ed.), *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP 2007)*, Vol. 4741 of *Lecture Notes in Computer Science*, pp. 651–665. Springer-Verlag.

van den Briel, M., Kambhampati, S., & Vossen, T. (2007b). Fluent merging: A general technique to improve reachability heuristics and factored planning. In *ICAPS 2007 Workshop on Heuristics for Domain-Independent Planning: Progress, Ideas, Limitations, Challenges*.

van den Briel, M., Vossen, T., & Kambhampati, S. (2005). Reviving integer programming approaches for AI planning: A branch-and-cut framework. In Biundo, S., Myers, K., & Rajan, K. (Eds.), *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, pp. 310–319. AAAI Press.

Wehrle, M., & Helmert, M. (2009). The causal graph revisited for directed model checking. In *Proceedings of the 16th International Static Analysis Symposium (SAS 2009)*, pp. 86–101.

Wehrle, M., Sievers, S., & Helmert, M. (2016). Graph-based factorization of classical planning problems. In Kambhampati, S. (Ed.), *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pp. 3286–3292. AAAI Press.

Yang, F., Culberson, J., Holte, R., Zahavi, U., & Felner, A. (2008). A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research*, *32*, 631–662.

Zanarini, A., Pesant, G., & Milano, M. (2006). Planning with soft regular constraints. In *ICAPS 2006 Workshop on Preferences and Soft Constraints in Planning*, pp. 73–78.