Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces

MALTE HELMERT, University of Basel, Switzerland
PATRIK HASLUM, The Australian National University and NICTA, Australia
JÖRG HOFFMANN, Saarland University, Germany
RAZ NISSIM, Ben-Gurion University of the Negev, Israel

Many areas of computer science require answering questions about reachability in compactly described discrete transition systems. Answering such questions effectively requires techniques to be able to do so without building the entire system. In particular, heuristic search uses lower-bounding ("admissible") heuristic functions to prune parts of the system known to not contain an optimal solution. A prominent technique for deriving such bounds is to consider abstract transition systems that aggregate groups of states into one. The key question is how to design and represent such abstractions. The most successful answer to this question are pattern databases, which aggregate states if and only if they agree on a subset of the state variables. *Merge-and-shrink abstraction* is a new paradigm that, as we show, allows to compactly represent a more general class of abstractions, strictly dominating pattern databases in theory. We identify the maximal class of transition systems, which we call *factored transition systems*, to which merge-and-shrink applies naturally, and we show that the well-known notion of bisimilarity can be adapted to this framework in a way that still guarantees perfect heuristic functions, while potentially reducing abstraction size exponentially. Applying these ideas to planning, one of the foundational subareas of artificial intelligence, we show that in some benchmarks this size reduction leads to the computation of perfect heuristic functions in polynomial time and that more approximate merge-and-shrink strategies yield heuristic functions competitive with the state of the art.

Categories and Subject Descriptors: I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Graph and tree search strategies*; *heuristic methods*; *plan execution, formation, and generation*

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Abstraction heuristics, heuristic search, AI planning

ACM Reference Format:
Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. J. ACM 61, 3, Article 16 (May 2014), 63 pages.
DOI: http://dx.doi.org/10.1145/2559951

M. Helmert's work was partly supported by the Swiss National Science Foundation (SNSF) as part of the project "Abstraction Heuristics for Planning and Combinatorial Search" (AHPACS). P. Haslum's work was partly supported by ARC project DP0985532 "Exploiting Structure in AI Planning". NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. J. Hoffmann's work was partly supported by the French National Research Agency (ANR), as part of the BARQ project (ANR-10-CEXC-003-01). R. Nissim's work was performed in part while visiting INRIA, Nancy, France, and was partly supported by INRIA, ISF Grant 1101/07, the Paul Ivanier Center for Robotics Research and Production Management, and the Lynn and William Frankel Center for Computer Science.
Authors' addresses: M. Helmert, University of Basel, Bernoullistrasse 16, 4056 Basel, Switzerland; email: malte.helmert@unibas.ch; P. Haslum, Australian National University, Building 108, Canberra ACT 0200, Australia; email: patrik.haslum@anu.edu.au; J. Hoffmann, Saarland University, Campus E1 1, 66123 Saarbruecken, Germany; email: hoffmann@cs.uni-saarland.de; R. Nissim, Ben-Gurion University, P.O.B. 653, Beer-Sheva 84105, Israel; email: raznis@cs.bgu.ac.il.

0004-5411/2014/05-ART16
DOI: http://dx.doi.org/10.1145/2559951

1. INTRODUCTION

Exploration of large discrete state transition systems to answer questions about reachability arises in many areas of computer science. For example, this is the case in artificial intelligence (AI) subareas such as planning [Ghallab et al. 2004] and diagnosis [Lamperti and Zanella 2003], in model checking [Clarke et al. 2000], and in the multiple sequence alignment problem in bioinformatics [Korf and Zhang 2000]. Systems of interest are defined in some representation, such as state variables and operators, networks of synchronized automata, or bounded Petri nets, which is exponentially more compact than the transition system it represents. This exponential difference between the size of a compact system representation and the number of represented states is known as the state explosion problem [Valmari 1989]. State explosion makes building an explicit representation of the complete transition system infeasible and pushes the complexity of the reachability problem from being solvable in polynomial time (in the size of the system) to hard for polynomial space (in the size of the compact representation). Search algorithms that explore such implicitly defined transition systems use information extracted from the compact representation to reduce the part of the system that must be built to find the desired answer.

We consider the extraction of a particular type of information, namely *admissible heuristics*, using a particular class of methods, namely *abstractions*. Heuristic functions, in general, estimate the cost of reaching an "end state" (goal state, deadlock state, etc.) from a given state and are used to guide informed search algorithms. Admissible heuristics in particular are lower bound functions: a heuristic is admissible iff it never overestimates the true cost. Admissible heuristics are an important tool when searching for optimal solutions (i.e., minimum cost plans, most plausible diagnoses, shortest error traces, etc.), because certain heuristic search algorithms, like A^* [Hart et al. 1968], guarantee that the solution returned is optimal under the condition that the heuristic function used is admissible. The efficiency of optimal heuristic search depends on the accuracy of the heuristic function: the closer its estimate is to the true optimal cost, the less search is required to find and prove the optimality of a solution [Dechter and Pearl 1985, but note Holte 2010]. Therefore, the question is how to compute good lower bounds.

Abstractions have a long history of use in the analysis of discrete state transition systems. An abstraction is a mapping to an abstract system that makes fewer distinctions between states—*aggregating* subsets of states into one—while preserving all transitions between states. With fewer states, analysis of the abstract system (based on an appropriate representation of that system) is more tractable. Since transitions are preserved, the possible behaviors—observable transition label sequences—exhibited by the original system are overapproximated. This makes reachability in the abstract system a necessary condition for reachability in the original system. For example, if reaching a deadlock state is impossible in the abstraction, it is also impossible in the original system (e.g., Cousot and Cousot [1977]). If the possible behaviors of the abstract system are exactly the same as those of the original, the systems are said to be *bisimilar* (e.g., Milner [1980]). Intuitively, the abstraction is then "perfect": it incurs no loss of information.

Any abstraction yields an admissible heuristic, and this is how we use abstractions in this work. Note the difference to verification:

(I) Any heuristic may yield useful search guidance, so the abstraction does not need to be perfect or preserve any reachability property exactly.
(II) We are interested not in preserving observable transition label sequences, but merely in preserving their cost.

Due to (I), we have full freedom in the tradeoff between the accuracy and computational cost of a heuristic. Due to (II), abstractions only need to account for the cost associated with transitions, not for their labels; this plays a key role in the techniques we propose.

The most widely known and used abstraction heuristics are *pattern databases* (PDBs), which have been shown to be effective for many search problems (e.g., Korf [1997], Culberson and Schaeffer [1998], Korf and Zhang [2000], and Edelkamp [2001]). They are based on *projection*, that is, they aggregate states iff they agree on a subset of state variables (the pattern). PDBs can be stored compactly with symbolic data structures [Edelkamp 2002; Ball and Holte 2008], they can be combined additively without losing admissibility under certain conditions [Felner et al. 2004; Holte et al. 2006], and suitable patterns can be computed automatically [Haslum et al. 2007]. The Achilles' heel of PDBs is that the patterns must be small because of the state explosion problem, and PDBs contain no information about the variables outside the pattern. Hence, there are many transition systems for which a very accurate, even perfect, lower bound heuristic function can be obtained from an abstraction of reasonable size, yet this abstraction is not a projection and hence outside the scope of PDBs. We introduce a more general class of abstractions—*merge-and-shrink abstractions*—that admits perfect heuristic functions for a significantly larger class of transition systems.

Merge-and-shrink abstraction originated in model checking of automata networks. Sabnani et al. [1989] proposed incrementally constructing the smallest bisimilar abstraction of a system represented by partially synchronized automata, by interleaving composition of component automata with reduction of the partial composites, preserving bisimilarity at each reduction step. It is often not feasible to build even this smallest bisimilar abstraction. This inspired Dräger et al. [2006, 2009] to design an approximate variant of the Sabnani et al. algorithm, thus inventing merge-and-shrink heuristics, which interleave the composition of automata (merging step) with (potentially non-bisimilar) abstraction of the composites (shrinking step). Such a heuristic requires an *abstraction strategy*, that is, a rule for choosing which automata to merge and how to shrink them. Dräger et al.'s abstraction strategy is to drop the bisimilarity requirement when a constant limit on abstraction size is reached. They then aggregate more states, in a potentially lossy way. The resulting heuristics were used in directed model checking, that is, for guiding explicit-state search towards error states in computer-aided verification.

The main contributions of our own work are:

(1) the extension of merge-and-shrink to the more general framework of *factored transition systems*, capturing the minimal requirements needed for merge-and-shrink to work;
(2) the investigation of the expressive power of merge-and-shrink, showing in particular that merge-and-shrink strictly subsumes pattern databases: it can always simulate these efficiently, and may be exponentially more compact;
(3) the design of conservative label reduction techniques, which may exponentially reduce bisimulation size while still computing perfect heuristics (as opposed to perfect abstractions, cf. (II));[1]
(4) the application of merge-and-shrink to AI planning, designing new abstraction strategies and demonstrating their theoretical and practical merits.

[1]With a perfect heuristic, most reachability questions can be answered without search (we discuss this in Section 2.3), so this is a possible alternative to the use of perfect abstractions. We are not aware of work in computer-aided verification exploring this possibility.

Factored transition systems encompass not only Dräger et al.'s automata networks, but also state variable/operator-based descriptions that are often a more natural way of modeling, for example, planning or diagnosis problems. Also, in difference to Dräger et al., our framework allows transitions to be associated with arbitrary nonnegative costs. Merge-and-shrink can also be applied to non-factored systems. However, factored systems are the largest class in which one can always reconstruct the original transition system by the synchronized product of its components, thus enabling this process to construct perfect heuristics.

In addition to subsuming PDBs, we show that merge-and-shrink can efficiently simulate admissibly additive ensembles of heuristics, thus capturing such additivity intrinsically, and subsuming even additive ensembles of PDBs. Exploiting a relationship between merge-and-shrink and algebraic decision diagrams, we shed some light on an open question [Helmert and Domshlak 2009] regarding the relationship between merge-and-shrink and critical-path heuristics [Haslum and Geffner 2000], another family of admissible heuristics from AI planning.

Our conservative abstraction strategy guarantees to compute perfect heuristics, and does so in polynomial time in three of the six classical planning benchmarks that have polynomial-time optimal solution algorithms. To design approximate strategies, we introduce *greedy bisimilarity*, requiring bisimilarity relative to a particular subset of transitions only. The resulting techniques are competitive with the state of the art in optimal planning. Apart from our own experiments, this is illustrated by the outcome of the 2011 International Planning Competition,[2] where our tool won a 2nd prize and was part of the 1st prize winner (a portfolio of four planning algorithms, two of which were variants of merge-and-shrink).

Preliminary versions of some of our results were presented in two conference papers [Helmert et al. 2007; Nissim et al. 2011]. The present article is structured as follows. Section 2 introduces the basic concepts of factored transition systems, heuristics, and AI planning. Section 3 defines merge-and-shrink abstractions mathematically, Section 4 discusses their algorithmic aspects. Section 5 introduces conservative label reduction. Section 6 applies Sabnani et al.'s use of bisimilarity to our framework and combines it with label reduction. Section 7 gives our results regarding expressive power. Section 8 provides our evaluation in planning. Section 9 concludes. The main text contains proof sketches only; full proofs are in Appendix A.

2. BASIC CONCEPTS

In this section, we introduce the notion of factored transition systems, which is the basis for the derivation of merge-and-shrink heuristics, and discuss some of its properties. In addition, in Sections 2.2 and 2.3, we briefly recap some basic facts about AI planning and heuristic search.

2.1. Factored Transition Systems

A discrete state transition system is a labeled directed graph where nodes represent states of the system and arcs represent transitions between states. Each transition has a label, identifying the event, action, or equivalent that causes the state change. Labels are not unique: the same label may appear on many arcs in the graph. However, we usually assume that the system is deterministic in the sense that the labels of outgoing arcs from any single state are distinct. A reachability problem, relative to a transition system, asks for a directed path through the graph from a distinguished *initial state* to any state in a given set of *solution states*. In a planning problem, these are states where

[2]http://www.icaps-conference.org/index.php/Main/Competitions.

the planning goal is achieved; in a verification problem, they may be deadlock states, or states where a safety property is violated. Note that the solution to a reachability problem is the *path* from initial state to solution state, not the solution state itself. Since we are interested in problems that require optimal solutions, we associate with each transition label a fixed nonnegative *cost*, and seek a path of minimum total cost. (That costs are nonnegative and transition systems are finite ensures that the minimum is well defined.)

In any realistic situation, the state/transition graph is not presented explicitly, but defined indirectly through some compact representation. Typically, states are assignments of values, from some finite domain, to a set of *state variables*, and transitions are instances of *operators*, each of which affects only a subset of variables. Alternatively, if the system is composed of several smaller systems interacting in some way (e.g., by synchronizing on shared transition labels), each subsystem can be seen as a state variable. This implies that the size of the transition system can be, and indeed usually is, exponential in the size of the system representation, but also that the system is not an arbitrary graph. By imposing a further restriction on how transitions and the set of solution states are defined, we obtain a class of systems, which we call *factored*, that have a particular structure. The most important feature of a factored system is that it is isomorphic to the synchronized product of the projections onto each of its state variables (cf. Theorem 4.5).

Definition 2.1. A *labeled transition system*, or *transition system* for short, is a 5-tuple $\Theta = (S, L, T, s_0, S_\star)$ where S is a finite set of *states*, L is a finite set of *transition labels* each associated with a *label cost* $c(l) \in \mathbb{R}_0^+$, $T \subseteq S \times L \times S$ is a set of *labeled transitions*, $s_0 \in S$ is the *initial state*, and $S_\star \subseteq S$ is the set of *solution states*. A *path from s to s' in Θ* is a sequence of transitions $t_1, \ldots, t_n$ such that there exist states $s = s_0, \ldots, s_n = s'$ such that $(s_{i-1}, t_i, s_i) \in T$ for all $i = 1, \ldots, n$.

The *cost* of a path in Θ is the sum of the label costs along the transitions it takes. The *solution cost* $h^* : S \to \mathbb{R}_0^+ \cup \{\infty\}$ is the function which assigns to each state $s \in S$ the minimal cost of any path in Θ from s to any $s_\star \in S_\star$, or $h^*(s) = \infty$ if there is no such path. A path from s_0 to any $s_\star \in S_\star$ is a *solution* for Θ. A solution is *optimal* if its cost is equal to $h^*(s_0)$.

We will only be concerned with algorithms that find provably optimal solutions to reachability problems.

While the definition admits arbitrary nonnegative label costs, important special cases that we will mention are discrete (integer) costs, as well as *uniform* costs where $c(l) = c(l')$ for all $l, l' \in L$. Note that we allow zero-cost labels, a natural model of cases where one wishes to account only for some of the transitions in Θ.

Next, we define precisely the meaning of factoredness:

Definition 2.2. Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system. Let $\mathcal{V}$ be a finite set, where each $v \in \mathcal{V}$ has an associated finite domain $\mathcal{D}_v$. We say that Θ *has state variables* $\mathcal{V}$ if the set of states S is the set of functions s on $\mathcal{V}$ with $s(v) \in \mathcal{D}_v$ for all $v \in \mathcal{V}$.

Definition 2.3. Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system with state variables $\mathcal{V}$. For $V \subseteq \mathcal{V}$ and $s, t \in S$, the *recombination* $\rho_V(s, t)$ of s and t under V is the state defined by

$$\rho_V(s, t) := s|_V \cup t|_{\mathcal{V} \setminus V}.$$

(Throughout the paper, $f|_X$ denotes the restriction of a function f to set X.)

We say that Θ is *factored* if the following two conditions hold.

(i) For all $s_\star, t_\star \in S_\star$ and $V \subseteq \mathcal{V}$, $\rho_V(s_\star, t_\star) \in S_\star$.
(ii) For all $(s, l, s'), (t, l, t') \in T$ and $V \subseteq \mathcal{V}$, $(\rho_V(s, t), l, \rho_V(s', t')) \in T$.

Note that a system that has only one variable is trivially factored. Every system can be viewed as a system with one state variable (that variable being the state of the system), but, as mentioned, almost any system of interest genuinely has multiple variables. For example, in a (bounded) Petri net, the marking of each place constitutes a state variable. In a transition system defined by composition of automata (which are single-variable systems), the state of each component automaton can be viewed as a variable. Thus, assuming that systems have variables, according to Definition 2.2, is not very restrictive.

Assuming that a multivariable system is factored, however, does restrict the relation between transitions and transition labels. Intuitively, it means that the effect of a transition on any state variable is determined only by the value of this state variable and the transition label: two transitions with the same label cannot change a state variable to different values depending on the value of some other variable. Viewed another way, it means that each transition label can be associated with an enabling condition which is a conjunction of conditions on individual state variables, and that the same is true also of the condition that defines solution states. The following proposition shows that this is the case.

PROPOSITION 2.4. *Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system with state variables $\mathcal{V}$. Then, Θ is factored if and only if the following two conditions hold.*

(i) There exists a family of "goal domains" $(\mathcal{D}_v^{\mathrm{g}})_{v \in \mathcal{V}}$ where $\mathcal{D}_v^{\mathrm{g}} \subseteq \mathcal{D}_v$ for all $v \in \mathcal{V}$, such that for all $s \in S$, we have $s \in S_\star$ iff $s(v) \in \mathcal{D}_v^{\mathrm{g}}$ for all $v \in \mathcal{V}$.
(ii) For each label $l \in L$, there exists a family of "precondition/effect domains" $(\mathcal{D}_v^{\mathrm{pe}}(l))_{v \in \mathcal{V}}$ where $\mathcal{D}_v^{\mathrm{pe}}(l) \subseteq \mathcal{D}_v \times \mathcal{D}_v$ for all $v \in \mathcal{V}$, such that for all $s, s' \in S$, we have $(s, l, s') \in T$ iff $(s(v), s'(v)) \in \mathcal{D}_v^{\mathrm{pe}}(l)$ for all $v \in \mathcal{V}$.

PROOF SKETCH. Proposition 2.4(i) implies Definition 2.3(i) because recombining states preserves the property that $s(v) \in \mathcal{D}_v^{\mathrm{g}}$ for all $v \in \mathcal{V}$. For the other direction, define $\mathcal{D}_v^{\mathrm{g}} := \{s_\star(v) \mid s_\star \in S_\star\}$. Say that s is a state where, for all v, $s(v) \in \mathcal{D}_v^{\mathrm{g}}$. Then, by construction, for each variable v we have a state $s_\star^v \in S_\star$ with $s_\star^v(v) = s(v)$. By Definition 2.3(i) these states $s_\star^v$ can be recombined to form s. On the other hand, if s is a state where $s(v) \notin \mathcal{D}_v^{\mathrm{g}}$ for some v, then s cannot be a goal state by definition of $\mathcal{D}_v^{\mathrm{g}}$. Together, this shows the claim. Condition (ii) is similar. □

The terminology of Proposition 2.4 is borrowed from the domain of AI planning (as detailed in Section 2.2 below), but many other modelling formalisms also satisfy the criteria for giving rise to factored transition systems. For example, a transition in a Petri net (which in our terminology is actually a transition label) also has a precondition and effect on each place it is connected to, and the enabling condition for the transition as a whole is the conjunction of these conditions. Note, however, that some common formalisms do not. As an example, planning formalisms that include conditional effects [Pednault 1989] do not, in general, describe factored systems.[3] The reason is precisely that with conditional effects, the effect that a transition label (i.e., action) has on a single state variable may depend on the values of other state variables at the time the transition is taken, without those variables affecting whether the transition is enabled or not.

[3]However, conditional effects can be compiled away at the cost of a polynomial increase in plan length [Nebel 2000].

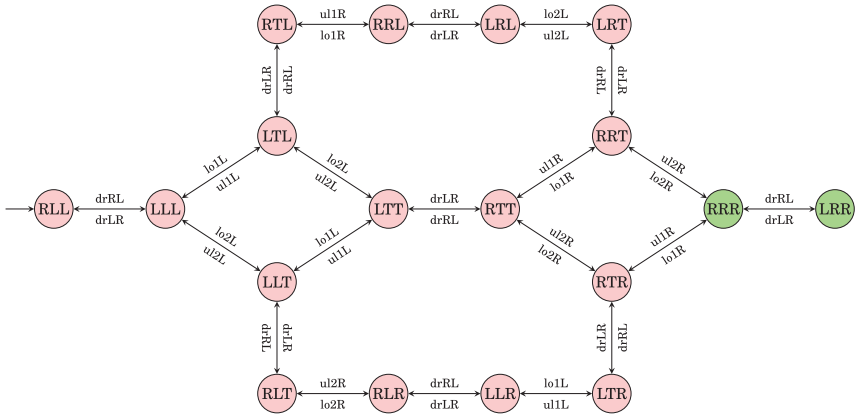


Fig. 1. The (factored) state space of our illustrative example.

We will henceforth assume that transition systems are factored, and that they are described in terms of the goal domains $\mathcal{D}^{\mathrm{g}}$ and the precondition/effect domains $\mathcal{D}^{\mathrm{pe}}(l)$ as specified (or in a form easily convertible into these). We further assume that, for all v and l, $\mathcal{D}_v^{\mathrm{g}}$ and $\mathcal{D}_v^{\mathrm{pe}}(l)$ are nonempty. (If $\mathcal{D}_v^{\mathrm{g}} = \emptyset$, then by Proposition 2.4(i) $S_\star$ is also empty, and the reachability problem trivially has no solution; if $\mathcal{D}_v^{\mathrm{pe}}(l) = \emptyset$, then l cannot appear as the label of any transition in Θ, and so can be removed from the label set.)

Example 2.5. Consider the transition system shown in Figure 1. This example is based on a simplified logistics planning problem, and will be used for illustration throughout this article. To be feasible for illustration, the example is very simplistic. We have the state variables $\mathcal{V} = \{truck, package\text{-}1, package\text{-}2\}$, whose domains are $\mathcal{D}_{truck} = \{L, R\}$, $\mathcal{D}_{package\text{-}1} = \{L, T, R\}$, and $\mathcal{D}_{package\text{-}2} = \{L, T, R\}$. In words, the example involves one truck and two packages that move between the two locations left (L) and right (R). In Figure 1, each state shows the value of $truck, package\text{-}1, package\text{-}2$ in this order. The initial state (at the left) is marked with an incoming arc not coming from any other state; goal states (at the right) are shown with green background. We will follow these conventions throughout this article. The labels correspond to driving the truck (e.g., "drLR" drives from left to right), loading a package (e.g., "lo1L" loads package 1 at the left), and unloading a package (e.g., "ul1R" unloads package 1 at the right). All transitions in the system are bidirectional; the label above a transition moves rightward ("$\rightarrow$"), the label below moves leftward ("$\leftarrow$"). The label costs are uniformly 1, and we will henceforth ignore them when discussing this example.

The transition system is factored. To see this, observe the conjunctive nature of the distribution of labels and solution states. The solution states are those where $package\text{-}1 = package\text{-}2 = R$. Transitions labeled with the operator of driving from left to right (label "drLR") connect exactly pairs of states where $truck = L$ in the current state, $truck = R$ in the new state, and the states are otherwise the same. Thus, the precondition of this transition label (or operator) is $truck = L$ and its effect is $truck = R$. The other operators are similarly defined.

Factoredness is important because, as will be shown later (Theorem 4.5), it allows Θ to be reconstructed from the projections onto each of its variables. This captures the case when Θ is *defined* as the product of a collection of automata, but is more general in capturing also transition systems not defined in this way a priori.

Merge-and-shrink heuristics can be derived and used also for transition systems that are not factored (yet compactly described), but many theoretical qualities of merge-and-shrink—everything that relies on Theorem 4.5—are then lost.

2.2. AI Planning

We will evaluate the effectiveness of merge-and-shrink heuristics, theoretically and empirically, in their application to AI planning. Planning is one of the oldest subareas of AI, originating in the early 1960s with the vision to achieve human problem solving flexibility [Newell and Simon 1963]. A plethora of approaches to planning have been proposed; see Ghallab et al. [2004] for a comprehensive overview. The approaches differ widely in intention and scope, but all have the common aim to design a tool that automatically finds a "plan", based on a high-level description of an "initial state", a "goal", and a set of available "operators". A plan is a schedule of operators that transforms the initial state into a state satisfying the goal. We consider problems falling into the category of so-called classical planning, arguably the most common form of AI planning. It assumes that the initial state is fully known and that operator effects are deterministic.

It would not serve this article to give a lengthy introduction to the practical uses of planning today. Its traditional use is in robotics, or more generally for controlling autonomous agents. Thanks to its flexibility, planning has recently been successfully applied in areas as diverse as large-scale printer control [Do et al. 2008; Ruml et al. 2011], natural language sentence generation [Koller and Hoffmann 2010; Koller and Petrick 2011], greenhouse logistics [Helmert and Lasinger 2010], and network security [Boddy et al. 2005; Lucangeli Obes et al. 2010]. All these recent applications are based on classical planning.

As mentioned, a planning task is provided to the automated planning tool in the form of a high-level description, that is, a modelling formalism. Even for classical planning, several different formalisms exist. We use a variant of the SAS$^+$ formalism [Bäckström and Nebel 1995], based on finite-domain state variables. This formalism is well suited to the construction of merge-and-shrink heuristics and is in widespread use. It is a direct generalization of the canonical classical planning formalism, STRIPS [Fikes and Nilsson 1971; McDermott et al. 1998], which allows Boolean variables only. Its formal definition is as follows.

Definition 2.6. A *finite-domain variable planning task*, or *planning task* for short, is a 4-tuple $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$ with the following components:

(i) $\mathcal{V}$ is a finite set of *state variables*, where each variable v has an associated finite domain $\mathcal{D}_v$. A *partial variable assignment* over $\mathcal{V}$ is a function s on a subset $\mathcal{V}_s$ of $\mathcal{V}$ such that $s(v) \in \mathcal{D}_v$ for all $v \in \mathcal{V}_s$. If $\mathcal{V}_s = \mathcal{V}$ then s is called a *state*. We identify (partial) variable assignments s with their set $\{(v, s(v)) \mid v \in V_s\}$ of variable/value pairs and write individual pairs (v, d) as $v = d$.
(ii) $\mathcal{O}$ is a finite set of *operators*, where each operator $o \in \mathcal{O}$ has an associated *precondition* pre_o and *effect* eff_o, both of which are partial variable assignments, and an associated *cost* $cost_o \in \mathbb{R}_0^+$.
(iii) s_0 is a state called the *initial state*, and $s_\star$ is a partial variable assignment called the *goal*.

The *state space* associated with Π is the transition system $\Theta(\Pi) = (S, L, T, s_0, S_\star)$, where: S is the set of all states; $L = \mathcal{O}$ with label costs $c(o) := cost_o$; $(s, o, s') \in T$ iff $pre_o \subseteq s$ and $s' = s|_{\mathcal{V} \setminus \mathcal{V}_{eff_o}} \cup eff_o$; $s \in S_\star$ iff $s_\star \subseteq s$. A solution for $\Theta(\Pi)$ is a *plan* for Π. The plan is *optimal* if the solution is.

This definition is sufficiently permissive that the problem of deciding if a given planning task has a solution (plan existence) is **PSPACE**-complete [Bylander 1994]. In other words, planning tasks are compact descriptions of potentially exponentially large transition systems, and hence they exhibit the state explosion problem. The transition systems associated with planning tasks are factored.

PROPOSITION 2.7. *Let $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$ be a planning task. Then $\Theta(\Pi)$ is factored with state variables $\mathcal{V}$.*

PROOF SKETCH. This is a straightforward application of Proposition 2.4. For example, goal domains are given by $\mathcal{D}_v^{\mathrm{g}} := \{s_\star(v)\}$ if $s_\star(v)$ is defined, and $\mathcal{D}_v^{\mathrm{g}} := \mathcal{D}_v$ otherwise. □

Example 2.8. Reconsider Figure 1. Clearly, this is the state space of the planning task $(\mathcal{V}, \mathcal{O}, s_0, s_\star)$ where: $\mathcal{V} = \{truck, package\text{-}1, package\text{-}2\}$ with $\mathcal{D}_{truck} = \{L, R\}$, $\mathcal{D}_{package\text{-}1} = \{L, T, R\}$, and $\mathcal{D}_{package\text{-}2} = \{L, T, R\}$; $s_0 = \{truck = R, package\text{-}1 = L, package\text{-}2 = L\}$; $s_\star = \{package\text{-}1 = R, package\text{-}2 = R\}$; and there are the following ten operators:

—"drLR" with precondition $\{truck = L\}$, effect $\{truck = R\}$ and cost 1
—"drRL" with precondition $\{truck = R\}$, effect $\{truck = L\}$ and cost 1
—four operators of the form "loij" ($i = 1, 2$, $j = L, R$) with precondition $\{truck = j, package\text{-}i = j\}$, effect $\{package\text{-}i = T\}$ and cost 1
—four operators of the form "ulij" ($i = 1, 2$, $j = L, R$) with precondition $\{truck = j, package\text{-}i = T\}$, effect $\{package\text{-}i = j\}$ and cost 1

In the following, we consider the most general framework for merge-and-shrink, factored transition systems. With Proposition 2.7, planning is a special case. We will return to that special case in Section 8, where we evaluate the performance of merge-and-shrink when applied to planning.

2.3. Heuristic Search

Heuristic search algorithms rely on a heuristic function to guide exploration.

Definition 2.9. Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system. A *heuristic function*, or *heuristic* for short, is a function $h : S \to \mathbb{R}_0^+ \cup \{\infty\}$. The heuristic is *admissible* if, for every $s \in S$, $h(s) \leq h^*(s)$. The heuristic is *consistent* if, for every $(s, l, s') \in T$, $h(s) \leq h(s') + c(l)$. The heuristic is *perfect* if $h = h^*$.

"Admissible" is essentially a synonym for "lower bound" (on h^*). We prefer the term "admissible" over "lower bound" in this article because of its ubiquity in the AI and search literature. Consistency together with the requirement that $h(s_\star) = 0$ for all goal states $s_\star$ implies admissibility. The heuristics created by merge-and-shrink are consistent, and, as we will show, in some cases even perfect.

A large body of literature is concerned with the possible uses of heuristic functions, that is, how to use heuristics to control the search on Θ. Provided with an admissible heuristic, some (but not all) heuristic search algorithms are guaranteed to return an optimal solution. Our focus is on optimal solutions, and hence we only consider such algorithms. Since we are investigating the derivation and computation of heuristics rather than their use, we evaluate merge-and-shrink heuristics using only the canonical optimal search algorithm, A* [Hart et al. 1968]. A* starts by expanding s_0, that is, generating its direct successors and inserting these into a search queue. A* then expands search states s by increasing order of $g(s) + h(s)$, where $g(s)$ is the accumulated cost on the path from s_0 to s that was generated by the search algorithm. If the expanded state is a goal state, A* stops. If the heuristic is admissible, the solution found, that is, the path by which the goal state was reached, is guaranteed to be optimal. If

the heuristic is also consistent, A* will reach every state it expands by an optimal path, so that it suffices to expand every state at most once, that is, duplicate search nodes arising from transpositions (several paths leading to the same state) can be pruned once the state has been expanded. If the heuristic is perfect, then the transition system contains a solution iff $h(s_0) \neq \infty$. In that case, A* does not need to search—it finds an optimal solution after expanding a number of nodes linear in the solution's length—provided that we break ties in favor of states s with smaller $h(s)$, and that there are no 0-cost transitions. (To understand the latter restriction, consider the case where all transition costs are 0, in which the perfect heuristic value is 0 in all states from which the goal is reachable.)

3. A MATHEMATICAL VIEW OF ABSTRACTIONS

Abstractions are at the core of our approach to constructing heuristics. In this section, we introduce them from a mathematical perspective. Algorithmic aspects—that is, how to actually compute these abstractions and their associated heuristics—will be discussed in the next section.

Section 3.1 defines abstractions and their associated heuristic functions. Section 3.2 defines pattern databases. Section 3.3 specifies how to merge abstractions. Section 3.4 introduces merge-and-shrink abstractions as proposed in this work, and explains how they generalize pattern databases.

3.1. Abstractions and Abstraction Heuristics

Abstracting in general means to ignore some information or some constraints to render a problem easier. In our case, it means to impose an equivalence relation over the states in a transition system. States within the same equivalence class are not distinguished in the abstraction. It will be convenient to define such abstractions as functions on states, where states deemed to be equivalent are mapped to the same abstract state.

Definition 3.1. Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system. An *abstraction* of Θ is a surjective function α mapping S to a set of *abstract states* S^α. We associate α with:

(i) the *abstract transition system* $\Theta^\alpha := (S^\alpha, L, T^\alpha, s_0^\alpha, S_\star^\alpha)$, where $T^\alpha := \{(\alpha(s), l, \alpha(s')) \mid (s, l, s') \in T\}$, $s_0^\alpha := \alpha(s_0)$, and $S_\star^\alpha := \{\alpha(s_\star) \mid s_\star \in S_\star\}$;
(ii) the *abstraction heuristic*, h^α, which maps each $s \in S$ to the optimal solution cost of $\alpha(s)$ in Θ^α; in other words, $h^\alpha(s) = h^*(\alpha(s))$;
(iii) the *induced equivalence relation on* Θ, $\sim^\alpha$, where $s \sim^\alpha t$ iff $\alpha(s) = \alpha(t)$.

The *size* of α, written $|\alpha|$, is the number of abstract states, $|S^\alpha|$.

Surjectivity of α is postulated only for simplicity of notation, to avoid useless states in Θ^α. An example of abstraction will be given in the next section.

We will also refer to the abstraction heuristic as the *induced heuristic function*. It is easy to see from the above definition that this heuristic is admissible and consistent. For example, to show admissibility, consider an optimal solution path $s = s_0, l_1, s_1, \ldots, l_n, s_n$ in Θ, where $s_n \in S_\star$ is a solution state: by definition of Θ^α, $(\alpha(s_{i-1}), l_i, \alpha(s_i))$ for $i = 1, \ldots, n$ is a transition in T^α, and $\alpha(s_n) \in S_\star^\alpha$, so the corresponding path is a solution for $\alpha(s)$ in Θ^α. Thus, the minimum cost of reaching a solution state from $\alpha(s)$ in Θ^α cannot be greater than the minimum cost of reaching a solution state from s in Θ, that is, h^α does not overestimate.

Note that Θ^α is isomorphic to $\Theta/\sim^\alpha$: each abstract state in S^α is the image of some set of states in S under α, and because they are all mapped to the same abstract state, that set of states is an equivalence class of $\sim^\alpha$. Both views of the abstract space are useful; we will use both views interchangeably. The quotient $\Theta/\sim^\alpha$ will be used to formulate certain classes of abstractions (whose induced equivalence relations are bisimulations).

The abstract transition system Θ^α is what underlies our algorithms, where α is defined in terms of partial variable assignments to compactly represent equivalence classes. For example, consider the abstract state "$\{v = d\}$" vs. the equivalence class of all states where the value of variable v is d.

To compute the abstraction heuristic, we rely on Θ^α being small enough to be represented explicitly. If α is injective, then Θ^α is identical to Θ, and its size is the same. As we have already mentioned, systems of practical interest are nearly always too large to be represented explicitly in this form. Thus, only noninjective abstractions are interesting for deriving heuristics. The potential power of abstraction heuristics stems from the observation that it is often possible to capture a significant amount of information about a system using an exponentially smaller abstract version of it.

In Definition 3.1, we fix the abstract transition system induced by an abstraction mapping α to be the "most precise" transition system for α: Θ^α is the transition system over the set of abstract states S^α that has exactly those transitions and solution states that are necessary for α to be a homomorphism into Θ^α and no more. A slightly more general definition, used in our earlier work [Helmert et al. 2007], leaves more flexibility in choosing Θ^α. In that definition, abstractions are defined by a pair consisting of a mapping α and any transition system Θ' over the set of abstract states S^α such that α is a (not necessarily strict) homomorphism into Θ'. This allows the abstract state space to contain additional solution states and/or transitions, which still results in an admissible and consistent (though potentially less accurate) heuristic. In this work, we prefer Definition 3.1 because it simplifies notation, and the generalization afforded by the alternative definition is irrelevant to almost all of our results. (The only situation where the generalization may matter is discussed informally at the end of Section 7.1.)

3.2. Projections

A projection is a particular kind of abstraction which considers two states s and s' to be equivalent if and only if they agree on the values of all variables in a distinguished subset. This corresponds to ignoring the values of all variables not in the distinguished set, that is, restricting the state to the selected subset of variables.

Definition 3.2. Let $\Theta = (S, L, T, s_0, S_\star)$ be a factored transition system with state variables $\mathcal{V}$, and let $V \subseteq \mathcal{V}$ be a subset of these variables. The *projection* onto V is the abstraction π_V of Θ defined by $\pi_V(s) = s|_V$. If V is a singleton set, $\pi_{\{v\}}$ is called an *atomic projection* and is also written as π_v.

Abstraction heuristics based on projections are known as pattern databases (PDBs); the term "pattern" refers to the subset of variables projected on. PDBs have been used extensively in heuristic search, and have been shown to be very effective for many problems (e.g., Korf [1997], Culberson and Schaeffer [1998], Korf and Zhang [2000], and Edelkamp [2001]). However, as mentioned in the introduction, PDB heuristics have a fundamental limitation: because the abstract transition system is represented explicitly, the number of variables included in the projection pattern must be small. In general, if we consider families of compactly represented transition systems of scaling size, polynomial-size PDBs must limit the number of variables in the pattern to be at most logarithmic in the problem size (if variable domains remain fixed with increasing problem size) or even constant (if variable domains increase with problem size). At the same time, the PDB heuristic provides no information about variables not included in the pattern. Transitions that depend on these ignored variables tend to create many "shortcuts" in the abstract space, and this may result in very uninformative heuristics. The following example illustrates this.

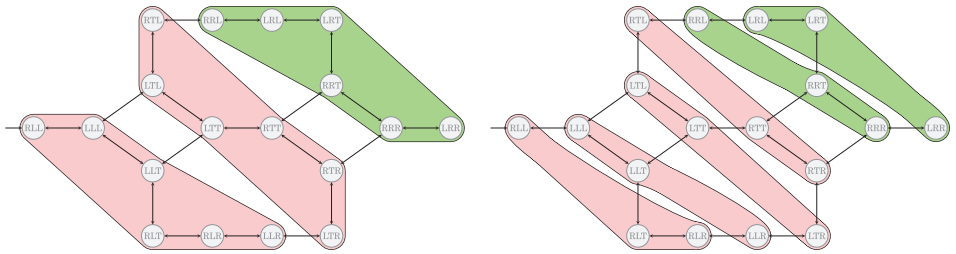


Fig. 2. Projections in our illustrative example, as equivalence relations over the state space. Left: project onto {*package*-1}; right: project onto {*truck*, *package*-1}.

Example 3.3. Figure 2 shows two projections of our illustrative example. The left-hand side of the figure shows a projection onto a single one of the two package variables. More precisely, what we see is the equivalence relation induced by the abstraction $\pi_{package\text{-}1}$. Since, in $\Theta^{\pi_{package\text{-}1}}$, transitions are between equivalence classes rather than between individual states, the solution cost of $\pi_{package\text{-}1}(s_0)$ in $\Theta^{\pi_{package\text{-}1}}$ is 2: all we need to do is to load package 1 at the left and unload it at the right, disregarding the roles of the truck and the other package (note also how the set of solution states is enlarged in this abstraction). Thus, $h^{\pi_{package\text{-}1}}(s_0) = 2$, in contrast to the actual solution cost $h^*(s_0) = 6$ that arises from having to drive the truck 2 times, and having to deal with package 2.

Adding the truck into the projection—Figure 2 (right)—improves the lower bound a little, yielding $h^{\pi_{\{package\text{-}1,truck\}}}(s_0) = 4$. But we still do not account for *package*-2, and we will not be able to do so unless we use the trivial projection onto the whole variable set. More generally, if there are n packages, a polynomial-size projection cannot include more than a logarithmic number of them, so the error of the resulting abstraction heuristic, that is, the amount by which it underestimates relative to the true optimal solution cost, will grow without bound as n increases.

As the example shows, even in very simple cases pattern databases are not able to compactly represent an accurate lower bound. To some extent this flaw can be sidestepped by summing over several independent PDBs, but even this does not fully overcome the problem. The proof of Theorem 7.4 in Section 7.2 presents an example in which no additive ensemble of PDBs can guarantee any bound on heuristic error. Part of the motivation for merge-and-shrink abstractions is to overcome this weakness of pattern databases, by considering a larger class of abstractions than only projections. Intuitively, where a PDB represents a subset of the variables exactly, merge-and-shrink abstractions allow us to capture some information about all variables, but in a lossy way if this is required to keep abstraction size at bay. Before we introduce merge-and-shrink abstractions formally, we need to discuss a basic operation on abstractions.

3.3. Merging Abstractions

Given any two abstractions of a transition system, we can combine them into another abstraction of the same system that represents all information contained in both of them. We call this *merging* the two abstractions.

Definition 3.4. Let Θ be a transition system, and let α_1 and α_2 be abstractions of Θ. The *merged abstraction* $\alpha_1 \otimes \alpha_2$ of Θ is defined by $(\alpha_1 \otimes \alpha_2)(s) = (\alpha_1(s), \alpha_2(s))$.

The merged abstraction $\alpha_1 \otimes \alpha_2$ maps two states to the same abstract state only if both α_1 and α_2 do that. In that sense, it is "less abstract" than either of the constituent abstractions. This also means that the induced equivalence relation $\sim^{(\alpha_1 \otimes \alpha_2)}$ is a refinement of both $\sim^{\alpha_1}$ and $\sim^{\alpha_2}$.

A special case that will appear frequently in our technical discussion is that of merging two projections, $\alpha_1 = \pi_{V_1}$ and $\alpha_1 = \pi_{V_2}$. Obviously, $\pi_{V_1} \otimes \pi_{V_2}$ is isomorphic to $\pi_{V_1 \cup V_2}$. Indeed, the abstract transition systems differ only in that any variable common to V_1 and V_2 is mentioned twice, but always with identical values, in states of $\pi_{V_1} \otimes \pi_{V_2}$. Given this, we will not distinguish between the two abstractions, that is, we identify $\pi_{V_1} \otimes \pi_{V_2}$ with $\pi_{V_1 \cup V_2}$.

Readers familiar with the operation of forming the synchronized product of transition systems should note that, in general, the transition system $\Theta^{\alpha_1 \otimes \alpha_2}$ induced by the merged abstraction is not necessarily isomorphic to the product of the transition systems Θ^{α_1} and Θ^{α_2} induced by the component abstractions. Later, in Section 4, we will show that this isomorphism holds precisely when the transition system is factored and the two abstractions depend on disjoint sets of state variables.

Example 3.5. In Figure 2, if we merge the projection $\pi_{package\text{-}1}$ shown on the left hand side with the projection π_{truck} onto the truck, then the resulting abstraction $\pi_{package\text{-}1} \otimes \pi_{truck}$ is identical to the abstraction $\pi_{\{package\text{-}1,truck\}}$ shown on the right hand side—the states $(\{package\text{-}1 = d\}, \{truck = e\})$ of the former correspond to the states $\{package\text{-}1 = d, truck = e\}$ of the latter. If we merge $\pi_{package\text{-}1}$ with itself, then the states $(\{package\text{-}1 = d\}, \{package\text{-}1 = d\})$ of $\pi_{package\text{-}1} \otimes \pi_{package\text{-}1}$ correspond to the states $\{package\text{-}1 = d\}$ of $\pi_{package\text{-}1}$ itself.

We have $\bigotimes_{v \in \mathcal{V}} \pi_v = \pi_{\mathcal{V}}$. Since $\pi_{\mathcal{V}}$ is the identity function, $\Theta^{\bigotimes_{v \in \mathcal{V}} \pi_v}$ is identical to the original transition system Θ, that is, a transition system can be reconstructed by merging all its atomic projections. This is the basic observation underlying merge-and-shrink abstractions.

3.4. Merge-and-Shrink Abstractions

Given a compact description of a factored transition system Θ, arbitrary abstractions can be created by first building an explicit representation of Θ, then aggregating individual pairs of states to make the desired abstraction. Of course, as we have repeatedly pointed out, building such an explicit representation is infeasible in most situations, so this is not a practical approach to creating abstractions. Rather, we need to abstract already while constructing Θ, to keep the construction from growing out of bounds. But how do we do this?

In the case of merge-and-shrink abstractions, the answer is to exploit the reconstruction of Θ as $\Theta^{\bigotimes_{v \in \mathcal{V}} \pi_v}$ mentioned previously. We construct an abstraction of Θ incrementally by starting with the collection of atomic projections π_v, and iteratively *merging* two abstractions. In between the merging steps, we *shrink* (some of) the partial composites by applying additional abstractions. For clarity, our definition makes this inductive construction explicit in the form of a *merge-and-shrink tree*.

Definition 3.6. Let Θ be a factored transition system with variables $\mathcal{V}$, and let $V \subseteq \mathcal{V}$. A *merge-and-shrink tree* $\mathcal{T}$ over V for Θ is a finite rooted tree whose nodes are labeled with abstractions α of Θ. We say that $\mathcal{T}$ *represents* the abstraction α that labels the root of $\mathcal{T}$. Merge-and-shrink trees are defined inductively using the following rules.

(A) *Atomic Projections.* For any $v \in \mathcal{V}$, the tree $\mathcal{T}_v$ that only consists of a single node labeled π_v is a merge-and-shrink tree over $\{v\}$ for Θ.
(S) *Shrinking.* If $\mathcal{T}_2$ is a merge-and-shrink tree over V for Θ representing α_2 and α_1 is a surjective function on S^{α_2}, then the tree $\mathcal{T}_1$ whose root is labeled with $\alpha_1 \circ \alpha_2$ and has $\mathcal{T}_2$ as its only child is a merge-and-shrink tree over V for Θ.

(M) *Merging.* If $\mathcal{T}_1$ and $\mathcal{T}_2$ are merge-and-shrink trees over V_1 and V_2 for Θ representing α_1 and α_2 such that $V_1 \cap V_2 = \emptyset$, then the tree $\mathcal{T}$ whose root is labeled with $\alpha_1 \otimes \alpha_2$ and has $\mathcal{T}_1$ and $\mathcal{T}_2$ as its children is a merge-and-shrink tree over $V_1 \cup V_2$ for Θ.

We say that $\mathcal{T}$ is *atomic* if it can be constructed using only rules (A) and (S); else, it is *nonatomic*. We say that $\mathcal{T}$ is *nonabstracting* if it can be constructed using only rules (A) and (M). The *maximum intermediate abstraction size* of $\mathcal{T}$, or *size* for short, written $|\mathcal{T}|^{\max}$, is the maximum over the sizes of the abstractions labeling the nodes in $\mathcal{T}$.

By $\mathcal{T}^\alpha$, we denote a merge-and-shrink tree whose root is labeled with α. When the distinction does not matter, we identify $\mathcal{T}$ with the abstraction it represents. In particular, when we say that α is a *merge-and-shrink abstraction* we mean that α is represented by a merge-and-shrink tree whose details are not important for the discussion.

We will refer back to rules (A), (S), and (M) throughout this article, so it is important to keep them in mind. *Rule (A)* allows us to start from atomic projections. *Rule (S)*, the *shrinking rule*, allows us to abstract a given abstraction α_2 further, by applying an additional abstraction α_1 to its image, thus reducing the size of its induced abstract transition system. *Rule (M)*, the *merging rule*, allows us to merge a pair of abstractions provided their associated variable sets are disjoint. As will be detailed in Section 4.1, the restriction to disjoint variable sets is required for reconstructing the abstract state space of the merged abstraction from those of its components. In theory, the restriction can be dropped; that possibility plays a role for some properties of merge-and-shrink abstractions that will be discussed in Sections 7.4 and 7.5.

Observe that any abstraction α is trivially a merge-and-shrink abstraction. A naïve merge-and-shrink tree $\mathcal{T}$ representing α is obtained by first constructing $\bigotimes_{v \in \mathcal{V}} \pi_v$ with rules (A) and (M), then using rule (S) instantiated with $\alpha_2 = \bigotimes_{v \in \mathcal{V}} \pi_v$ and $\alpha_1 = \alpha$. The important question is which abstractions we can represent *compactly* as a merge-and-shrink tree. We capture this in terms of maximum intermediate abstraction size $|\mathcal{T}|^{\max}$, the maximum size of any abstraction created throughout the merge-and-shrink process resulting in α.[4] The naïve $\mathcal{T}$ above is not compact, its size being equal to the size of the original transition system. An important class of compactly representable abstractions are projections.

PROPOSITION 3.7. *Let Θ be a factored transition system with variables $\mathcal{V}$, and let $V \subseteq \mathcal{V}$. Let $\mathcal{T}^\alpha$ be a nonabstracting merge-and-shrink tree over V. Then, $\Theta^\alpha = \Theta^{\pi_V}$, and $|\mathcal{T}|^{\max} = |\pi_V|$.*

PROOF. Not applying any shrinking steps, clearly $\alpha = \bigotimes_{v \in V} \pi_v$, thus $\Theta^\alpha = \Theta^{\bigotimes_{v \in V} \pi_v}$ which with our notational conventions is identical to the first part of the claim. The second part of the claim holds obviously. □

In other words, merge-and-shrink abstractions generalize pattern databases, which can be constructed using only rules (A) and (M), with merge-and-shrink trees whose size relates polynomially to the size of the pattern database. The generalization is strict, in the sense that merge-and-shrink can compactly represent the perfect heuristic in certain transition system families where PDBs cannot (Theorem 7.4 will show that this is the case even for additive ensembles of PDBs).

Example 3.8. Figure 3(a) shows an abstraction α_1 of $\pi_{package\text{-}1} \otimes \pi_{package\text{-}2}$. Precisely, α_1 results from the following steps as per Definition 3.6. We first apply rule (A) to each

[4]At implementation level, other important size measures are the peak memory usage during the merge-and-shrink process, as well as the memory usage of the final heuristic function representation at the end of the merge-and-shrink process. Both measures relate polynomially to maximum intermediate abstraction size, and for brevity, we do not discuss them in detail.

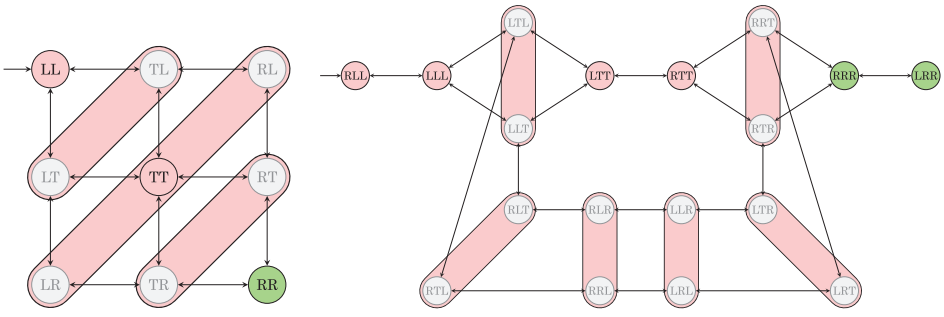


Fig. 3. Merge-and-shrink abstractions in our illustrative example, shown as equivalence relations. (a) A merge-and-shrink abstraction α_1 constructed by shrinking $\pi_{package\text{-}1} \otimes \pi_{package\text{-}2}$ (TT is not a member of the equivalence class containing LR and RL). (b) $\alpha_1 \otimes \pi_{truck}$, yielding a perfect heuristic; for readability, the layout has been rearranged with respect to Figures 1 and 2.

of *package*-1 and *package*-2. Then, we apply rule (M) to the resulting two abstractions $\pi_{package\text{-}1}$ and $\pi_{package\text{-}2}$. We finally apply rule (S), removing the distinction between the state pairs $\{LT, TL\}$, $\{LR, RL\}$, and $\{TR, RT\}$. This results in the equivalence relation induced by α_1 as shown in the figure.

Note that the shrinking step has removed any distinction between *permutations* of the two packages. Indeed, no matter where the truck is located, such permutations have no influence on the solution cost. Therefore, merging α_1 with the projection π_{truck} onto the truck variable yields a perfect heuristic. The equivalence relation induced by $\alpha_1 \otimes \pi_{truck}$, in the full state space of the example, is shown in Figure 3(b). Such removal of permutations can be done for an arbitrary number of packages, and incrementally. Thus, in difference to pattern databases (cf. Example 3.3), merge-and-shrink can compute a perfect heuristic for a generalization of the example, with an arbitrary number of packages, in polynomial time.

In this example, the ability of merge-and-shrink to compactly represent a perfect heuristic exploits the symmetry between packages. The same phenomenon can be observed in some other cases for which merge-and-shrink yields perfect heuristics, discussed in Section 8. However, it is possible to construct problem families that have no symmetries, but where a polynomially sized perfect merge-and-shrink abstraction still exists. For example, say we have n independent variables, each of which can be changed from its initial to its target value in two steps, but where each variable has several intermediate values connected to each other in different non-symmetric ways. A perfect merge-and-shrink abstraction reduces to counting how many of the variables are one or two steps, respectively, from their target value. Conversely, there exist problems that with certain kinds of symmetry reduction can be solved in polynomial time, but which do not admit compact perfect merge-and-shrink abstractions. Thus, while there is some overlap in the problem features exploited by symmetry reduction and merge-and-shrink abstractions, neither is completely subsumed by another. The same is true of partial-order reduction (e.g., Valmari [1989] and Clarke et al. [2000]). (To show a separation with merge-and-shrink abstraction, it suffices to extend the above example with a binary variable whose value is changed by each transition.)

Note that Proposition 3.7 holds even if we drop the requirement that $V_1 \cap V_2 = \emptyset$ in rule (M). That constraint plays an important role in the *computation* of h^α, which is, in principle, separate from the *design* of α. We could design α first, and only thereafter build its abstract transition system and use that to compute h^α. However, we do both

at the same time: while constructing α, we maintain also the abstract system Θ^α associated with it. The advantage is that Θ^α provides information that we can use to decide which states are most useful to aggregate in each shrinking step. We will return to this issue later. Next, we describe how this incremental maintenance of the induced transition system during the construction of a merge-and-shrink abstraction is done.

4. AN ALGORITHMIC VIEW OF ABSTRACTIONS

We now discuss merge-and-shrink abstractions from an algorithmic perspective, explaining how to represent and compute them efficiently.

Section 4.1 explains how we build the abstract state space in parallel with the construction of a merge-and-shrink abstraction. Section 4.2 presents, at a high level, the algorithm we use to create merge-and-shrink abstraction heuristics. Section 4.3 briefly discusses some data structures and optimizations that are important for the efficient implementation of this algorithm.

4.1. Maintaining Abstract State Spaces in Merge-and-Shrink

Assume a merge-and-shrink abstraction α, constructed according to the rules (A), (S), (M) of Definition 3.6. We show how to build the abstract transition system Θ^α inductively alongside the construction of the abstraction. The core technique is to reconstruct the abstract transition system of a merged abstraction as the synchronized product of the abstract transition systems associated with its components. This is where the factoredness of the transition system becomes important: as we will show, factoredness is necessary for this operation to be correct. Indeed, we will identify a necessary and sufficient condition for this to be the case. First, we recap the definition of the synchronized product of transition systems.

Definition 4.1. Let $\Theta^1 = (S^1, L^1, T^1, s_0^1, S_\star^1)$ and $\Theta^2 = (S^2, L^2, T^2, s_0^2, S_\star^2)$ be transition systems. The *synchronized product* of Θ^1 and Θ^2 is defined as $\Theta^1 \otimes \Theta^2 = (S^{12}, L^{12}, T^{12}, s_0^{12}, S_\star^{12})$ where $S^{12} = S^1 \times S^2$, $L^{12} = L^1 \cap L^2$, $((s_1, s_2), l, (s_1', s_2')) \in T^{12}$ iff $(s_1, l, s_1') \in T^1$ and $(s_2, l, s_2') \in T^2$, $s_0^{12} = (s_0^1, s_0^2)$, and $S_\star^{12} = S_\star^1 \times S_\star^2$.

Let us exemplify how this product of components can indeed reconstruct the abstract transition system of a merged abstraction.

Example 4.2. Consider Figure 4, showing a synchronized product in our illustrative example. In part (a) of the figure, we show the abstract state space $\Theta^{\pi_{\{package\text{-}1, package\text{-}2\}}}$ of the projection onto $\{package\text{-}1, package\text{-}2\}$. In part (b), we show the abstract state space $\Theta^{\pi_{truck}}$ of the projection onto $\{truck\}$. These will play the role of Θ^1 and Θ^2 in Definition 4.1. What we will see is that their synchronized product, $\Theta^{\pi_{\{package\text{-}1, package\text{-}2\}}} \otimes \Theta^{\pi_{truck}}$, is equal to $\Theta^{\pi_{\{package\text{-}1, package\text{-}2\}} \otimes \pi_{truck}}$, that is, to the original state space because the abstraction $\pi_{\{package\text{-}1, package\text{-}2\}} \otimes \pi_{truck}$ is injective.

Note that the labels we use here are the planning operators as per Example 2.8. This will be important for our label reduction techniques, to be discussed in the next section. Note also the self-loops in (a) and (b). These correspond to operators that do not affect the variable(s) in question. They are unimportant for each of (a) and (b) in its own right (they do not affect solution cost). They are important, however, to be able to properly synchronize the two systems.[5]

The combined state space $\Theta^{\pi_{\{package\text{-}1, package\text{-}2\}}}$ of the two package variables is just a grid multiplying all the transitions of the individual packages. This is because the possible transitions for *package*-1 do not depend on the current value of *package*-2, and vice

[5]The presence of these self-loops is, in particular, what distinguishes an atomic projection onto a variable from its so-called "domain transition graph" [Helmert 2004].

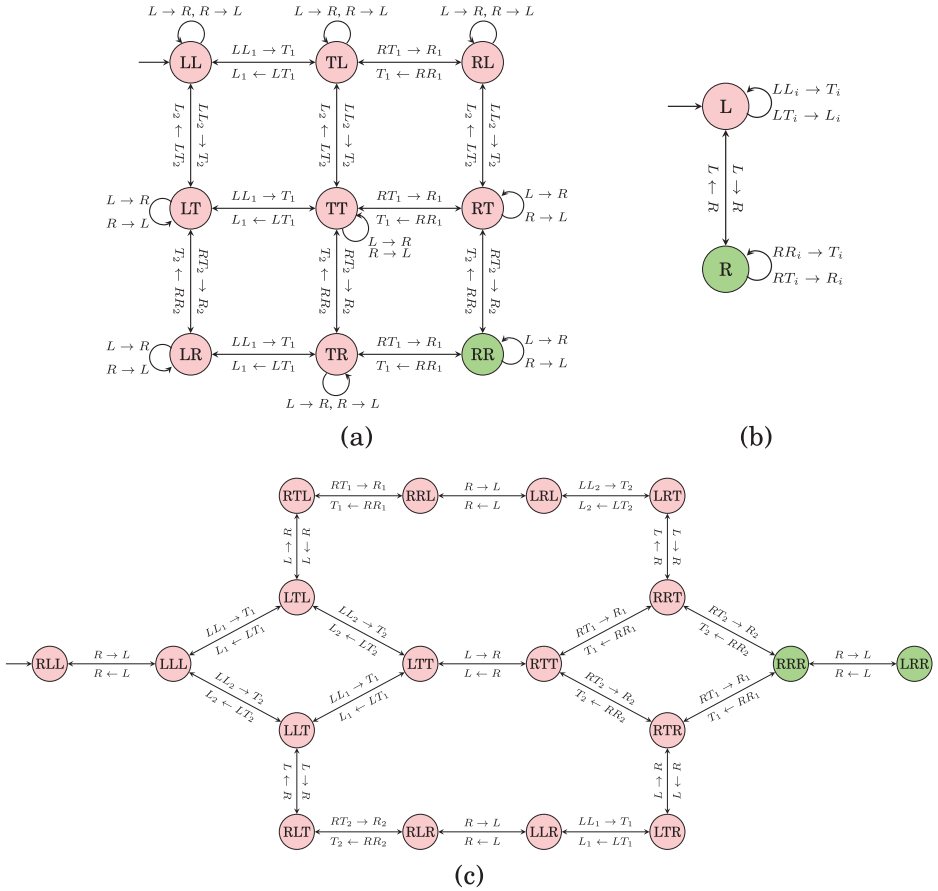


Fig. 4. A synchronized product in our illustrative example: (a) is the abstract state space $\Theta^{\pi_{\{package\text{-}1,package\text{-}2\}}}$ of the projection onto $\{package\text{-}1, package\text{-}2\}$; (b) is the abstract state space $\Theta^{\pi_{truck}}$ of the projection onto $\{truck\}$; (c) is $\Theta^{\pi_{\{package\text{-}1,package\text{-}2\}}} \otimes \Theta^{\pi_{truck}}$, which is equal to $\Theta^{\pi_{\{package\text{-}1,package\text{-}2\}} \otimes \pi_{truck}}$ (and thus to the original state space). The labels are taken to be the respective planning operators, cf. Example 2.8; we use arrows $\rightarrow, \leftarrow$ to make these operators more readable; we use L and R to refer to values of the *truck* variable, and L_i, T_i, R_i to refer to values of variable *package-i*.

versa. Evidently, this is not so for the product of $\Theta^{\pi_{\{package\text{-}1,package\text{-}2\}}}$ with $\Theta^{\pi_{truck}}$, as shown in Figure 4(c). The reason is that the possible transitions for each package *do* depend on where the truck currently is. In Figure 4(b), we see this difference in the self-loops attached to state L vs. state R: the former only allow loading/unloading operators at L, the latter only allow these operators at R. This difference results in the complex structure of the merged system (c).

To illustrate this, just consider the product of a single state in (a), LL, with the two states R and L of (b). This yields the states RLL and LLL on the left-hand side of (c). LL in (a) has the outgoing labels $LL_1 \rightarrow T_1$ to TL and $LL_2 \rightarrow T_2$ to LT, corresponding to loading the packages at L; these are self-loops of L in (b), hence LLL in (c) gets the outgoing arcs to LTL and LLT. By contrast, R in (b) has self-loops only for loading/unloading at R; these do not match any of the labels attached to LL in (a), hence RLL in (c) is connected only to LLL (via the truck moves, which are self-loop labels of every state in (a)).

The abstract transition system of a merged abstraction cannot always be reconstructed from the transition systems of its components. A sufficient and necessary condition is given by Theorem 4.5. To state and prove the theorem, it will be convenient to separate, notationally, the abstract transition system induced by an abstraction α, that is, Θ^α, which is mathematically defined as a function of α, from the abstract transition system that we build inductively alongside a merge-and-shrink abstraction. We refer to the latter as the transition system *associated with* α, and denote it by Θ_α. The correspondence which we will then prove is that $\Theta_\alpha = \Theta^\alpha$, for any merge-and-shrink abstraction α. Θ_α is defined inductively, by three rules that mirror those of Definition 3.6.

Definition 4.3. Let Θ be a factored transition system with variables $\mathcal{V}$, and let $\mathcal{T}^\alpha$ be a merge-and-shrink tree for Θ. Then, the *transition system associated with* α, written Θ_α, is constructed inductively using the following rules.

(A) If $\mathcal{T}^\alpha = \mathcal{T}_v$ for a variable $v \in \mathcal{V}$, then $\Theta_\alpha := \Theta^{\pi_v}$.
(S) If the root of $\mathcal{T}^\alpha$ has a single child $\mathcal{T}^{\alpha_2}$, and $\alpha = \alpha_1 \circ \alpha_2$, then $\Theta_\alpha := (\Theta_{\alpha_2})^{\alpha_1}$.
(M) If the root of $\mathcal{T}^\alpha$ has two children $\mathcal{T}^{\alpha_1}$ and $\mathcal{T}^{\alpha_2}$, then $\Theta_\alpha := \Theta_{\alpha_1} \otimes \Theta_{\alpha_2}$.

The correctness of this construction, that is, that $\Theta_\alpha = \Theta^\alpha$, follows from a straightforward induction over the three design rules for merge-and-shrink abstractions. The only difficult case is rule (M), where we need the reconstruction property discussed above. The necessary and sufficient condition for this case is, first, that the original transition system is factored (Definition 2.3), and, second, that the component abstractions being merged are independent, in the following sense.

Definition 4.4. Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system with state variables $\mathcal{V}$. Let α be an abstraction of Θ. We say that α *depends on variable* $v \in \mathcal{V}$ iff there exist states s and t with $\alpha(s) \neq \alpha(t)$ and $s(w) = t(w)$ for all $w \in \mathcal{V} \setminus \{v\}$. The set of *relevant variables* for α, written $rvars(\alpha)$, is the set of variables in $\mathcal{V}$ on which α depends. Abstractions α_1 and α_2 of Θ are *orthogonal* iff $rvars(\alpha_1) \cap rvars(\alpha_2) = \emptyset$.

THEOREM 4.5. *Let $\Theta = (S, L, T, s_0, S_\star)$ be a factored transition system, and let α_1 and α_2 be orthogonal abstractions of Θ. Then, $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2} = \Theta^{\alpha_1 \otimes \alpha_2}$.*

PROOF SKETCH. The equality is shown by considering each component—states, transitions, initial and solution states—of the transition systems in turn. As an example, we consider the solution states: If $(s^1_\star, s^2_\star)$ is a solution state in $\Theta^{\alpha_1 \otimes \alpha_2}$, then there exists a solution state $s_\star \in S_\star$ such that $\alpha_1(s_\star) = s^1_\star$ and $\alpha_2(s_\star) = s^2_\star$ are solution states in Θ^{α_1} and Θ^{α_2}, respectively. But then, by definition of the synchronized product, $(s^1_\star, s^2_\star)$ is also a solution state in $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$. The opposite direction is trickier. If $(s^1_\star, s^2_\star)$ is a solution state in $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$, then by definition $s^i_\star$ is a solution state in Θ^{α_i}, and there must exist two solution states $s_{\star i} \in S_\star$ such that $\alpha_i(s_{\star i}) = s^i_\star$. However, to apply $\alpha_1 \otimes \alpha_2$ and thus obtain a solution state in $\Theta^{\alpha_1 \otimes \alpha_2}$, we need a *single* state $s_\star \in S_\star$. This is where factoredness and orthogonality come into play. Let $s_\star = \rho_V(s_{\star 1}, s_{\star 2})$ where V is the set of variables that α_1 depends on. Thanks to factoredness (condition (i) of Definition 2.3), we have that $s_\star \in S_\star$. Thanks to orthogonality, $\rho(s_1, s_2)$ agrees with $s_{\star i}$ on all state variables on which α_i depends, and thus $\alpha_i(s_\star) = \alpha_i(s_{\star i})$. But then, $(\alpha_1 \otimes \alpha_2)(s_\star) = (s^1_\star, s^2_\star)$, and since $s_\star \in S_\star$ we have that $(s^1_\star, s^2_\star)$ is a solution state also in $\Theta^{\alpha_1 \otimes \alpha_2}$, as desired. □

Note that the first direction in the proof does not rely on either of orthogonality or factoredness. Indeed, $\Theta^{\alpha_1 \otimes \alpha_2}$ is always a substructure of $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$. This is not the case for the opposite direction. To see that orthogonality is required, consider the composition of a transition system with itself. In $\Theta^{\alpha_1} \otimes \Theta^{\alpha_1}$, the state set contains all

pairs of abstract states, whereas $\Theta^{\alpha_1 \otimes \alpha_1}$ is isomorphic to Θ^{α_1}. To see that factoredness is required, consider an example with only two Boolean variables v_1, v_2 where the solution states are $(v_1 = 1, v_2 = 0)$ and $(v_1 = 0, v_2 = 1)$, and α_i is the projection onto v_i. Then $v_i = 1$ is a solution state in Θ^{α_i} and thus $(v_1 = 1, v_2 = 1)$ is a solution state in $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$. But $(v_1 = 1, v_2 = 1)$ is not a solution state in $\Theta^{\alpha_1 \otimes \alpha_2} = \Theta$.

Theorem 4.5 is the basis of a number of properties of our algorithms, which will be discussed at various points in this article. In particular, it provides the crucial step in the proof of correctness of Definition 4.3.

COROLLARY 4.6. *Let Θ be a factored transition system with variables $\mathcal{V}$, and let $\mathcal{T}^\alpha$ be a merge-and-shrink tree for Θ. Then $\Theta_\alpha = \Theta^\alpha$.*

PROOF. This property clearly holds for atomic $\mathcal{T}^\alpha$. It is obviously invariant over rule (S). To see that the property is invariant over rule (M), just observe that, whenever α is a merge-and-shrink abstraction over V, then $rvars(\alpha) \subseteq V$. This is easy to see by induction: it trivially holds for rule (A); it is invariant over rule (S) because applying an additional abstraction can only reduce, never increase, the relevant variables; it is trivially invariant over (M). But then, whenever we merge α_1 and α_2 in (M), they are orthogonal. The claim follows from this with Theorem 4.5. □

Note that Corollary 4.6 holds also if we replace the condition $V_1 \cap V_2 = \emptyset$ in rule (M) of Definition 3.6 with the slightly weaker restriction that $rvars(\alpha_1) \cap rvars(\alpha_2) = \emptyset$, that is, that the two abstractions are orthogonal. This is not a useful generalization (it generalizes the original definition only if applications of rule (S) remove all distinctions between the values of a variable in an abstraction), so we have omitted it here for the sake of simplicity.

To make Definition 4.3 operational, we need to specify how to perform the steps required by rules (A), (S), (M). Rule (M) consists in forming the synchronized product of two transition systems, as per Definition 4.1. This, of course, can be done in time proportional to the product of the sizes of the component systems.

For rule (S), assume that α is an abstraction of Θ, and that we have an explicit representation of Θ—in Definition 4.3 (S), this applies to α_1 and Θ_{α_2}. We can compute the abstract transition system Θ^α by a single pass through Θ, aggregating the current state t into an already passed state s—that is, inserting t's transitions at s—whenever $\alpha(t) = \alpha(s)$. This takes time linear in the size of Θ, and requires $|S|$ calls to α (S being the states in Θ).

Consider finally rule (A) of Definition 4.3, which requires us to construct the abstract state space Θ^{π_v} of an atomic projection π_v. The ability to do so efficiently—without constructing Θ in the first place—relies on the factoredness of Θ. By Proposition 2.4, Θ can be described in terms of goal domains $\mathcal{D}^{\mathrm{g}}$ and precondition/effect domains $\mathcal{D}^{\mathrm{pe}}(l)$. Assuming our input is such a description, we can construct Θ^{π_v} directly from the domains $\mathcal{D}^{\mathrm{g}}_v$ and $\mathcal{D}^{\mathrm{pe}}_v(l)$. Namely, by definition, (d, l, d') is a transition in Θ^{π_v} iff there is a transition (s, l, s') in Θ where $s(v) = d$ and $s'(v) = d'$. By a direct application of Proposition 2.4, the latter is the case iff $(d, d') \in \mathcal{D}^{\mathrm{pe}}_v(l)$. Similarly, the solution states in Θ^{π_v} are given by $\mathcal{D}^{\mathrm{g}}_v$. We can thus construct Θ^{π_v} by creating a state for each value $d \in \mathcal{D}_v$, and traversing each of $\mathcal{D}^{\mathrm{g}}_v$ and $\mathcal{D}^{\mathrm{pe}}_v$ once to collect the solution states and insert the transitions. This takes time linear in the sum of the sizes of $\mathcal{D}_v$, $\mathcal{D}^{\mathrm{g}}_v$, and $\mathcal{D}^{\mathrm{pe}}_v$.

4.2. The Generic Merge & Shrink Algorithm

To provide an overview and to make explicit some choices we made in our implementation of the merge-and-shrink framework, Figure 1 presents a generic algorithm for the construction of merge-and-shrink abstractions.

ALGORITHM 1: Generic merge-and-shrink algorithm. Computes an abstraction for a factored transition system Θ, optionally with abstraction size bound N. For $A = (\alpha, \Theta)$, $|A|$ denotes the number of states in Θ.

Input: Factored transition system Θ with variables $\mathcal{V}$ [and size bound N].
Output: Merge-and-shrink abstraction α with associated abstract state space Θ^α.
$\mathcal{A} := \{(\pi_v, \Theta^{\pi_v}) \mid v \in \mathcal{V}\}$; **while** $|\mathcal{A}| > 1$ **do**
    Select $A_1 = (\alpha_1, \Theta_1), A_2 = (\alpha_2, \Theta_2) \in \mathcal{A}$; Shrink A_1 and/or A_2 [until $|A_1| \cdot |A_2| \leq N$];
    $A := (\alpha_1 \otimes \alpha_2, \Theta_1 \otimes \Theta_2)$; Shrink A; $\mathcal{A} := (\mathcal{A} \setminus \{A_1, A_2\}) \cup \{A\}$;
end
return *The only element of* $\mathcal{A}$*;*

The algorithm maintains a pool of orthogonal abstractions, which initially consists of all atomic projections. Each abstraction α is paired with its associated abstract state space Θ^α. The algorithm alternates between performing merge and shrink steps until only a single abstraction is left, at which point all atomic projections have been incorporated.

Note that the algorithm removes any "used" abstraction from the set $\mathcal{A}$. Hence, at any time, the underlying variable sets of all abstractions in the pool, that is, the sets of atomic projections incorporated into each of them, are disjoint. Thus, whenever two abstractions A_1 and A_2 are merged, the disjointness condition in rule (M) of Definition 3.6 is satisfied, and so, for all $(\alpha, \Theta) \in \mathcal{A}$ at any stage of the algorithm, α is a merge-and-shrink abstraction and by Corollary 4.6 we have that $\Theta = \Theta^\alpha$.

The algorithm is more specific than Definition 3.6, in that it creates only abstractions that incorporate all variables. This makes sense since, intuitively, the power of merge-and-shrink abstractions, compared to pattern databases, lies in being able to take into account some information about all variables. The algorithm also allows to impose a limit, N, on the size of any abstraction created and thus on maximum intermediate abstraction size. This is a simple means to keep time and space requirements under control: if N is polynomially bounded by the input size (e.g., a constant), then computing the abstraction requires only polynomial time and space. Note that this limit is optional; some of our implemented strategies do not apply it.

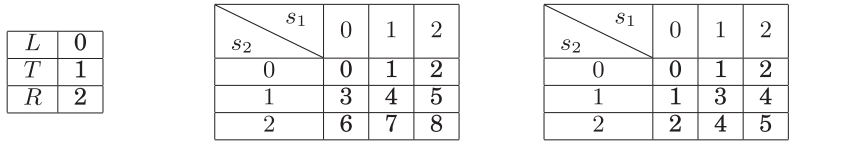
Speaking of strategies brings us to the two important choice points left unspecified in the generic algorithm.

—*Merging Strategy*. Which abstractions α_1 and α_2 to select from the current pool?
—*Shrinking Strategy*. Which states to aggregate, and in which abstractions?

We refer to the combination of a particular merging and shrinking strategy as an *abstraction strategy*. The generic algorithm must be augmented with such a strategy in order to obtain a concrete algorithm. For any abstraction strategy, the resulting heuristic function h^α is admissible and consistent. Still, selecting a suitable strategy is of paramount importance, as it determines the accuracy of the heuristic. It depends on the application domain—and even on the particular problem instance considered—what a good abstraction strategy is.

Our main tool for obtaining shrinking strategies is to *relax bisimilarity*. Bisimilarity [Milner 1990] is a well-known condition under which aggregating states does not result in a loss of information; we will discuss it in detail in Section 6. Since achieving bisimilarity is typically too costly in practice—it does not allow to aggregate a sufficient number of states to keep abstraction size small—we relax it, that is, we devise approximate versions. Some (but not all) of these assume a constant abstraction size bound N.

We do not investigate different merging strategies in depth in this paper, using only some simple strategies that tend to work well for AI planning. One thing to note is that



L	0
T	1
R	2

(a) $\pi_{package\text{-}1}, \pi_{package\text{-}2}$

s_2 \ s_1	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8

(b) $\pi_{package\text{-}1} \otimes \pi_{package\text{-}2}$

s_2 \ s_1	0	1	2
0	0	1	2
1	1	3	4
2	2	4	5

(c) $\alpha_1 \circ (\pi_{package\text{-}1} \otimes \pi_{package\text{-}2})$

Fig. 5. Part of the cascading tables implementing h^α in our running example. The abstraction α_1 on the right-hand side corresponds to the shrinking step illustrated in Figure 3(a).

our implemented strategies are, in fact, *linear merging strategies*: when we select two abstractions from the current pool to merge, after the first merging operation, one of the selected abstractions is always the abstraction added to the pool in the previous round. Thus, at any time, the pool contains at most one nonatomic abstraction. A linear merging strategy is fully specified by a variable order, $v_1, \ldots, v_n$, such that the algorithm incorporates the atomic projection of each variable sequentially in that order.

We describe our abstraction strategies with the experiments in Section 8.2. Next, we discuss some important implementation details for merge-and-shrink in general.

4.3. Efficient Implementation

During A* search on the original transition system Θ, we will be given states s of Θ, and will need to compute $h^\alpha(s)$. It is essential for this to be as efficient as possible. Say that $A = (\alpha, \Theta^\alpha)$ is the final abstraction returned by the algorithm from Figure 1. We need to (1) determine $s_\alpha := \alpha(s)$, then (2) determine $h^\alpha(s) := h^*(\Theta^\alpha, s_\alpha)$ where $h^*(\Theta^\alpha, .)$ denotes remaining cost in Θ^α. For (2), we simply use a look-up table created as part of the process constructing A. For (1), we maintain during this process what we refer to as *cascading tables* (a technique mentioned but not detailed by Dräger et al. [2006, 2009]). Each abstraction created has a corresponding look-up table, and the entries of this table are used as indices into the look-up table for the subsequent abstraction. Figure 5 illustrates.

At the start, the look-up tables correspond to atomic projections, mapping values of the respective variable domain to integers as for the package variables in Figure 5(a). Whenever we merge two abstractions A_1 and A_2, we create a combined table as for the combination of both packages in Figure 5(b), numbering the abstract states arbitrarily. Shrinking steps, as illustrated in Figure 5(c), are more subtle to handle appropriately. In principle, whenever one aggregates two abstract states s and t, one can simply find all table entries that map to t, and replace these with s. Doing a table scan, however, takes time in the size of the abstract state space each time we aggregate two states, which is very inefficient. Our implementation performs this operation in constant time, as follows. Prior to shrinking, we associate each table entry s (e.g., cell 1 in Figure 5(b)) with a linked list $L(s)$ containing the respective index pair (e.g., $\{(1, 0)\}$ for cell 1 in Figure 5(b)). Then, we discard the table and start the state aggregations. When merging s and t, we splice $L(t)$ into $L(s)$. For example, if $s = 1$ and $t = 3$ in the first aggregation step on Figure 5(b), then $L(s) = \{(0, 1)\}, L(t) = \{(1, 0)\}$ prior to the aggregation, and $L(s) = \{(0, 1), (1, 0)\}, L(t) = \emptyset$ afterwards. Once shrinking is completed, we renumber the abstract states so that there are no gaps in the numbering. Then, we regenerate the look-up table from the linked list information.

Computing $s_\alpha := \alpha(s)$ based on the cascading tables representation takes $2|\mathcal{V}| - 1$ table look-ups. To compute $h^\alpha(s) := h^*(\Theta^\alpha, s_\alpha)$, a last table is indexed by the (numbers encoding the) abstract states of the final abstraction $A = (\alpha, \Theta^\alpha)$, and maps to the solution costs of these abstract states in Θ^α. Thus, overall, computing $h^\alpha(s)$ takes $2|\mathcal{V}|$ table look-ups and $O(|\mathcal{V}|)$ runtime.

Throughout the merge-and-shrink process, we can remove unreachable and irrelevant abstract states. If an abstract state s cannot be reached from the initial state of the current abstraction, then, the abstraction being an overapproximation of the real transition system Θ, no concrete state corresponding to s can be reached in Θ either. We can thus remove s and all its incident abstract transitions from the abstract state space (and erase the corresponding look-up table entry) while still retaining admissibility of h^α on the reachable part of Θ. Similarly, if from s there is no abstract path of transitions to a solution state, then there is no such path in Θ either. We remove s from the abstract state space and replace the corresponding table entry with heuristic value ∞, retaining admissibility of h^α.

When merging abstractions, we need to compute the synchronized product of their abstract state spaces, which involves finding all pairs of states with a transition labeled by the same operator in both. To do this efficiently, we store the transitions by labels. For each abstraction $A = (\alpha, \Theta^\alpha)$ and label l, $T(\Theta^\alpha, l)$ is the list of pairs of states (s, s') in Θ^α where l labels a transition from s to s'. The transitions of the synchronized product of abstractions A_1 and A_2 can then be generated simply by pairing up all options for each l, that is, taking $T(\Theta_1^\alpha \otimes \Theta_2^\alpha, l)$ to contain all pairs $((s_1, s_2), (s_1', s_2'))$ where (s_1, s_1') in $T(\Theta_1^\alpha, l)$ and (s_2, s_2') in $T(\Theta_2^\alpha, l)$.

Finally, an important aspect of efficient implementation, at least in planning, is the maintenance of transition labels. Planning tasks often contain thousands of different labels (operators). Even in small abstractions, the techniques presented so far maintain all these labels, so that the same transitions between abstract states often are given by sizable numbers of labels. We introduce in the next section a technique reducing distinctions between labels without affecting the heuristic computed; here, notice that there is a simple but frequent special case we can catch quite easily. If a planning operator o does not mention any of the variables underlying abstraction $A = (\alpha, \Theta^\alpha)$ (more generally, if the precondition/effect domains describing a label neither impose restrictions, nor allow changes, on these variables), then the transitions labeled by o are the pairs (s, s). That is, o labels a self-loop for every state in Θ^α. In this case, we leave $T(\Theta^\alpha, o)$ empty and just flag it as "self-loop". The synchronized product operation then handles these flagged lists in the obvious manner (acting as if they contained all self-loops).

5. LABEL REDUCTION

We now discuss a technique that allows us to distinguish between fewer transition labels during the merge-and-shrink process. The technique is *conservative* in the sense that it does not change the abstractions and heuristic functions computed. It is important because (a) it reduces the time and memory requirements of merge-and-shrink construction, and (b) as we will show in Section 6, it may lead to exponentially smaller bisimulations.

Section 5.1 introduces conservative label reductions and their basic properties. Section 5.2 explains how we use this machinery in the construction of merge-and-shrink abstractions. Section 5.3 specifies how to reduce labels maximally while remaining conservative.

5.1. Conservative Label Reduction

A label reduction is similar to an abstraction mapping, but operates on transition labels instead of states. In other words, it is a function from one set of labels to another. Applying the function to a transition system gives rise to a reduced transition system. The function must respect the cost associated with transition labels, meaning it cannot map two labels with different costs to the same new label.

Definition 5.1. Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system. A *label reduction* is a function τ mapping L to a label set L^τ so that, for all $l_1, l_2 \in L$, $\tau(l_1) = \tau(l_2)$ only if $c(l_1) = c(l_2)$. We associate τ with the *reduced transition system* $\Theta|_\tau := (S, L^\tau, T|_\tau, s_0, S_\star)$ with $T|_\tau := \{(s, \tau(l), s') \mid (s, l, s') \in T\}$ and with $c(\tau(l)) = c(l)$ for $l \in L$.

The purpose of label reduction is to simplify the transition system while keeping path costs intact, thus allowing us to represent the same heuristic more compactly. The reduced transition system may contain fewer transitions than the original one, namely if $(s, l_1, s') \in T$ and $(s, l_2, s') \in T$ for $l_1 \neq l_2$ and $\tau(l_1) = \tau(l_2)$. This happens quite frequently in the application to planning; we will present some empirical data in Section 8.2. For example, the transition system associated with the perfect abstraction shown in Example 3.8 has two "parallel" transitions, for example, from state LL to $\{LT, TL\}$, corresponding to loading each of *package*-1 and *package*-2; by projecting the labels onto only those aspects of the operator description pertaining to the *truck* variable, we may simplify these into a single transition, yielding a more compact transition system that represents the same heuristic. Since label reduction does not alter the label *costs*, by itself it does not change solution cost in the transition system. Note that, if $\tau(l_1) = \tau(l_2)$ but $c(l_1) \neq c(l_2)$, the cost of the reduced label would not be well defined, so we must exclude this case.

In Example 4.2, we used the preconditions and effects of the planning operators that cause state transitions as transition labels, instead of using the operator names. (To see the difference, compare the representation of the transition system in Figure 1 with that of the same transition system in Figure 4(c).) By Proposition 2.4, we can do this, using the precondition/effect pairs that characterize transitions, for any factored transition system, not just ones that are defined in such terms. The label reductions we will use are projections of these conditions onto subsets of variables. In Section 5.3, we will show that these are the maximally coarse "conservative" label reductions, that is, label reductions one can apply while guaranteeing to not change the heuristic computed. In this section and the next, we state the conditions for this guarantee, and show how label reduction is integrated into the merge-and-shrink framework.

Label reduction can interfere with the synchronized product. If we reduce labels of two systems prior to forming their product, then the outcome may be different from what we would get by first forming the product, and then applying the reduction to it. This may happen even if the reduction applied on each side is the same. The reason is that label reduction may introduce additional combined transitions, due to pairs of transitions that did not previously have the same label, but that do after the reduction has been applied.

Example 5.2. Reconsider Figure 4, and assume that we reduce all labels to some unique symbol τ, prior to computing the synchronized product of Figure 4(a) with Figure 4(b). Then every transition from (a) will be combined with every transition from (b), resulting in an outcome vastly different from the actual synchronized product as shown in Figure 4(c). For example, by contrast to our discussion in Example 4.2, the state RLL in (c) will then have direct transitions to the states RTL and RLT, because the difference between the labels $RR_1 \to T_1$ and $LL_1 \to T_1$, as well as between $RR_2 \to T_2$ and $LL_2 \to T_2$, is lost.

Therefore, to be able to maintain the abstract transition system during the merge-and-shrink process while using label reductions, we have to impose some additional restriction on the reduction functions. The way in which we do this is to consider classes of equivalent labels.

Definition 5.3. Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system, and let $l_1, l_2 \in L$ be labels. Then l_1 and l_2 are *equivalent* in Θ iff $c(l_1) = c(l_2)$ and, for all states $s, s' \in S$,

$(s, l_1, s') \in T$ if and only if $(s, l_2, s') \in T$. A label reduction τ is *conservative* for Θ if, for all $l_1, l_2 \in L$, $\tau(l_1) = \tau(l_2)$ only if l_1 and l_2 are equivalent.

The label reduction in Example 5.2 is not conservative for the transition system from Figure 4(b), because (for example) it maps $RR_1 \to T_1$ and $LL_1 \to T_1$ to the same label, although these labels are not equivalent in this system.

The following are two simple properties we will need in our analysis:

PROPOSITION 5.4. *Let Θ be a transition system, and let α be an abstraction of Θ. If τ is a conservative label reduction for Θ, then τ is a conservative label reduction for Θ^α.*

PROPOSITION 5.5. *Let L be a set of labels, let Θ^1 and Θ^2 be transition systems using L, and let τ be a label reduction on L. If τ is conservative for both Θ^1 and Θ^2, then τ is conservative for $\Theta^1 \otimes \Theta^2$.*

The main property we need label reductions to have, to integrate seamlessly in the merge-and-shrink process, is that reduction distributes over synchronized product. For this, it suffices to be conservative for one of the component systems.

LEMMA 5.6. *Let L be a set of labels, let Θ^1 and Θ^2 be transition systems using L, and let τ be a label reduction on L. If τ is conservative for Θ^2, then $\Theta^1|_\tau \otimes \Theta^2|_\tau = (\Theta^1 \otimes \Theta^2)|_\tau$.*

PROOF SKETCH. The only interesting part here are the transitions. Obviously, $\Theta^1|_\tau \otimes \Theta^2|_\tau$ is always a super-structure of $(\Theta^1 \otimes \Theta^2)|_\tau$: by reducing labels prior to the synchronized product, we can only ever introduce more combined transitions, because we increase the sets of transitions with a common label. Conservativeness ensures the other direction, that is, that no new combined transitions are introduced. If $((s_1, s_2), r, (s_1', s_2'))$ is a transition in $\Theta^1|_\tau \otimes \Theta^2|_\tau$, then by definition, we have transitions (s_1, l_1, s_1') in Θ^1 and (s_2, l_2, s_2') in Θ^2 where $\tau(l_1) = \tau(l_2) = r$. Because τ is conservative for Θ^2, this implies that (s_2, l_1, s_2') is a transition in Θ^2, from which it follows that $((s_1, s_2), \tau(l_1), (s_1', s_2'))$ is a transition in $(\Theta^1 \otimes \Theta^2)|_\tau$, as desired. □

Of course, the lemma holds symmetrically if τ is conservative for Θ^1. However, we cannot reduce the labels in both transition systems independently. If we apply different label reductions to each of Θ^1 and Θ^2, then the outcome of the synchronized product is not necessarily preserved, even if each label reduction is conservative for one of the systems. A trivial example is that where each τ^i applied to Θ^i simply renames the labels injectively, but into disjoint images $L^{\tau^1} \cap L^{\tau^2} = \emptyset$: then $\Theta^1|_{\tau^1} \otimes \Theta^2|_{\tau^2}$ contains no transitions at all. In other cases, independent label reductions may lead to spurious transitions due to unwanted name clashes after the reduction. For example, this happens if Θ^1 and Θ^2 use no common labels originally, and τ^i renames the labels injectively, but into overlapping label sets $L^{\tau^1} \cap L^{\tau^2} \neq \emptyset$, and there exists l with $\tau^1(l) = \tau^2(l)$.

5.2. Conservative Label Reduction in the Merge and Shrink Framework

We now explain how we employ conservative label reduction in the process of constructing merge-and-shrink abstractions in such a way that we can obtain more compact abstract transition systems but without loss of heuristic information, that is, without altering path costs in the final abstract transition system. As shown by Lemma 5.6, this can be done if the reductions that we use are always conservative for one side of each merging step. We will use a family of label reductions, one for each merge that may take place throughout the merge-and-shrink process.

The intuitive idea is easiest to explain if we consider a linear merging strategy, that is, one that builds up the abstract transition system by merging the atomic projections onto variables $v_1, v_2, \ldots, v_n$ one by one into a growing composite system, abstracting the

composite in between merge steps: $(\cdots((\Theta^{\pi_{v_1}} \otimes \Theta^{\pi_{v_2}})^{\alpha_2} \otimes \Theta^{\pi_{v_3}})^{\alpha_3} \otimes \cdots \Theta^{\pi_{v_n}})$. The idea is to associate a label reduction τ_i with each variable v_i, where the image of τ_i is contained in the domain of τ_{i+1}. After merging in the ith variable, we apply the label reduction $\tau_i \circ \cdots \circ \tau_1$, and thus reduce labels more and more. In our setup, $\tau_{i+1} \circ \tau_i = \tau_{i+1}$, so the composed label reduction is always equivalent to the most recent single one. Given this, if each τ_i is conservative for the projection on variable v_{i+1}, then, by Lemma 5.6, the end result is equivalent to applying the label reductions after building the product system. Note that this condition implies that τ_i is conservative for all the remaining variables $v_{i+1}, \ldots, v_n$, because $\tau_j = \tau_j \circ \cdots \circ \tau_1$ for $i + 1 \leq j \leq n$ contains τ_i. The last reduction does not have to be conservative for any variable, so it may conflate any labels that have the same cost, resulting in a representation of the heuristic that is, in this sense, minimal.

To generalize this idea to any merging strategy, we require a family of label reduction functions, with members for each possible merging step that may occur. We need to be able to compose these reductions, and they need to be conservative.

Definition 5.7. Let Θ be a factored transition system with labels L and variables $\mathcal{V}$. A *label reduction family* for Θ is a family of reduction functions $\Gamma = \{\tau^{V_1} \mid V_1 \subseteq \mathcal{V}\}$ each of whose domains contains L, and where, for $V_1 \supseteq V_2$, the image of τ^{V_1} is contained in the domain of τ^{V_2}, and $\tau^{V_2} \circ \tau^{V_1} = \tau^{V_2}$. We say that Γ is *conservative* if, for every $V_1 \subseteq \mathcal{V}$, τ^{V_1} is conservative for $\Theta^{\pi_{V_1}}$.

As previously outlined, we will obtain such a label reduction family by representing labels as precondition/effect pairs according to Proposition 2.4, and projecting these pairs onto subsets of variables. Namely, for an intermediate merge-and-shrink abstraction α_1 over variables V', we will project the labels onto the remaining variables, $\mathcal{V} \setminus V'$. We will see here that this method, which we refer to as *label projection*, is maximal in the sense that, for each reduction τ^{V_1} in Definition 5.7 ($V_1 = \mathcal{V} \setminus V'$ in the setting just described), we cannot reduce labels more without losing conservativeness.

It remains to specify how to actually employ these label reductions during the construction of a merge-and-shrink tree. Say we wish to apply rule (M), the merging step, to two merge-and-shrink trees $\mathcal{T}^{\alpha_1}$ and $\mathcal{T}^{\alpha_2}$, for which we have previously constructed label-reduced abstract transition systems $\Theta^{\tau}_{\alpha_1}$ and $\Theta^{\tau}_{\alpha_2}$. We have already pointed out above that we cannot in general reduce labels independently, even with conservative label reductions, on both sides of a merging step (a synchronized product), without risking to alter the outcome of that merging step. As the next example shows, this is true even for the special case of label projection. More importantly, we cannot even apply label projection *within the subtrees* $\mathcal{T}^{\alpha_1}$ *and* $\mathcal{T}^{\alpha_2}$ on both sides of the merging step.

Example 5.8. Consider the planning task $(\mathcal{V}, \mathcal{O}, s_0, s_\star)$ where: $\mathcal{V} = \{x, y\}$ with $\mathcal{D}_x = \{0, 1\}$ and $\mathcal{D}_y = \{0, 1\}$; $s_0 = \{x = 0, y = 0\}$; $s_\star = \{x = 1, y = 1\}$; and the set of operators $\mathcal{O}$ consists of (using the same shorthand notation as before):

—"opX": $\emptyset \to \{x = 1\}$;
—"opY": $\emptyset \to \{y = 1\}$.

The optimal solution cost of this task is 2. Consider the reconstruction of the state space in terms of $\Theta^{\pi_x} \otimes \Theta^{\pi_y}$. Say we use label projection on each transition system, that is, we reduce labels in Θ^{π_x} by $\tau^{\{y\}}$ which projects all labels onto the variable subset $\{y\}$, and we reduce labels in Θ^{π_y} by $\tau^{\{x\}}$ which projects all labels onto the variable subset $\{x\}$. Clearly, $\tau^{\{y\}}$ is conservative for Θ^{π_y}, and $\tau^{\{x\}}$ is conservative for Θ^{π_x}—each retains all information on the respective variable. However, the synchronized product of the two label-reduced transition systems contains a spurious transition, and its solution

cost is 1. In $\Theta^{\pi_x}|_{\tau^{\{y\}}}$, opX is reduced to $\tau^{\{y\}}(\text{opX}) = \emptyset \to \emptyset$, and in $\Theta^{\pi_y}|_{\tau^{\{x\}}}$, opY is reduced to $\tau^{\{x\}}(\text{opY}) = \emptyset \to \emptyset$. Thus, in $\Theta^{\pi_x}|_{\tau^{\{y\}}} \otimes \Theta^{\pi_y}|_{\tau^{\{x\}}}$, the two operators are taken together, leading to a direct combined transition from s_0 to $s_\star$.

Say now that we change the goal to $s_\star = \{x = 0, y = 1\}$. In order to reach this state in $\Theta^{\pi_x}|_{\tau^{\{y\}}} \otimes \Theta^{\pi_y}|_{\tau^{\{x\}}}$, we would need to synchronize using opY as a transition from $y = 0$ to $y = 1$ in $\Theta^{\pi_y}|_{\tau^{\{x\}}}$, and as a self-loop on $x = 0$ in $\Theta^{\pi_x}|_{\tau^{\{y\}}}$. However, $\tau^{\{y\}}(\text{opY}) = \emptyset \to \{y = 1\}$ whereas $\tau^{\{x\}}(\text{opY}) = \emptyset \to \emptyset$. Hence, here, solution cost in the synchronized product of the two label-reduced transition systems is ∞.

Say finally that we first merge $\Theta^{\pi_x}|_{\tau^{\{y\}}}$ with some other variable u that does not interact with either of x or y (no operator that mentions u also mentions x or y), obtaining a new transition system $\Theta^\tau_{\alpha_1}$. Similarly, say we first merge $\Theta^{\pi_y}|_{\tau^{\{x\}}}$ with some other irrelevant variable v, obtaining a new transition system $\Theta^\tau_{\alpha_2}$. Clearly, if we now merge $\Theta^\tau_{\alpha_1}$ with $\Theta^\tau_{\alpha_2}$, the same issues appear.

Given Example 5.8, in order to use label projection we must decide, *prior* to starting the merge-and-shrink construction, within which of the two subtrees $\mathcal{T}^{\alpha_1}$ and $\mathcal{T}^{\alpha_2}$ of every possible merging step to reduce the labels. This can easily be accomplished by fixing a variable order (of the leaves in the merge-and-shrink tree) a priori, and projecting only "upwards" according to that order. As it turns out, one can make do with a smaller commitment that allows to choose the next variable online, during merge-and-shrink: a priori, we only select some variable v^*, which we call the *pivot variable*. During merge-and-shrink, for any merge-and-shrink tree $\mathcal{T}^\alpha$ over variables V', we project the labels in the associated transition system Θ_α iff $v^* \in V'$. (In the case of a linear merge strategy, as described previously, the role of the pivot variable is played by the first variable in the sequence.)

We are now ready to define how the label-reduced transition system associated with a merge-and-shrink abstraction α is built up inductively alongside the abstraction. The following definition mirrors Definition 4.3, only extending it by specifying which label reduction (if any) is applied at each step.

Definition 5.9. Let Θ be a factored transition system with variables $\mathcal{V}$, and let $V \subseteq \mathcal{V}$. Let Γ be a conservative label reduction family for Θ, let $\mathcal{T}^\alpha$ be a merge-and-shrink tree over V for Θ, and let $v^* \in V$. For $V_1 \subseteq V$, by $\overline{V_1}$ we denote $V \backslash V_1$. The *label-reduced transition system associated with* α, written Θ^τ_α, is constructed using these rules.

(A) If $\mathcal{T}^\alpha = \mathcal{T}_v$ for a variable $v \in \mathcal{V}$, then $\Theta^\tau_\alpha := \Theta^{\pi_{v_i}}$ if $v_i \neq v^*$, and $\Theta^\tau_\alpha := \Theta^{\pi_{v_i}}|_{\tau^{\overline{\{v_i\}}}}$ if $v_i = v^*$.
(S) If the root of $\mathcal{T}^\alpha$ has a single child $\mathcal{T}^{\alpha_2}$, and $\alpha = \alpha_1 \circ \alpha_2$, then $\Theta^\tau_\alpha := (\Theta^\tau_{\alpha_2})^{\alpha_1}$.
(M) If the root of $\mathcal{T}^\alpha$ has two children $\mathcal{T}^{\alpha_1}$ and $\mathcal{T}^{\alpha_2}$ over V_1 and V_2, then $\Theta^\tau_\alpha := \Theta^\tau_{\alpha_1} \otimes \Theta^\tau_{\alpha_2}$ if $v^* \notin V_1 \cup V_2$, and $\Theta^\tau_\alpha := (\Theta^\tau_{\alpha_1} \otimes \Theta^\tau_{\alpha_2}|_{\tau^{\overline{V_1}}})|_{\tau^{\overline{V_1 \cup V_2}}}$ if $v^* \in V_1$; symmetrically if $v^* \in V_2$.

Note that in a merging step where $v^\star \in V_1$, the reduction $\tau^{\overline{V_1}}$ has already been applied during the construction of $\Theta^\tau_{\alpha_1}$: it is applied directly to the atomic projection on $\{v^\star\}$ (rule (A)), and to any product over a variable set that includes $v^\star$ after the merging step (rule (M)). This is why we form the product as $(\Theta^\tau_{\alpha_1} \otimes \Theta^\tau_{\alpha_2}|_{\tau^{\overline{V_1}}})|_{\tau^{\overline{V_1 \cup V_2}}}$ rather than $(\Theta^\tau_{\alpha_1}|_{\tau^{\overline{V_1}}} \otimes \Theta^\tau_{\alpha_2}|_{\tau^{\overline{V_1}}})|_{\tau^{\overline{V_1 \cup V_2}}}$.

Once all variables in V have been incorporated into α, the final label reduction applied will be $\tau^\emptyset$. This function is maximally unrestricted, in the sense that any pair of labels in $\Theta^{\pi_\emptyset}$ that have the same cost are equivalent.

Example 5.10. Figure 6 corresponds to Figure 4, but under label projection, that is, the label reduction family where τ^{V_1}, for any $V_1 \subseteq \{\textit{package}\text{-}1, \textit{package}\text{-}2, \textit{truck}\}$,

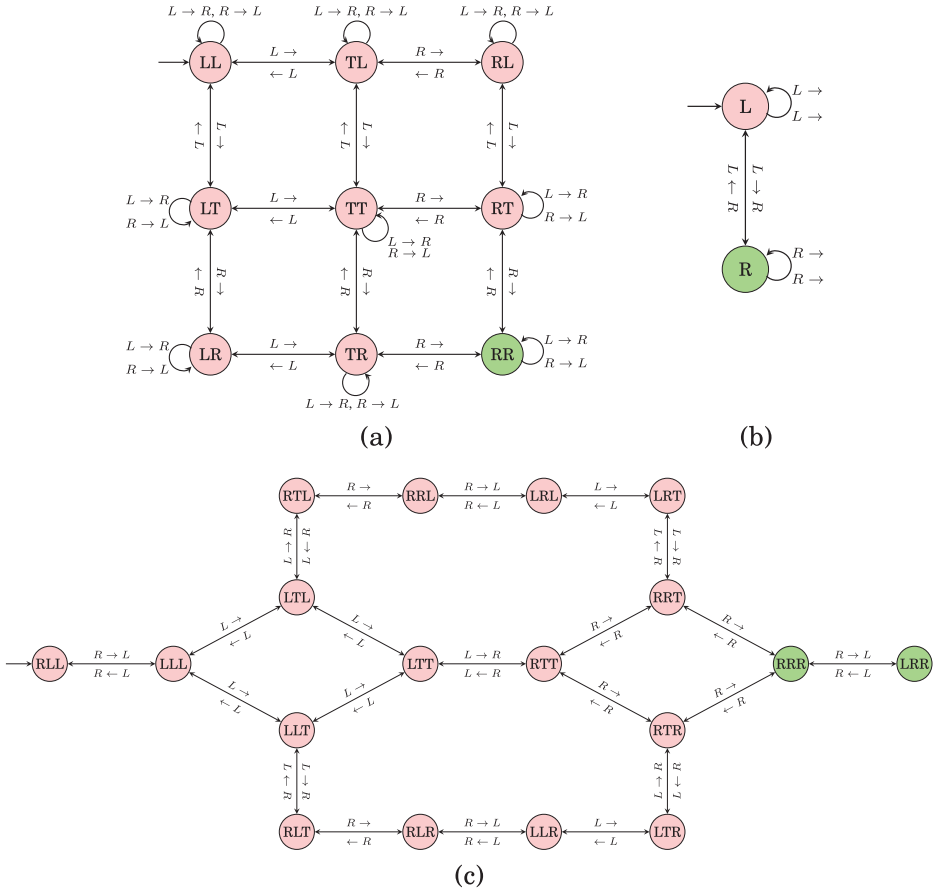


Fig. 6. The synchronized product from Figure 4, now under a label reduction that projects away all but the *truck* variable. Notations are as in Figure 4.

projects the label l onto the variables V_1. We choose the pivot variable $v^* := \mathit{package}\text{-}1$. During the construction process described by Definition 5.9, we are at the point where rule (A) has been applied for $v_1 = \mathit{package}\text{-}1$, associating this abstraction with the transition system $\Theta^{\pi_{\mathit{package}\text{-}1}}|_{\tau^{\{\mathit{package}\text{-}2,\mathit{truck}\}}}$ where $\mathit{package}\text{-}1$ is projected away from the labels. Rule (A) has also been applied for $v_2 = \mathit{package}\text{-}2$, associated with $\Theta^{\pi_{\mathit{package}\text{-}2}}$ with no label reduction. Then, rule (M) has been applied to the former two, reducing first the labels in $\Theta^{\pi_{\mathit{package}\text{-}2}}$ by $\tau^{\{\mathit{package}\text{-}2,\mathit{truck}\}}$, then computing the synchronized product, then reducing the labels further by applying $\tau^{\{\mathit{truck}\}}$. The latter projects away everything except the truck variable, and yields Figure 6(a). Further, rule (A) has been applied to $v_3 = \text{truck}$, associated with $\Theta^{\pi_{\mathit{truck}}}$. Finally, the situation we consider arises when rule (M) is being applied to (a) as well as the reduction of $\Theta^{\pi_{\mathit{truck}}}$ by $\tau^{\{\mathit{truck}\}}$, shown in (b). The outcome of the synchronized product is shown in (c). As desired, this outcome is identical to the original state space with labels reduced by $\tau^{\{\mathit{truck}\}}$.

To illustrate this, consider the product of LL in (a) with the two states R and L of (b), yielding the states RLL and LLL on the left-hand side of (c). As discussed in Example 4.2, what enables the synchronized product to assign the correct transitions is the difference between the labels $RR_1 \to T_1, RR_2 \to T_2$, as present at R, vs. the labels $LL_1 \to T_1, LL_2 \to T_2$, as present at LL. This difference is preserved under $\tau^{\{\mathit{truck}\}}$,

where these labels now read $R\rightarrow$, $R\rightarrow$ vs. $L\rightarrow$, $L\rightarrow$. The difference has simply been reduced to its essence, the truck position. Hence, we get the correct arrangement in (c), with *RLL* having a single connection to *LLL*, which by contrast has the matching label $L\rightarrow$ and gets the desired transitions to *LTL* and *LLT*.

Note that we could not have applied $\tau^{\emptyset}$, projecting away all variables, prior to the synchronized product: then, every reduced label has the same form ("$\rightarrow$", in the notation of Figure 6), bringing us back to the same situation as in Example 5.2, with many spurious transitions.

Our main result here is that the correctness just observed holds in general.

THEOREM 5.11. *Let Θ be a factored transition system with variables $\mathcal{V}$, and let $V \subseteq \mathcal{V}$. Let Γ be a conservative label reduction family for Θ, let $\mathcal{T}^\alpha$ be a merge-and-shrink tree over V for Θ. Then, $\Theta^\tau_\alpha = \Theta^\alpha|_{\tau^\emptyset}$ for any choice of pivot $v^* \in V$.*[6]

PROOF SKETCH. We prove by induction over the construction of T^α that, for any intermediate merge-and-shrink abstraction β over V': $\Theta^\tau_\beta = \Theta^\beta$ if $v^* \notin V'$, and $\Theta^\tau_\beta = \Theta^\beta|_{\tau^{\overline{V'}}}$ if $v^* \in V'$. The single tricky case in the induction is the case where $\beta = \alpha_1 \otimes \alpha_2$ and without loss of generality, $v^* \in V_1$. Using the induction hypothesis, we then need to prove that $(\Theta^{\alpha_1}|_{\tau^{\overline{V_1}}} \otimes \Theta^{\alpha_2}|_{\tau^{\overline{V_1}}})|_{\tau^{\overline{V_1 \cup V_2}}} = \Theta^{\alpha_1 \otimes \alpha_2}|_{\tau^{\overline{V_1 \cup V_2}}}$. Since $\tau^{\overline{V_1}}$ is conservative for $\Theta^{\pi_{\overline{V_1}}}$, with $V_2 \subseteq \overline{V_1}$ and Proposition 5.4, it is conservative also for Θ^{α_2}. Hence, Lemma 5.6 reduces the left-hand side of our proof obligation to $((\Theta^{\alpha_1} \otimes \Theta^{\alpha_2})|_{\tau^{\overline{V_1}}})|_{\tau^{\overline{V_1 \cup V_2}}}$, which with $\tau^{\overline{V_1 \cup V_2}} \circ \tau^{\overline{V_1}} = \tau^{\overline{V_1 \cup V_2}}$ is equal to $(\Theta^{\alpha_1} \otimes \Theta^{\alpha_2})|_{\tau^{\overline{V_1 \cup V_2}}}$. The claim then follows with Theorem 4.5. □

Thus, when conservatively reducing labels, the resulting transition system associated with α is equal to the label-reduced version of the abstract transition system of α. Since label reduction does not affect solution cost, the heuristic computed from Θ^τ_α will be the same. In particular, even if we apply label reduction, not aggregating any states results in a pattern database, and in a perfect heuristic, in case we incorporate all variables.

COROLLARY 5.12. *Let Θ be a factored transition system with variables $\mathcal{V}$, and let $V \subseteq \mathcal{V}$. Let Γ be a conservative label reduction family for Θ, let $\mathcal{T}^\alpha$ be a nonabstracting merge-and-shrink tree over V for Θ. Then $\Theta^\tau_\alpha = \Theta^{\pi_V}|_{\tau^\emptyset}$ for any choice of pivot $v^* \in V$.*

PROOF. This follows directly from Theorem 5.11 and Proposition 3.7. □

5.3. Maximal Conservative Label Reductions

To obtain a conservative label reduction family $\Gamma = \{\tau^{V_1} \mid V_1 \subseteq \mathcal{V}\}$ for an arbitrary factored transition system, we exploit the characterization of transition labels by their preconditions/effects on state variables, and project away some of the variables in those labels. Next, we show that these reductions are *maximal* among all conservative label reductions. A label reduction τ is maximal within a set of reductions if it is *at least as coarse as* all other members of the set. A label reduction τ is at least as coarse as another reduction τ' if, for all labels l_1, l_2, $\tau'(l_1) = \tau'(l_2)$ implies that $\tau(l_1) = \tau(l_2)$.

By Proposition 2.4(ii), we can characterize the transitions T of any factored transition system Θ as follows.

For each label $l \in L$, there exists a set $\mathcal{D}^{\mathrm{pe}}(l) := \{\mathcal{D}^{\mathrm{pe}}_v(l) \mid v \in \mathcal{V}\}$ where $\mathcal{D}^{\mathrm{pe}}_v(l) \subseteq \mathcal{D}_v \times \mathcal{D}_v$, so that $(s, l, s') \in T$ iff $(s(v), s'(v)) \in \mathcal{D}^{\mathrm{pe}}_v(l)$ for all v.

[6]In particular, Θ^τ_α does not depend on the choice of pivot, which justifies the fact that the chosen pivot variable is not part of the notation.

This means that we can replace each label symbol l by the *structured label* $(c(l), (\mathcal{D}_v^{\text{pe}}(l))_{v\in\mathcal{V}})$, since the two carry exactly the same information. (Note that in general, we must include also the cost, since it is a function of the label. In the special case where label costs are uniform, we can omit them from the structured label.) Given this, the *label projection* is the label reduction family defined by $\tau^{V_i}(l) := (c(l), (\mathcal{D}_v^{\text{pe}}(l))_{v\in V_i})$. In other words, we apply a projection onto the variable subset V_i to the structured label, thus mapping each label onto its cost along with the part of its description pertaining to the set of variables projected upon. In the state spaces of planning tasks, this form of label reduction is equivalent to projecting the operator descriptions, as done in Figure 6.

We now show that label projection is conservative and maximal. The basic observation is that two labels are equivalent for a variable v iff their projection onto v is the same.

PROPOSITION 5.13. *Let $\Theta = (S, L, T, s_0, S_\star)$ be a factored transition system with variables $\mathcal{V}$. Let $v \in \mathcal{V}$ be a variable, and let $l_1, l_2 \in L$. Then l_1 and l_2 are equivalent in Θ^{π_v} if and only if $c(l_1) = c(l_2)$ and $\mathcal{D}_v^{\text{pe}}(l_1) = \mathcal{D}_v^{\text{pe}}(l_2)$.*

PROOF SKETCH. The "only if" direction is trivial. The "if" direction is direct from Proposition 2.4. □

COROLLARY 5.14. *Let Θ be a factored transition system with variables $\mathcal{V}$. Then, every label projection τ^{V_1} is conservative for $\Theta^{\pi_{V_1}}$ and is maximal among all label reductions with this property.*

PROOF. By the "if" direction in Proposition 5.13, for every $v \in V_i$, τ^{V_1} is conservative for Θ^{π_v}. With Proposition 5.5, τ^{V_1} thus is conservative for $\Theta^{\pi_{V_1}}$ as desired. To see maximality, assume that l_1 and l_2 are equivalent in $\Theta^{\pi_{V_1}}$, but that $\tau^{V_1}(l_1) \neq \tau^{V_1}(l_2)$, that is, there exists $v \in V_i$ so that $\tau^{\{v\}}(l_1) \neq \tau^{\{v\}}(l_2)$. This yields a contradiction with the "only if" direction in Proposition 5.13. □

6. BISIMILARITY

Bisimilarity is a well-known condition under which aggregating states does not result in a loss of information. The present section shows how this can be exploited in merge-and-shrink to design shrinking strategies that guarantee to compute perfect heuristics. We also introduce a relaxation of bisimilarity that underlies our more practical shrinking strategies, used in some of our experiments.

Section 6.1 shows that bisimilarity integrates favorably with the merge-and-shrink framework. Section 6.2 shows that the same is true of label reduction, and in particular that this combination may reduce abstraction size exponentially. Section 6.3 discusses our relaxation of bisimilarity.

6.1. Bisimilarity and the Merge and Shrink Framework

Bisimulations, in our framework, are a particular kind of equivalence relation.

Definition 6.1. Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system. An equivalence relation $\sim$ on S is a *bisimulation* for Θ if $s \sim t$ implies that, for every transition label $l \in L$, $\{[s'] \mid (s, l, s') \in T\} = \{[t'] \mid (t, l, t') \in T\}$. We say that $\sim$ is *goal-respecting* for Θ if $\sim \subseteq \sim^G$ where $s \sim^G t$ iff either $s, t \in S_\star$ or $s, t \notin S_\star$. We say that $\sim$ is the *coarsest goal-respecting bisimulation* iff, for every goal-respecting bisimulation $\sim'$, we have $\sim' \subseteq \sim$. If α is an abstraction of Θ, then we say that α is a goal-respecting bisimulation for Θ iff $\sim^\alpha$ is.

A unique coarsest goal-respecting bisimulation always exists and can be computed efficiently based on an explicit representation of Θ [Milner 1990]. Bisimulations

essentially preserve the observable behavior of the transition system. In particular, they preserve solution cost, provided they are goal-respecting.

PROPOSITION 6.2. *Let Θ be a transition system, and let α be an abstraction of Θ. If α is a goal-respecting bisimulation for Θ, then h^α is perfect.*

PROOF. Since h^α is admissible, it suffices to show that $h^*(s) \leq h^\alpha(s)$. To see this, observe that, for every transition $([s], l, [s'])$ in $\Theta/{\sim^\alpha}$, since α is a bisimulation, we have a transition (s, l, t') in Θ so that $t' \in [s']$. Since α is goal-respecting, if $[s']$ is a solution state in $\Theta/{\sim^\alpha}$, then t' is a solution state in Θ. Thus, every solution path in $\Theta/{\sim^\alpha}$ corresponds to a solution path in Θ. □

We now essentially restate the result originally demonstrated and exploited by Sabnani et al. [1989]: if rule (S) always sticks to goal-respecting bisimilarity, then that property is preserved during merge-and-shrink. We include this result here for the sake of self-containedness, and because we will generalize it in the next subsection. Also, different from the work of Sabnani et al., our transition systems are not a priori defined as a synchronized product. We rely on Theorem 4.5 for making this correspondence. We will also need the following two simple properties.

PROPOSITION 6.3. *Let Θ be a transition system, let α be an abstraction of Θ, and let β be an abstraction of Θ^α. If α is a bisimulation for Θ and β is a bisimulation for Θ^α, then $\beta \circ \alpha$ is a bisimulation for Θ.*

PROPOSITION 6.4. *Let Θ^1 and Θ^2 be transition systems, and let α_1 and α_2 be abstractions for Θ^1 and Θ^2, respectively. If α_1 is a bisimulation for Θ^1, and α_2 is a bisimulation for Θ^2, then $\overline{\alpha_1} \otimes \overline{\alpha_2}$ is a bisimulation for $\Theta^1 \otimes \Theta^2$, where $\overline{\alpha_1}(s_1, s_2) := \alpha_1(s_1)$ and $\overline{\alpha_2}(s_1, s_2) := \alpha_2(s_2)$.*

Both propositions follow directly from inserting the respective definitions (proofs are in the appendix). It is obvious that the same properties hold for goal-respecting equivalence relations in general, and hence for goal-respecting bisimulations in particular. In the following, we will also use these variations of Proposition 6.3 and Proposition 6.4 without mentioning them specifically.

We next show that if α is a merge-and-shrink abstraction over V where rule (S) always uses a goal-respecting bisimulation, then α is a goal-respecting bisimulation for Θ^{π_V}. Note here the changed scope of α: α is a function on Θ, not on Θ^{π_V}, so strictly speaking the proved property is ill-defined. However, any merge-and-shrink abstraction α over V can be seen as the composition $\alpha = \alpha_V \circ \pi_V$ of π_V with an abstraction α_V on Θ^{π_V}.[7] Given this, henceforth, if we say that a merge-and-shrink abstraction α is a goal-respecting bisimulation for Θ^{π_V}, what we mean is that $\alpha = \alpha_V \circ \pi_V$ and α_V is a goal-respecting bisimulation for Θ^{π_V}.

COROLLARY 6.5. *Let Θ be a factored transition system with variables $\mathcal{V}$, and let $V \subseteq \mathcal{V}$. Let $\mathcal{T}^\alpha$ be a merge-and-shrink tree over V for Θ constructed so that, in each application of Definition 3.6, rule (S) α_1 is a goal-respecting bisimulation for Θ^{α_2}. Then α is a goal-respecting bisimulation for Θ^{π_V}.*

PROOF SKETCH. This property clearly holds for atomic T^α. By prerequisite and a direct application of Proposition 6.3, the property is invariant over rule (S). To see invariance over rule (M), presume that $\alpha_1 = \alpha_{1V_1} \circ \pi_{V_1}$ and $\alpha_2 = \alpha_{2V_2} \circ \pi_{V_2}$ are goal-respecting

[7]This is obvious for rule (A). For rule (S), if $\alpha_2 = \alpha_{2V} \circ \pi_V$, then $\alpha_1 \circ \alpha_2 = (\alpha_1 \circ \alpha_{2V}) \circ \pi_V$. For rule (M), if $\alpha_1 = \alpha_{1V_1} \circ \pi_{V_1}$ and $\alpha_2 = \alpha_{2V_2} \circ \pi_{V_2}$, then $\alpha_1 \otimes \alpha_2 = (\alpha_{1V_1} \circ \pi_{V_1}) \otimes (\alpha_{2V_2} \circ \pi_{V_2}) = [(\alpha_{1V_1} \circ \pi_{V_1}) \otimes (\alpha_{2V_2} \circ \pi_{V_2})] \circ \pi_{V_1 \cup V_2}$.

bisimulations for $\Theta^{\pi_{V_1}}$ and $\Theta^{\pi_{V_2}}$, respectively. By Proposition 6.4, $\overline{\alpha_{1V_1}} \otimes \overline{\alpha_{2V_2}}$ is a goal-respecting bisimulation for $\Theta^{\pi_{V_1}} \otimes \Theta^{\pi_{V_2}}$ which by Theorem 4.5 is identical with $\Theta^{\pi_{V_1 \cup V_2}}$. The claim then follows because, with $V_1 \cap V_2 = \emptyset$, we have that $\overline{\alpha_{1V_1}} = \alpha_{1V_1} \circ \pi_{V_1} = \alpha_1$ and $\overline{\alpha_{2V_2}} = \alpha_{2V_2} \circ \pi_{V_2} = \alpha_2$. □

From Corollary 6.5 and Proposition 6.2, by setting $V := \mathcal{V}$, it follows immediately that, if we always stick to goal-respecting bisimulation during step (S), then we obtain a perfect heuristic. The problem with this method is that, in practice, compact bisimulations hardly ever exist. This applies even to trivial examples.

Example 6.6. Consider the family of factored transition systems $\Theta(n) = (S(n), L(n), T(n), s_0(n), S_\star(n))$ with Boolean variables $\mathcal{V}(n) = \{v_1, \ldots, v_n\}$, with transitions labeled $l_1, \ldots, l_n$ where $c(l_i) = 1$ and l_i applies if $v_i = 0$ and sets v_i to 1, where $s_0(n)$ assigns each v_i to 0, and where $S_\star(n)$ contains the single state assigning each v_i to 1. For any two different states s, t in $S(n)$, the subset of variables with value 0 is different, and hence the sets of labels at the outgoing transitions are different. Thus, s and t are not bisimilar. The coarsest bisimulation is the identity function.

A similar situation occurs in the abstractions yielding a perfect heuristic in our running example, shown in Figure 3. For example, consider the states *TR* and *RT* in Figure 3(a). As the figure shows, both are directly connected to the solution state *RR*. What the figure does not show is that *TR* and *RT* need different labels for reaching *RR*: $RT_1 \to R_1$ for *TR*, $RT_2 \to R_2$ for *RT*. The same is true of the states *RTR* and *RRT* in Figure 3(b), requiring different labels to reach *RRR*. Thus, neither of the shown equivalence relations are bisimulations.

6.2. Combining Bisimilarity with Label Reduction

Bisimilarity is too strict for our purposes. It was conceived to preserve the observable behavior of transition systems. By contrast, all we wish to preserve is solution cost. Thus, we do not care about the transition labels. In other words, we can combine bisimilarity with label reduction.

PROPOSITION 6.7. *Let Θ be a transition system, let τ be a label reduction, and let α be an abstraction of Θ. If α is a goal-respecting bisimulation for $\Theta|_\tau$, then h^α is perfect.*

PROOF. As in the proof to Proposition 6.2, it is easy to show that every solution path in $(\Theta|_\tau)/{\sim^\alpha}$ corresponds to a solution path in $\Theta|_\tau$. Since $c(\tau(l)) = c(l)$ for all labels l, the costs of these paths is the same as in Θ. □

This simple result is important because, after label reduction, the bisimulation may be exponentially more compact.

PROPOSITION 6.8. *There exist families of transition systems Θ with associated label reductions τ so that the coarsest goal-respecting bisimulation for $\Theta|_\tau$ is exponentially smaller than the coarsest goal-respecting bisimulation for Θ.*

PROOF SKETCH. Consider the family $\Theta(n) = (S(n), L(n), T(n), s_0(n), S_\star(n))$ of factored transition systems from Example 6.6. As argued there, the coarsest goal-respecting bisimulation for $\Theta(n)$ is the identity function, size 2^n. On the other hand, let $\tau(n)$ be a label reduction mapping every label to the same unique symbol. Let $\alpha(n)$ be an abstraction that aggregates s and t iff they agree on the count of variables already set to 1. Trivially, $\alpha(n)$ is goal-respecting for $\Theta(n)|_{\tau(n)}$. It is also a bisimulation for $\Theta(n)|_{\tau(n)}$, because whenever we can set a variable v_j to 1 in s, we can set some (potentially different) variable v_k to 1 in t; the labels l_j and l_k are not distinguished under $\tau(n)$. The size of $\alpha(n)$ is $n + 1$, proving the claim. □

Considering again Figure 3, it is easy to verify that the shown equivalence relations are bisimulations when ignoring the labels, that is, when reducing them all to the same unique symbol. This form of reduction can be done for an arbitrary number of packages, and hence our illustrative example also proves Proposition 6.8.

The question now remains how to combine bisimilarity with label reduction within the merge-and-shrink framework. As pointed out in Section 5, we cannot remove all labels to begin with, because they are needed for correctly computing the synchronized product in merging steps. We now show that our solution to this issue, conservative label reduction, combines gracefully with bisimilarity.

Definition 6.9. Let Θ be a factored transition system with variables $\mathcal{V}$, and let $V \subseteq \mathcal{V}$. Let Γ be a label reduction family for Θ, let $\mathcal{T}^\alpha$ be a merge-and-shrink tree over V for Θ, and let $v^* \in V$. We say that T^α is *constructed by label-reduced goal-respecting bisimulations* if, in each application of Definition 3.6 rule (S) where T^{α_2} is over the variables V_2, the following holds: if $v^* \notin V_2$, then α_1 is a goal-respecting bisimulation for Θ^{α_2}; if $v^* \in V_2$, then α_1 is a goal-respecting bisimulation for $\Theta^{\alpha_2}|_{\tau^{\overline{V_2}}}$.

THEOREM 6.10. *Let Θ be a factored transition system with variables $\mathcal{V}$, and let $V \subseteq \mathcal{V}$. Let Γ be a conservative label reduction family for Θ, and let $\mathcal{T}^\alpha$ be a merge-and-shrink tree over V for Θ. If $\mathcal{T}^\alpha$ is constructed by label-reduced goal-respecting bisimulations, then α is a goal-respecting bisimulation for $\Theta^{\pi_V}|_{\tau^\emptyset}$.*

PROOF SKETCH. Let v^* be the pivot variable in the construction of $\mathcal{T}^\alpha$. We prove by induction over the construction of $\mathcal{T}^\alpha$ that, for any intermediate merge-and-shrink abstraction β over V': β is a goal-respecting bisimulation for $\Theta^{\pi_{V'}}$ if $v^* \notin V'$, and β is a goal-respecting bisimulation for $\Theta^{\pi_{V'}}|_{\tau^{\overline{V'}}}$ if $v^* \in V'$. The claim follows by setting $\beta = \alpha$.

The base case, rule (A) of Definition 3.6, is trivial. Induction over rule (S) is a direct consequence of Proposition 6.3, as in Corollary 6.5. For induction over rule (M), say that $\beta = \alpha_1 \otimes \alpha_2$, where V_1 and V_2 are the respective sets of variables. The case $v^* \notin V_1 \cup V_2$ works exactly as in Corollary 6.5. For the remaining case where, without loss of generality, $v^* \in V_1$, we apply the trivial observation that label reduction can only make bisimilarity easier to achieve, thus it suffices to prove that $\alpha_1 \otimes \alpha_2$ is a goal-respecting bisimulation for $\Theta^{\pi_{V_1 \cup V_2}}|_{\tau^{\overline{V_1}}}$ (rather than for $\Theta^{\pi_{V_1 \cup V_2}}|_{\tau^{\overline{V_1 \cup V_2}}}$). We apply Theorem 4.5 to see that $\Theta^{\pi_{V_1 \cup V_2}}|_{\tau^{\overline{V_1}}} = (\Theta^{\pi_{V_1}} \otimes \Theta^{\pi_{V_2}})|_{\tau^{\overline{V_1}}}$. We apply Proposition 5.5 and Lemma 5.6 to see that $(\Theta^{\pi_{V_1}} \otimes \Theta^{\pi_{V_2}})|_{\tau^{\overline{V_1}}} = \Theta^{\pi_{V_1}}|_{\tau^{\overline{V_1}}} \otimes \Theta^{\pi_{V_2}}|_{\tau^{\overline{V_1}}}$. Thus, it suffices to prove that $\alpha_1 \otimes \alpha_2$ is a goal-respecting bisimulation for $\Theta^{\pi_{V_1}}|_{\tau^{\overline{V_1}}} \otimes \Theta^{\pi_{V_2}}|_{\tau^{\overline{V_1}}}$. From here, the claim follows with Proposition 6.4, like in the proof to Corollary 6.5. □

The equivalence relations from Figure 3, and thus a compactly represented perfect heuristic for our illustrative example, can be constructed by label-reduced goal-respecting bisimulations.

Example 6.11. Consider first the abstraction α_1 on the left-hand side of Figure 7. That abstraction results from a shrinking step applied to $\pi_{package\text{-}1} \otimes \pi_{package\text{-}2}$. In line with Definition 6.9, the shrinking step is a goal-respecting bisimulation of the transition system associated with $\pi_{package\text{-}1} \otimes \pi_{package\text{-}2}$. In line with Definition 5.9, this transition system has reduced labels. More precisely, the construction is as in Example 5.10, using label projection applied directly to the planning-based operator descriptions. Rule (A) is applied to $v_1 = package\text{-}1$ and $v_2 = package\text{-}2$. Thereafter, rule (M) is applied by first reducing the labels in $\Theta^{\pi_{package\text{-}2}}$, projecting away *package*-1; then computing the synchronized product; then reducing the labels further by projecting away also *package*-2. The resulting system is equal to $\Theta^{\pi_{package\text{-}1} \otimes \pi_{package\text{-}2}}|_{\tau^{\{truck\}}}$. The abstraction α_1 in Figure 7 (left) is the coarsest goal-respecting bisimulation for that system. To see

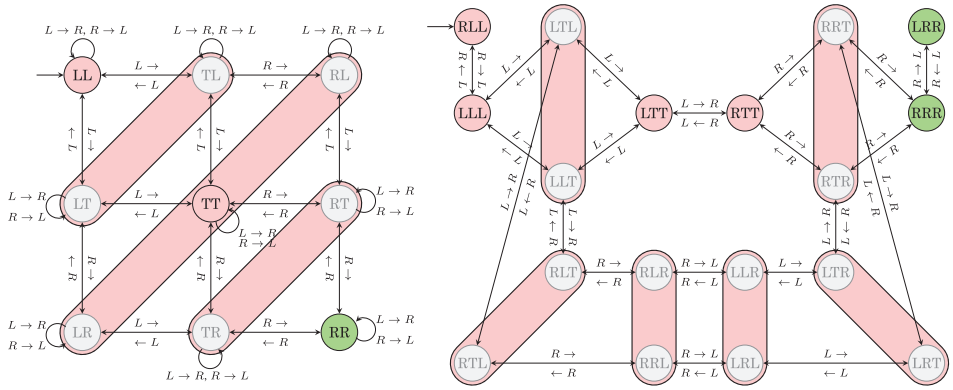


Fig. 7. Merge-and-shrink abstractions constructed by label-reduced goal-respecting bisimulations, in our illustrative example. The abstractions are shown in terms of equivalence relations, as in Figure 3, and the underlying labeled transition systems are notated as in Figures 4 and 6. Left-hand side: a merge-and-shrink abstraction α_1 constructed as the coarsest goal-respecting bisimulation of $\Theta^{\pi_{package\text{-}1} \otimes \pi_{package\text{-}2}}$ with labels projected onto the *truck* variable. Right-hand side: $\alpha_1 \otimes \pi_{truck}$, yielding a perfect heuristic for the original state space.

this, just note that the states within each equivalence class can reach the same other equivalence classes using the same (reduced) labels, and that this property is not preserved when enlarging any of the equivalence classes.

The abstraction on the right-hand side of Figure 7 is exactly what we get when we merge α_1 with π_{truck}. Thus, as desired, the abstraction of Figure 7 (right) can be constructed by label-reduced goal-respecting bisimulations. Since the underlying transition system is equal to $\Theta|_{\tau\{truck\}}$, and solution costs in $\Theta|_{\tau\{truck\}}$ are the same as those in the original state space Θ, the induced heuristic is perfect.

It is easy to see that this works for an arbitrary number of packages, that is, label-reduced goal-respecting bisimulations yield the same abstractions as discussed in Example 3.8, not unnecessarily distinguishing between permutations of packages. Thus, we can obtain a perfect heuristic in polynomial time.

With Theorem 6.10 and Proposition 6.7, if α is constructed by label-reduced goal-respecting bisimulations under the specified conditions, then the resulting lower bound is exact. With Theorem 5.11 and Corollary 5.14, we know how to compute the associated label-reduced abstract transition systems with the merge-and-shrink algorithm. Together with the previously known methods for computing coarsest goal-respecting bisimulations, this gives us an algorithmic method (a concrete shrinking strategy) for computing perfect lower bounds. We refer to this shrinking strategy as *Bisim-LR*: "Bisim" for bisimulation, "LR" for label reduction.

By Proposition 6.8, and as can be observed in our illustrative example, the abstractions computed by Bisim-LR can be exponentially smaller than when using bisimulations without label reduction. We will see in Section 8.1 that this happens also in a number of commonly used planning benchmarks.

6.3. Greedy Bisimilarity

The previous results notwithstanding, Bisim-LR is much too expensive—does not sufficiently shrink the abstractions—on most planning benchmarks. This is only natural since Bisim-LR guarantees a perfect heuristic, thus solving the planning problem. In practice, we need more approximate shrinking strategies that trade off heuristic

accuracy against the effort required for building the heuristic. Our key concept for defining such strategies is the following relaxed version of bisimilarity.

Definition 6.12. Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system. An equivalence relation $\sim$ on S is a *greedy bisimulation* for Θ if it is a bisimulation for the system $(S, L, T^G, s_0, S_\star)$ where $T^G = \{(s, l, s') \mid (s, l, s') \in T, h^*(s) = h^*(s') + c(l)\}$.

In other words, when deciding whether two states are deemed to be equivalent, greedy bisimulations consider only the transitions that are used in an optimal solution for some state in the transition system at hand.[8] Since greedy bisimulations are bisimulations on a modified transition system, it follows trivially that coarsest greedy bisimulations exist, are unique, and can be computed exactly like coarsest bisimulations. More interestingly, greedy bisimilarity is what we call *h-preserving*: it still preserves solution cost within the transition system to which it is being applied.

PROPOSITION 6.13. *Let Θ be a transition system, let τ be a label reduction, and let α be an abstraction of Θ. If α is a goal-respecting greedy bisimulation for $\Theta|_\tau$, then h^α is perfect.*

PROOF. Let Θ^G be the transition system identical to Θ except that it contains only the transitions (s, l, s') where $h^*(s) = h^*(s') + c(l)$. Clearly, solution cost in $\Theta^G|_\tau$ is the same as that in Θ: Θ^G contains all transitions participating in optimal solutions in Θ. Now let h' be the heuristic function that results from applying α to $\Theta^G|_\tau$. By Proposition 6.7, h' is equal to solution cost in $\Theta^G|_\tau$. Thus, it suffices to show that $h' = h^\alpha$. The difficulty in doing so is that h^α, while using the same abstraction function α, is based on the original transition system Θ which contains more transitions than Θ^G, potentially causing $h^\alpha < h'$. However, the latter cannot happen. Observe that h^α can be obtained by starting from the quotient system $\Theta^G|_\tau/{\sim^\alpha}$ underlying h', and adding the abstract transitions corresponding to $(s, l, s') \in T$ where $h^*(s) > h^*(s') + c(l)$. Since solution costs in $\Theta^G|_\tau/{\sim^\alpha}$ are equal to those in $\Theta^G|_\tau$ and thus to h^*, these transitions do not change h'. □

The shrinking strategy we will use is like Bisim-LR, except that it uses greedy bisimilarity. We refer to this as *GBisim-LR*. Note that Proposition 6.13 does not imply that GBisim-LR guarantees a perfect heuristic. While greedy bisimilarity is *h*-preserving and thus preserves solution cost "locally" within each abstraction, it does not preserve solution cost globally, because it is not invariant over merging steps. In other words, an equivalent of Proposition 6.4 does not hold. This is because transitions optimal in the merged system $\Theta_1 \otimes \Theta_2$ might not be optimal in one of the component systems, Θ_1 or Θ_2. A simple example is one where Θ_1 represents a truck, Θ_2 represents a package, and the transition drives the truck away from its own goal—which globally can be a good idea in order to transport the package.

7. EXPRESSIVE POWER

We now proceed to the evaluation of our approach. In the present section, we evaluate the merge-and-shrink framework from a theoretical perspective, comparing its "expressive power" to that of other known frameworks. Positive results state what the approach can accomplish, provided we choose a good abstraction strategy. Negative results state what the approach cannot accomplish, independently of the abstraction strategy we

[8]We remark that the experiments in our previous conference paper [Nissim et al. 2011] used a slightly different notion of "greedy bisimilarity", namely considering the transitions where $h^*(s') > h^*(s)$. The present definition is more accurate with respect to: (a) optimal solutions in Θ ($h^*(s') > h^*(s)$ does not imply that the transition participates in such a solution); and (b) 0-cost labels (which participate in an optimal solution iff $h^*(s') = h^*(s)$).

choose. In that sense, we quantify over abstraction strategies, and in particular we will not refer to the label reduction and bisimilarity techniques just discussed. This will change in the next section where we evaluate merge-and-shrink by its concrete application in AI planning.

Section 7.1 shows that merge-and-shrink can capture additive heuristic ensembles within a single heuristic. Section 7.2 shows that merge-and-shrink strictly generalizes pattern databases. Section 7.3 argues that merge-and-shrink heuristics allow to efficiently construct corresponding ADDs. Section 7.4 establishes a relationship between merge-and-shrink trees and model counting problems. Section 7.5 exploits this connection to show a negative result regarding the relationship between merge-and-shrink and another heuristic construction principle originating in AI planning, thus partially closing an open question posed by Helmert and Domshlak [2009].

7.1. Additive Ensembles of Heuristics

A key technique to improve admissible heuristics is to combine them. If $h_1, \ldots, h_k$ are admissible heuristics, then their pointwise maximum is an admissible heuristic dominating each individual heuristic. Under certain conditions, their pointwise sum, which dominates the maximum, is also admissible. Many recent advances in the accuracy of admissible planning heuristics are due to better, more fine-grained methods for finding admissible additive heuristics.

Katz and Domshlak [2008] introduced a very general criterion for admissible additive combinations. Let $\Theta = (S, L, T, s_0, S_\star)$ be a factored transition system. Let $\Theta_1, \ldots, \Theta_k$ be factored transition systems that are identical to Θ except for the label costs. Let $c_i : L \to \mathbb{R}_0^+$ denote the label cost function for Θ_i; let c be the original label cost function in Θ. We say that $c_1, \ldots, c_k$ is a *cost partitioning* for Θ if $\sum_{i=1}^k c_i \leq c$ (where the sum and $\leq$ operators are applied pointwise). The sum of arbitrary admissible heuristics for Θ_i is then an admissible heuristic for Θ.

Cost partitioning offers a very flexible means to additively combine different heuristic estimates in an admissible way. In particular, it subsumes earlier additivity criteria for pattern database heuristics (e.g., Edelkamp [2001] and Felner et al. [2004]) and for admissible heuristics in general [Haslum et al. 2005]. These criteria correspond to the special case where $c_i(l) \in \{0, c(l)\}$, that is, they require to account for the cost of each label in at most one of the heuristics, rather than allowing to distribute that cost arbitrarily across all heuristics.

We refer to an ensemble of abstractions $\alpha_1, \ldots, \alpha_k$ together with a cost partitioning $c_1, \ldots, c_k$ as an *additive ensemble* of heuristics. Given a factored transition system Θ, an abstraction α of Θ, and a modified label cost function c' for Θ, we write $h^{\alpha c'}$ for the heuristic induced by α when assuming label costs c'.

Our observation here is that cost partitioning is essentially not needed in the merge-and-shrink framework: additive ensembles can be captured within a single merge-and-shrink heuristic. This is easy to see for ensembles consisting of only two heuristics.

PROPOSITION 7.1. *Let $\Theta = (S, L, T, s_0, S_\star)$ be a factored transition system, let α_1 and α_2 be merge-and-shrink abstractions of Θ over disjoint variable sets V_1 and V_2, and let c_1, c_2 be a cost partitioning for Θ. Then, one can in time bounded by a polynomial in $|\alpha_1|$ and $|\alpha_2|$ construct a merge-and-shrink abstraction α_{12} with cost function $c_{12} \leq c$ so that $h^{\alpha_{12} c_{12}} \geq h^{\alpha_1 c_1} + h^{\alpha_2 c_2}$ and $|\alpha_{12}| = |\{h^{\alpha_{12} c_{12}}(s) \mid s \in S\}|$.*

PROOF SKETCH. We define $\overline{\alpha_{12}} := \alpha_1 \otimes \alpha_2$ and associate this new abstraction with the cost function $c_{12} := c_1 + c_2$. We have $c_{12} \leq c$ because c_1, c_2 is a cost partitioning. We have $h^{\overline{\alpha_{12}} c_{12}} \geq h^{\alpha_1 c_1} + h^{\alpha_2 c_2}$ because solution paths in $\Theta^{\overline{\alpha_{12}}}$ contain solution paths for Θ^{α_1} and Θ^{α_2}, dominating their summed up cost by construction of c_{12}. The desired abstraction

α_{12} is obtained from $\overline{\alpha_{12}}$ by aggregating all (abstract) states with identical solution cost. □

The fact that the size of α_{12} is equal to the number of different heuristic values—obtained simply by aggregating abstract states having the same value, that is, applying the maximal h-preserving abstraction—is not essential for Proposition 7.1 itself. It is important when considering arbitrarily large additive ensembles, to keep the overhead polynomial. For that, it suffices to combine this shrinking strategy with one additional aggregation step.

THEOREM 7.2. *Let $\Theta = (S, L, T, s_0, S_\star)$ be a factored transition system, let $\alpha_1, \ldots, \alpha_k$ be merge-and-shrink abstractions of Θ over pairwise disjoint variable sets $V_1, \ldots, V_k$, and let $c_1, \ldots, c_k$ be a cost partitioning for Θ. Then, for each $2 \leq j \leq k$, there exists a merge-and-shrink abstraction $\alpha_{1\ldots j}$ with cost function $c_{1\ldots j} \leq c$ so that $h^{\alpha_{1\ldots j}c_{1\ldots j}} \geq \sum_{i=1}^{j} h^{\alpha_i c_i}$, $|\alpha_{1\ldots j}| = |\{h^{\alpha_{1\ldots j}c_{1\ldots j}}(s) \mid s \in S\}|$, where $h^{\alpha_{1\ldots j}c_{1\ldots j}}(s) \leq \sum_{i=1}^{j} \max_{s\in S} h^{\alpha_i c_i}(s)$ for all $s \in S$, and $\alpha_{1\ldots j}$ can be constructed in time bounded by a polynomial in $|\alpha_1|, \ldots, |\alpha_j|, |\alpha_{12}|, \ldots, |\alpha_{1\ldots(j-1)}|$.*

PROOF SKETCH. We simply iterate the construction underlying Proposition 7.1, merging $\alpha_{1\ldots(j-1)}$ with α_j to obtain $\overline{\alpha_{1\ldots j}}$ with cost function $c_{1\ldots j} := \sum_{i=1}^{j} c_i$, then aggregating identical-cost abstract states in $\overline{\alpha_{1\ldots j}}$. We then apply an additional shrinking step, obtaining $\alpha_{1\ldots j}$ by aggregating all abstract states whose solution cost is at least $\sum_{i=1}^{j} \max_{s\in S} h^{\alpha_i c_i}(s)$. It is easy to see that this will satisfy the claims regarding $h^{\alpha_{1\ldots j}c_{1\ldots j}}$ and $|\alpha_{1\ldots j}|$. The construction of $\alpha_{1\ldots j}$ takes time polynomial in $|\alpha_{1\ldots(j-1)}|$ and $|\alpha_j|$, and does otherwise not depend on the size of the synchronized product of the original abstractions $\alpha_1, \ldots, \alpha_{j-1}$. □

The only entity with worst-case exponential growth in this construction is the number of different solution costs within the abstraction $\alpha_{1\ldots k}$. This growth has two possible sources. First, when merging k abstractions, solution cost may grow exponentially in k. Our construction avoids this source of complexity by aggregating any states exceeding the sum of individual maximal costs $\max_{s\in S} h^{\alpha_i c_i}(s)$. Second, the number of different sums of costs $\alpha_i(s)$ may be exponential in k. For integer cost functions, such behavior is pathological: the original label costs themselves would have to be exponentially big.

COROLLARY 7.3. *Let $\Theta = (S, L, T, s_0, S_\star)$ be a factored transition system with integer label costs, let $\alpha_1, \ldots, \alpha_k$ be merge-and-shrink abstractions of Θ over pairwise disjoint variable sets $V_1, \ldots, V_k$, and let $c_1, \ldots, c_k$ be a cost partitioning for Θ where all cost functions are integer-valued. Then, one can in time bounded by a polynomial in $|\alpha_1|, \ldots, |\alpha_k|$ and $\max_{l\in L} c(l)$ construct a merge-and-shrink abstraction $\alpha_{1\ldots k}$ with cost function $c_{1\ldots k} \leq c$ so that $h^{\alpha_{1\ldots k}c_{1\ldots k}} \geq \sum_{i=1}^{k} h^{\alpha_i c_i}$.*

PROOF. This is a direct application of Theorem 7.2. (Note that all finite heuristic values $h^{\alpha_i c_i}$ are bounded by $|\alpha_i| \cdot \max_{l\in L} c(l)$, so in the absence of fractional costs, the number of different solution costs is polynomially bounded in the given quantities.) □

As stated, additive ensembles are a key technology for deriving strong admissible heuristics. Integer label costs are often not a very limiting assumption unless the set of relevant different costs is too large to be discretized. Thus, Corollary 7.3 is a strong result in favor of the merge-and-shrink framework.

The construction underlying Corollary 7.3 (along with Proposition 7.1 and Theorem 7.2) can be optimized by shrinking α_j before merging it with $\alpha_{1\ldots(j-1)}$. Similarly, as done after the merge for $\alpha_{1\ldots j}$, before the merge we can aggregate all α_j abstract states with identical solution cost and still obtain the same guarantees.

An analogous result can be shown also without the condition that $\alpha_1, \ldots, \alpha_k$ be pairwise orthogonal. The abstraction $\alpha_{1\ldots k}$ is then no longer a merge-and-shrink abstraction according to our Definition 3.6. However, recall that orthogonality is required only to correctly maintain the abstract state spaces by using the synchronized product: without orthogonality, that product may include additional states and transitions. Yet, $\alpha_{1\ldots k}$ is still an abstraction, and optimal path cost in the constructed transition system is still an admissible heuristic for path cost in the original (cf. Section 3.1). Hence, we can *define* the abstraction heuristic as solution cost in the resulting transition system. It is easy to see that, this way, we still obtain heuristics satisfying the claims just stated.

7.2. Pattern Databases

Proposition 3.7 and Corollary 7.3 together show that any additive ensemble of PDB heuristics can (for polynomially bounded integer label costs) with polynomial overhead be encoded as a merge-and-shrink heuristic. One simply simulates each PDB by a nonabstracting merge-and-shrink abstraction (Proposition 3.7), then combines this ensemble into a single merge-and-shrink heuristic (Corollary 7.3). Thus, merge-and-shrink generalizes additive PDBs. The generalization is strict, in the sense that there exist cases where merge-and-shrink can compactly represent the perfect heuristic and additive PDBs cannot.

THEOREM 7.4. *There exist families of transition systems where (a) there exists an abstraction strategy for which merge-and-shrink yields a perfect heuristic in polynomial time, whereas (b) there exists no ensemble of polynomial-size projections $\pi_{V_1}, \ldots, \pi_{V_k}$ with cost partitioning $c_1, \ldots, c_k$ which can guarantee that $\sum_{i=1}^{k} h^{\pi_{V_i} c_i}(s_0) \geq (\frac{1}{2} + \varepsilon) h^*(s_0)$, for any $\varepsilon > 0$.*

PROOF SKETCH. It suffices to extend Example 6.6 by introducing a variable v_0 that must be set to 1, using label l_0 with cost 1, in between the moves on any of the variables $v_1, \ldots, v_n$. Then, l_0 contributes half of the cost of an optimal solution. A polynomial-sized merge-and-shrink abstraction simply aggregates s and t iff they agree on v_0 as well as the count of variables $j > 0$ already set to 1. Any ensemble $\pi_{V_1}, \ldots, \pi_{V_k}$ with cost partitioning $c_1, \ldots, c_k$, however, accounts only for a logarithmic number of v_0 moves: for any π_{V_i}, the cost pertaining to v_0 in $h^{\pi_{V_i} c_i}(s_0)$ is bounded from above by $(|V_i| + 1) \cdot c_i(l_0)$; since $|\pi_{V_i}| = 2^{|V_i|}$ is polynomial in n this bound becomes $(A_i + B_i \log n) \cdot c_i(l_0)$, for some constants A_i and B_i; summing this up we get $\sum_{i=1}^{k} (A_i + B_i \log n) \cdot c_i(l_0) \leq A + B \log n$, for some constants A and B, because $c_1, \ldots, c_k$ is a cost partitioning. □

As a special case of Corollary 7.3 and Theorem 7.4, merge-and-shrink strictly generalizes individual PDB heuristics. Note that, since it considers only a single PDB, this special case does not require the additional prerequisite of Corollary 7.3 (polynomially bounded integer label costs) needed to polynomially simulate PDB ensembles. Note also that Theorem 7.4 does not require this prerequisite at all. The theorem holds for arbitrary additive ensembles of PDBs, regardless whether they are pairwise orthogonal or not, regardless of the cost partitioning used, and regardless of the number k of patterns (even exponentially large k would not help). Given the previously cited practical importance of PDB heuristics [Korf 1997; Culberson and Schaeffer 1998; Korf and Zhang 2000; Edelkamp 2001; Felner et al. 2004], these are quite remarkable results.

7.3. ADDs

Merge-and-shrink abstractions α are recursively defined mappings from states to abstract states. The endpoint of the mapping—the abstract state—is associated with the heuristic value. The abstraction mapping is represented by cascading tables, as explained in Section 4.3. In this and the next section, we show two results about the

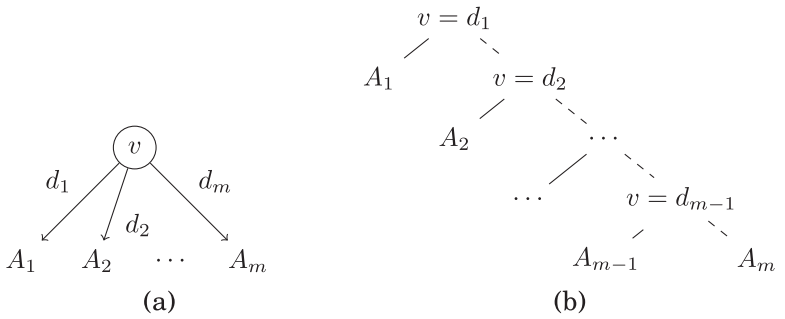


Fig. 8. Translation from an ADD with finite-domain variables (a) to a standard, binary branching ADD (b). In (b), "yes" branches are drawn with solid edges, "no" branches with dashed edges.

power of this representation to compactly describe functions and support efficient operations. In particular, we contrast it with function representations based on (ordered) decision diagrams, known as ADDs [Bahar et al. 1993]. First, we will show that a subclass of merge-and-shrink abstractions, namely those built according to a linear merging strategy, can be efficiently translated into an ADD of size polynomial in that of the merge-and-shrink representation. We proceed in Section 7.4 by showing that for any merge-and-shrink abstraction, as long as the merges are over orthogonal abstractions, our representation of the abstraction function supports polynomial-time model counting, that is, counting the number of states that the abstraction heuristic maps to a given value. (This generalizes a similar property for ADDs.) In Section 7.5, we will use this to prove a negative result regarding merge-and-shrink's expressive power.

Let $\mathcal{T}^\alpha$ be a merge-and-shrink tree. We explain the construction of an ADD A^α that represents α in two steps: First, we use a "generalized" form of ADDs with finite-domain variables at the inner vertices (i.e., nonbinary branching); then we translate each such node into a small network of binary branching nodes.

If $\alpha = \pi_v$ is atomic, the ADD has a single inner vertex v, with one branch for each value in $\mathcal{D}_v$. For a merged abstraction, $\alpha = \alpha_1 \otimes \alpha_2$, we attach a copy of the ADD for α_2 to each leaf of the ADD for α_1 and adjust the leaves of the composite ADD to represent the combined values. For $\alpha = \alpha_1 \circ \alpha_2$, we redirect the leaves of the ADD for α_2 to leaves representing the values of α_1.

To replace a finite-domain variable v, with $\mathcal{D}_v = \{d_1, \ldots, d_m\}$, in the ADD with Boolean variables, it is sufficient to construct a "linear" tree with $|\mathcal{D}_v| - 1$ interior nodes. Each node corresponds to the choice $v = d_i$ or $v \neq d_i$. At the last node, since all but the last two values have been ruled out, the two branches point to $v = d_{m-1}$ and $v = d_m$. Figure 8 illustrates the construction.

PROPOSITION 7.5. *Let $\mathcal{T}^\alpha$ be a merge-and-shrink tree built according to a linear merge strategy, with variable order $v_1, \ldots, v_n$. The ADD constructed according to the scheme described in the preceding paragraphs, with the same variable order, is of size $O(|V||\mathcal{T}|^{\max}|\mathcal{D}_v|)$.*

PROOF. First, we consider the generalized, finite-domain, ADD. Consider the abstraction α_i built by merging $v_1, \ldots, v_i$ (and intermediate shrink steps), where $i < |V|$. This abstraction has at most $|\mathcal{T}|^{\max}/|\mathcal{D}_{v_{i+1}}|$ states, because the abstraction that results from merging it with the atomic abstraction $\pi_{v_{i+1}}$ must be of size at most $|\mathcal{T}|^{\max}$. Hence, the corresponding ADD has at most $|\mathcal{T}|^{\max}/|\mathcal{D}_{v_{i+1}}|$ leaves. The ADD corresponding to $\pi_{v_{i+1}}$ has one interior node and $|\mathcal{D}_{v_{i+1}}|$ leaves. Thus, the ADD for the merged abstraction, $\alpha_i \oplus \pi_{v_{i+1}}$, adds at most $|\mathcal{T}|^{\max}$ nodes to the ADD for α_i. If the merge step is followed by a shrink step, then this cannot increase the size of the ADD in the construction because

shrink steps do not lead to new ADD nodes. Thus, the size of the complete ADD is at most $|V||\mathcal{T}|^{\max}$. Transforming the finite-domain ADD into one with binary branching at the interior nodes increases the size by at most a factor $|\mathcal{D}_v|$. □

Having shown that merge-and-shrink trees with linear merge strategies can be efficiently converted to ADDs, a natural question is whether the restriction to linear merge strategies is necessary. We conjecture that this is indeed the case, and more specifically, that there exists a family $\mathcal{T}_n$ of (nonlinear) merge-and-shrink trees over n variables such that the smallest ADD representation of $\mathcal{T}_n$ is of size $\Omega((|\mathcal{T}_n|^{\max})^{c \log n})$ for some constant c. Furthermore, we conjecture that a bound of this form is tight and that the $\log n$ factor in the exponent can be more accurately expressed with the *Horton-Strahler number* of the merge-and-shrink tree, which is a measure of "bushiness" for trees that ranges between 1 for degenerate trees (chains) and $\lceil \log_2 n \rceil$ for complete binary trees [Horton 1945].

7.4. Model Counting

Let $\mathcal{T}^\alpha$ be a merge-and-shrink tree over a set of variables V, representing abstraction α. Let $N_{\alpha,V}(s)$ denote the number of complete assignments over V that α maps to abstract state s. In this section, we will show that the merge-and-shrink tree representation can be used to compute $N_{\alpha,V}(s)$, for all abstract states s, in time and space that is polynomial in the size of $\mathcal{T}^\alpha$. This is a property that merge-and-shrink trees share with several other structured function representations, such as free binary decision diagrams (FBDDs) and formulas in deterministic decomposable negation normal form (d-DNNF) [Darwiche and Marquis 2002].

Note that the following proof relies on the restriction, postulated by Definition 3.6, that every merging node merges abstractions over disjoint variable sets. Without this restriction, it would be easy to provide a parsimonious reduction from the model-counting problem for propositional logic formulas in conjunctive normal form, and hence computing $N_{\alpha,V}(s)$ would be **#P**-hard.

THEOREM 7.6. *From a merge-and-shrink tree $\mathcal{T}^\alpha$, $N_{\alpha,V}(s)$ can be computed in time and space $O(|\mathcal{T}|^{\max}|V|^2 \log |\mathcal{D}_v|)$, where $|\mathcal{D}_v|$ denotes the size of the largest variable domain.*

PROOF. We assume that the merge-and-shrink tree does not contain two consecutive shrink nodes, that is, there is no shrink node with a shrink node as its child: a compact representation would combine these two consecutive abstraction steps into one. With this assumption, a merge-and-shrink tree with variables V contains at most $4|V| - 2$ nodes: there must be exactly $|V|$ leaves (one for each variable) and $|V|-1$ merging nodes (because a tree with maximal degree 2 must have one less binary node than its number of leaves), for a total of $2|V| - 1$ nodes that are not shrink nodes. Each such node can have a shrink node as a parent, for a total of at most $2 \cdot (2|V| - 1) = 4|V| - 2$ nodes.

The algorithm for computing $N_{\alpha,V}(s)$ recursively constructs a tree, mirroring $\mathcal{T}^\alpha$, of tables (similar to the cascading table representation of α). For each node in $\mathcal{T}^\alpha$, labeled with an abstraction α', the table associated with the corresponding node in the new tree is indexed by the abstract states in $S^{\alpha'}$ and contains the value of $N_{\alpha',V'}(s)$, where V' is the set of variables for this node. At each node, we first recursively invoke the algorithm to construct these tables for the abstractions labelling each immediate child node, and then use them to construct the table for the node itself. The size of the table for abstraction α is bounded by $|\mathcal{T}|^{\max}|V| \log |\mathcal{D}_v|$ (the factor $|V| \log |\mathcal{D}_v|$ is because $N_{\alpha,V}(s)$ can take values up to $2^{|V| \log |\mathcal{D}_v|}$, requiring $|V| \log |\mathcal{D}_v|$ bits to represent), and the time to construct it is linear in its size. The recursion visits each node in $\mathcal{T}^\alpha$ exactly once, which, as noted previously, is linear in the number of variables. The construction of the table representation of $N_{\alpha,V}(s)$, and the proof that it is correct, is by cases:

$\mathcal{T}^\alpha$ is an atomic abstraction, $\alpha = \pi_v$: In this case, the variable set V equals $\{v\}$. The abstract state space of α, S^α, is simply $\mathcal{D}_v$, and there is a one-to-one correspondence between complete assignments over $\{v\}$ and states in S^α, so $N_{\alpha,V}(s) = 1$ for all $s \in S^\alpha$. Thus, the table representing $N_{\alpha,V}(s)$ is just an array of $|\mathcal{D}_v|$ 1's.

$\mathcal{T}^\alpha$ is shrinking node, $\alpha = \alpha_1 \circ \alpha_2$: In this case, the variable set V that α is defined over equals that which α_2 is defined over. Let s_V be a complete assignment over V: $\alpha(s_V) = s$ iff there exists an $s' \in S^{\alpha_2}$ such that $\alpha_2(s_V) = s'$ and $\alpha_1(s') = s$. Summing over the existentially quantified intermediate state, we therefore have $N_{\alpha,V}(s) = \sum_{s' \in S^{\alpha_2}:\alpha_1(s')=s} N_{\alpha_2,V}(s')$. We have (by recursive invocation) a table representing $N_{\alpha_2,V}(s)$. Thus, initializing the table for $N_{\alpha,V}(s)$ with all 0's, then making a linear sweep over all $s \in S^{\alpha_2}$, adding $N_{\alpha_2,V}(s)$ to the entry for $\alpha_1(s)$ computes the table for $N_{\alpha,V}(s)$.

$\mathcal{T}^\alpha$ is a merging node, $\alpha = \alpha_1 \otimes \alpha_2$, where α_1 and α_2 are defined over disjoint variable sets V_1 and V_2, respectively: In this case, $V = V_1 \cup V_2$. For every complete assignment s_{V_1} over V_1 that α maps to s_1 there are $N_{\alpha_2,V_2}(s_2)$ assignments s_{V_2} over V_2 that α_2 maps to s_2, and since V_1 and V_2 are disjoint the composition of s_{V_1} and s_{V_2} forms a distinct assignment over V. Thus, there are $\sum_{s_{V_1}:\alpha_1(s_{V_1})=s_1} N_{\alpha_2,V_2}(s_2) = N_{\alpha_1,V_1}(s_1)N_{\alpha_2,V_2}(s_2)$ assignments over V that α maps to the abstract state (s_1, s_2). In other words, $N_{\alpha,V}((s_1, s_2)) = N_{\alpha_1,V_1}(s_1)N_{\alpha_2,V_2}(s_2)$. To represent $N_{\alpha,V}(s)$ we construct a two-dimensional table indexed by pairs (s_1, s_2), of size $|\alpha| = |\alpha_1||\alpha_2|$, and assign each entry $N_{\alpha,V}((s_1, s_2))$ the value $N_{\alpha_1,V_1}(s_1)N_{\alpha_2,V_2}(s_2)$, computed from the tables representing $N_{\alpha_1,V_1}(s_1)$ and $N_{\alpha_2,V_2}(s_2)$ which are obtained by the recursive call for each of $\mathcal{T}^\alpha$'s two children. □

COROLLARY 7.7. *Let Θ be a transition system with variables and $\mathcal{T}^\alpha$ a merge-and-shrink tree such that $|\mathcal{T}^\alpha|^{\max}$ is polynomial in the compact description size of Θ. From $\mathcal{T}^\alpha$, the number of states s in Θ with $h^\alpha(s) = c$, for any given c, can be counted in polynomial time.*

PROOF. From the merge-and-shrink tree, we can easily extract the set of variables V^α that the abstraction α is defined over. For every abstract state s', we can also compute $h^\alpha(s')$, the distance to the closest solution state, in polynomial time. The number of states s in Θ that α maps to a given abstract state s' is $N_{\alpha,V^\alpha}(s') \prod_{v \in V \setminus V^\alpha} |\mathcal{D}_v|$. Thus, we can obtain the desired count by a linear scan through the abstract states of Θ^α, evaluating the h^α value on each abstract state to test if it matches, and summing up the count of those that do. □

7.5. Critical-Path Heuristics

AI planning research has come up with several different heuristic construction principles. Helmert and Domshlak [2009] conducted a study of dominance relationships between these, discovering some surprising results. A family of heuristics H is said to dominate a family H' if, for any individual state, H can always simulate H' efficiently. That is, given any state s and any $h' \in H'$, we can in polynomial time construct an additive ensemble of heuristics $h_1, \ldots, h_k \in H$ (with a suitable cost partitioning) such that $\sum_{i=1}^k h_i(s) \geq h'(s)$.[9]

Helmert and Domshlak [2009] proved dominance, or absence thereof, between all four major classes of heuristics in planning (discovering, e.g., that two by appearance quite different classes are in fact equivalent). A single question remained open. That question concerns the family of so-called *critical-path* heuristics. These approximate

[9]Note that this is a weaker form of dominance than what we showed above for merge-and-shrink in comparison to pattern databases. Here, we construct a suitable heuristic $h \in H$ per individual state. By contrast, Proposition 3.7 and Corollary 7.3 show that H = merge-and-shrink always contains a heuristic h dominating $h' \in H'$ = PDBs across *all* states under the conditions of the corollary.

the cost of reaching a state s (an assignment to all variables) by the maximum cost of reaching any m-subset of s (an assignment to at most m variables that complies with s), where m is a parameter. The resulting heuristic function is denoted h^m. Its computation is exponential in m, but polynomial in task size for fixed m. Helmert and Domshlak proved that h^1 is dominated by merge-and-shrink, and that merge-and-shrink is not dominated by h^m (for arbitrary m). The open question is whether h^m, for $m > 1$, is dominated by merge-and-shrink. We now answer this question in the negative for the case of a single merge-and-shrink heuristic (rather than an additive ensemble). For the sake of brevity, we omit the formal definition of h^m, and we only outline our argument. The details are in Appendix A.

Our proof relies on results regarding the counting complexity class **#P**, which consists of the set of functions f for which there exists a nondeterministic Turing machine M that runs in polynomial time, and where $f(x)$ is the number of accepting runs of M on input x. Knowing the value of $f(x)$ clearly allows deciding whether M accepts x, thus the existence of a polynomial time procedure for computing any **#P**-hard function implies that $\mathbf{P} = \mathbf{NP}$. An example of a counting problem is #SAT, the problem of counting the satisfying assignments of a propositional logic formula.

Creignou and Hermann [1996] provide a classification of the complexity of #SAT. A class of formulas for which the problem is **#P**-hard is $\varphi_W = \bigvee_{\{p,q\}\in W}(p \wedge q)$, where W is any set of distinct pairs in a set V of propositions. In our proof, we construct a family of planning tasks corresponding to φ_W, using as variables only the propositions V, plus one additional goal-marker L. The tasks are designed so that, for any abstraction heuristic h^α where $h^\alpha(s_0) = h^*(s_0)$, the states with $h^\alpha(s) = 2$ are exactly those where s projected to V is an assignment that does *not* satisfy φ_W, and where $s(\mathrm{L}) = 0$. Assume that α is a polynomial-time constructible merge-and-shrink abstraction yielding a heuristic with $h^\alpha(s_0) = h^*(s_0)$. By Corollary 7.7, we can use the representation of h^α to count the states with $h^\alpha(s) = 2$, and thus to count the models of φ_W, in polynomial time, implying that $\mathbf{P} = \mathbf{NP}$. On the other hand, h^2 is perfect in these planning tasks, which proves our result.

THEOREM 7.8. *There exist families of factored transition systems where h^2 is a perfect heuristic, whereas unless $\mathbf{P} = \mathbf{NP}$ there exists no polynomial-time abstraction strategy so that merge-and-shrink yields abstractions α guaranteeing that $h^\alpha(s_0) = h^*(s_0)$.*

Together with the earlier results [Helmert and Domshlak 2009], this shows that a single merge-and-shrink is incomparable with the critical-path heuristics h^m for $m \geq 2$. However, to dominate h^2 in the sense of Helmert and Domshlak [2009], we are allowed to use additive ensembles of merge-and-shrink heuristics. For now, the question whether such an additive ensemble of merge-and-shrink heuristics that dominates h^2 exists and can be constructed in polynomial time remains open.

8. APPLICATION TO AI PLANNING

We now evaluate the performance of the merge-and-shrink framework when applied to AI planning, more specifically to the commonly used benchmark set of that field. Section 8.1 discusses theoretical guarantees given in particular benchmarks, Section 8.2 describes our experiments.

8.1. Perfect Heuristics in Polynomial Time

Planning benchmarks distinguish between *domains*. A domain is a parameterized (and typically infinite) set of related planning tasks. The dominant benchmark set in planning, which we also use here, is the one from the International Planning

Competition[10] (IPC). We consider the question whether merge-and-shrink can compute a perfect heuristic, within a given domain, in time polynomial in the domain's size parameters. This is possible only in domains with polynomial-time optimal domain-specific solution algorithms. Helmert [2006b] identifies six such domains in the IPC benchmark set, named Gripper, Movie, PSR, Schedule-Strips, Dining-Philosophers, and Optical-Telegraph. We show that, of these six domains, in all but PSR there exist polynomial-time abstraction strategies for computing perfect merge-and-shrink heuristics.[11] In three of the domains, the Bisim-LR algorithm (cf. Section 6.2) yields such a polynomial-time perfect abstraction strategy. In the interest of brevity, we only outline the investigation.

The Movie and Gripper domains originated in the 1998 IPC. Both are quite simple: Movie involves getting n snacks, and Gripper involves transporting n objects from A to B using a robot with two hands. We have the following proposition.

PROPOSITION 8.1. *Let $\mathcal{P} = \{\Pi_n\}$ be the family of Movie planning tasks, where n is the number of snacks. Let $\mathcal{P}' = \{\Pi'_n\}$ be the family of Gripper planning tasks, where n is the number of transportable objects. For any linear merging strategy, maximum intermediate abstraction size for Bisim-LR is bounded by a linear function in n for $\mathcal{P}$ and by a cubic function in n for $\mathcal{P}'$.*

Both results hold because the Bisim-LR shrinking strategy aggregates states that agree on the relevant object counts (number of snacks obtained, number of objects at A vs. B). In particular, we can solve Movie and Gripper to optimality in polynomial time. This is, of course, not an achievement in its own right (the domains are trivial). The point is that we solve them with a *domain-independent* method. The input to our algorithm is merely a description of the domains, giving no hints whatsoever how to solve them. Proposition 8.1, in this light, means that Bisim-LR is clever enough to automatically uncover all relevant domain structure. Bisim-LR constitutes, in fact, the first domain-independent heuristic function able to do so.

We next consider *scheduling-like* domains, where each task consists of some machines used to change the features $f(o)$ of a set of processable objects o.[12] The relevant property is that, for $o \neq o'$, $f(o)$ and $f(o')$ are mutually independent, that is, not affected by any common operator. Processing an object affects the status of the machines, but does not directly affect any other object. During merge-and-shrink, we can thus treat each of these objects independently, and indeed that is what Bisim-LR does. It is easy to see that the Schedule-Strips domain used in the 2000 IPC is a scheduling-like domain. We thus have the following.

PROPOSITION 8.2. *Let $\mathcal{P}$ be the family of Schedule-Strips planning tasks. There exists a linear merging strategy such that maximum intermediate abstraction size for Bisim-LR in $\mathcal{P}$ is bounded by a polynomial in the number of objects to process in the given task.*

Suitable merging strategies here are ones that start with all machines and then tackle each object in turn.

We remark that label reduction is required for the above results to hold. In all three domains, non-label-reduced bisimulations are exponentially large.

[10] http://www.icaps-conference.org/index.php/Main/Competitions.

[11] We do not have a positive or negative result for PSR, but believe that no perfect polynomial-sized abstraction heuristics exist.

[12] Note the difference to typical Scheduling problems that require to optimize completion time: here, we optimize summed-up cost.

We finally consider Dining-Philosophers and Optical-Telegraph. The former is the well-known problem involving a round table with n philosophers, where the planning goal is to find the deadlock in which each philosopher holds exactly one fork. Optical-Telegraph is essentially the same, except that the "philosophers" now have more internal states. During merge-and-shrink, if we aggregate states that agree on the counts of philosophers/telegraphs in a particular internal state, then the resulting heuristic is perfect. Hence, we have the following.

PROPOSITION 8.3. *Let $\mathcal{P} = \{\Pi_n\}$ be the family of Dining-Philosophers (Optical-Telegraph) planning tasks, where n is the number of philosophers (telegraphs). For any linear merging strategy, there exists a shrinking strategy such that merge-and-shrink in $\mathcal{P}$ yields a perfect heuristic and maximum intermediate abstraction size is bounded by a polynomial in n.*

It is difficult to answer whether Bisim-LR actually yields such abstraction strategies. Bisim-LR depends directly on syntax, and these domains are syntactically very complicated. They were compiled from the Promela language [Holzmann 2004] into planning, using a generic scheme for such compilation [Edelkamp 2003], and relying on many language features not directly supported by our formalism.[13]

Bisim-LR cannot in general reproduce the best possible abstraction strategy. Indeed, the difference can be exponential. One example for this is an extension of Gripper allowing to scale the number of robot hands. Then there still exists an abstraction strategy that yields a perfect heuristic while maximum intermediate abstraction size is polynomially bounded. Such a bound does not hold for Bisim-LR because states differing on the gripper hands are not bisimilar.

8.2. Experiments

In most planning benchmark domains, bisimulations are still large even under label reduction, and Bisim-LR is not feasible. We thus designed a family of (mostly) more approximate merge-and-shrink variants. The family is characterized by four parameters: (1) merging strategy, (2) state aggregation strategy, (3) abstraction size bound N, (4) state aggregation threshold T.

We instantiate (1) with both merging strategies tried in planning so far, called *CausalGraph-Goal-Level* (CGL) and *ReverseLevel* (RL). Both are linear merging strategies (defined by an ordering of the variables), and both are based on well-known notions from planning ([e.g., Helmert 2006a]). In a nutshell, the idea behind CGL is to select the most relevant variables first (close to the planning goal), and the idea behind RL is to select the most influential variables first (prerequisite for many planning operators).

We instantiate (2) with the two strategies discussed, Bisim-LR and GBisim-LR, which we will refer to as B and G, as well as a strategy not based on bisimulation, which we will refer to as H. The latter is h-preserving, aggregating states only if they agree on their remaining cost within the abstract transition system considered, unless the abstraction size bound N forces states with different remaining cost to be aggregated; states with largest remaining cost are aggregated first to avoid introducing errors close to the goal, the intuition being that such errors would propagate and cause errors in large parts of the state space.

We instantiate (3) with $N \in \{10k, 100k, 200k, \infty\}$ and (4) with $T \in \{1, 10k, 100k, 200k, \infty\}$ where $T \leq N$.

The overall shrinking strategy results from the combination of parameters (2), (3), and (4). H performs no aggregations at all until abstraction size reaches the bound

[13]For a straightforward direct planning encoding of Dining-Philosophers, and a variant of greedy bisimilarity, a perfect heuristic is computed in polynomial time. We omit this for brevity.

N, and then proceeds as specified in Figure 1; the value of T has no effect on H. B and G aggregate nothing until abstraction size reaches the threshold T. Then, they attempt to build coarsest (greedy) bisimulations, by starting from a single equivalence class containing all states, and splitting that class between states that are not (greedy) bisimilar. Splits closer to the goal are preferred, and the splits stop once the number of classes reaches the bound N. (This approach was originally proposed by Dräger et al. [2006].)

Members of our merge-and-shrink family, that is, combinations of values for each of the parameters, will be denoted in the text according to the scheme (1)-(2)-N(3)-T(4). The total number of possible parameter combinations, and thus merge-and-shrink variants explored here, is 64.

Our merge-and-shrink variants were quite successful at the optimal planning track of the 2011 International Planning Competition, together with two other planners, BJOLP [Domshlak et al. 2011] and LM-cut [Helmert and Domshlak 2009], which are also based on A* search but with heuristic functions generated in a very different manner. A portfolio of two merge-and-shrink variants, namely RL-G-$N\infty$-T1 and RL-G-$N200k$-$T200k$, won a 2nd prize at this track of IPC'11; combined in a simple sequential portfolio with BJOLP and LM-cut (running each algorithm for a fixed amount of time), the same two merge-and-shrink variants won the 1st prize. The five best performing optimal planners at IPC'11 were all based on A* and used one or several out of these four heuristic functions (and no others). Thus these heuristics represent the state of the art in optimal planning as measured at IPC'11. Accordingly, in the following we compare the performance of our merge-and-shrink variants to that of BJOLP and LM-cut.

We also compare with pattern database (PDB) heuristics, that is, abstractions based only on projection [Culberson and Schaeffer 1998; Edelkamp 2001]. This comparison is relevant because these are the closest relatives to merge-and-shrink.

PDB heuristics require a strategy to select the set, or sets, of variables to project on (the pattern). When we use more than one pattern, the pattern set may contain subsets of mutually independent patterns; heuristic values are aggregated within these subsets using the sum, and across these subsets using the max. This is called the canonical PDB heuristic [Haslum et al. 2007] and yields the best (i.e., highest) admissible value that can be obtained from any given PDB collection without partitioning the cost of operators. We consider four different strategies for selecting patterns.

Strategy (a) chooses a single pattern, including variables in the order given by the CGL merge strategy until no more variable can be added without the size of the abstract transition system exceeding a fixed bound P. We used $P \in \{100k, 1000k, 10000k, 100000k\}$. Note that these limits are larger than what is feasible to use with merge-and-shrink; this is because a PDB, being a projection, can be indexed by a perfect hash function, and therefore computed more efficiently and stored more compactly [Edelkamp 2001].

Strategy (b) creates one pattern for each variable that has a goal value, containing only that variable.

Strategy (c) combines the first two, choosing one pattern up to the size limit P, then one with each remaining goal variable.

Strategy (d) is an incremental construction [Haslum et al. 2007]. It selects patterns by a hill-climbing search in the space of pattern collections, starting from a collection with a single-variable PDB for each goal variable and growing it by iteratively adding one new pattern that extends one of those in the current collection with one variable. The pattern to add is chosen to maximize an estimate of the improvement that including it will make to the canonical heuristic. The estimate is obtained by evaluating a (relatively small) random sample of states. The strategy has four different

					RL												CGL			No L.P. CGL			RL
			PDB		$10K$			$100K$			$200K$			∞			$10K$			∞	∞	$10k$	$100K$
D	BJ	LM	(a)	(d)	B	G	H	B	G	H	B	G	H	B	G	H	B	G	H	B	B	H	RND
airport	28	28	22	27	21	22	11	12	22	3	11	22	2	1	22	1	22	22	18	15	16	18	0
barman	4	4	4	4	4	4	4	4	4	0	4	4	0	0	4	0	4	4	4	0	0	0	0
blocks	26	28	22	22	21	21	18	18	21	18	18	21	18	9	21	9	21	21	18	9	9	18	9
depot	7	7	7	7	6	7	4	7	7	3	5	7	3	1	7	1	7	7	7	2	2	6	1
driverlog	14	13	11	12	12	12	12	12	12	12	12	12	12	5	12	4	12	12	12	5	4	12	5
elevators	13	18	12	16	9	9	10	9	9	10	11	11	11	0	0	0	9	9	8	3	3	6	0
floortile	2	7	3	2	3	2	2	6	2	3	7	2	3	8	2	6	2	2	2	2	2	2	0
freecell	60	15	18	20	16	16	12	6	16	4	3	16	3	3	16	3	16	16	15	3	3	13	0
grid	2	2	2	3	2	2	1	3	2	1	3	2	1	0	2	0	2	2	2	0	0	2	0
gripper	7	7	7	7	11	7	7	20	7	7	20	7	7	20	7	7	7	7	7	4	4	7	5
logistics00	20	20	16	20	20	16	17	20	16	20	20	16	20	10	16	10	19	16	16	10	10	16	10
logistics98	6	6	4	5	4	4	4	5	4	5	5	4	5	2	4	2	4	4	5	2	2	5	2
miconic	142	141	61	45	58	50	52	64	50	55	67	50	55	59	50	40	65	50	55	60	45	55	45
mprime	21	23	23	23	18	23	23	14	23	12	10	23	8	1	23	1	23	23	22	1	1	20	1
mystery	15	17	15	15	12	15	15	8	15	11	7	15	9	4	15	2	15	15	14	3	2	13	2
nomystery	20	15	12	16	18	12	14	20	12	17	20	12	18	13	12	11	18	12	14	13	11	14	8
openstacks	12	14	13	14	13	13	0	14	14	0	14	14	0	3	4	1	14	14	1	3	2	7	0
openstack-str	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	5	0	5
parcprinter	10	13	9	11	12	11	10	13	11	12	13	11	12	12	11	12	10	11	8	8	9	8	8
parking	3	2	1	5	4	5	0	0	5	0	0	5	0	0	5	0	4	5	1	0	0	0	0
pathways	4	5	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	2	2	4	2
pegsol	17	17	17	17	19	19	2	19	19	7	19	19	6	0	0	0	17	17	0	15	15	0	0
pipes-notank	17	17	15	16	17	15	10	11	15	1	6	15	1	2	15	2	13	15	11	0	0	9	0
pipes-tank	13	12	16	16	14	15	11	8	16	6	7	16	3	2	16	2	14	14	14	6	6	7	0
psr-small	49	49	50	49	49	49	49	49	50	49	49	50	49	45	50	46	50	50	50	49	45	50	43
rovers	7	7	6	7	6	6	6	8	6	6	8	6	6	4	6	4	7	6	7	4	4	7	4
satellite	7	7	6	6	6	6	6	7	6	6	7	6	6	6	6	4	6	6	6	5	4	6	4
scanalyzer	6	11	9	10	9	10	9	10	10	8	7	10	9	3	3	3	9	9	9	3	3	9	3
sokoban	20	20	17	20	19	19	1	20	19	1	19	18	1	0	1	0	20	19	6	7	8	2	0
tidybot	14	14	1	14	4	13	0	0	13	0	0	13	0	0	13	0	8	13	1	0	0	1	0
tpp	6	6	6	6	6	6	6	7	6	7	7	6	7	5	6	5	6	6	6	5	5	6	5
transport	6	6	6	6	6	6	6	6	6	6	6	6	6	1	6	1	6	6	6	4	4	6	1
trucks	7	10	8	8	7	6	6	6	6	6	6	6	6	5	6	4	6	6	6	5	4	6	2
visitall	10	11	9	12	9	12	9	10	12	10	10	12	10	8	12	8	9	12	9	8	8	9	8
woodworking	8	11	6	6	6	8	2	4	9	2	4	9	2	2	9	1	5	9	6	4	2	6	0
zenotravel	10	13	10	11	11	9	9	12	9	11	12	9	11	7	9	7	11	9	11	7	7	11	7
total	620	603	455	489	463	461	359	443	465	330	428	466	321	252	402	208	472	460	388	274	247	361	180
w/o miconic, freecell	418	447	376	424	389	395	295	373	399	271	358	400	263	190	336	165	391	394	318	211	199	293	135
MS built					998	1054	839	774	1126	651	707	826	581	252	1061	208	1011	1057	899	274	247	684	180
#D in-best	14	21	6	11	5	8	4	14	10	3	12	10	3	3	8	1	8	9	5	1	0	3	0
#(D >BJ)					9	8	2	10	10	5	10	10	5	3	7	2	7	9	5	0	0	3	0
#(D >LM)					6	5	0	6	6	2	6	6	2	2	5	0	5	5	2	0	0	1	0
#(D >PDB)					3	2	0	10	2	2	10	2	2	3	1	2	1	1	0	0	0	0	0

Fig. 9. Coverage data in IPC benchmarks for a selection of algorithms. Abbreviations: "D" for domains; "BJ" is BJOLP; "LM" is LM-cut; "PDB (a)" is strategy (a) with $P = 100000k$; "PDB (d)" is strategy (d) with $P = 10000k, C = 100000k, S = 100, I = 10$; "No L.P." means no label projection; "RND" is a random state aggregation strategy; "MS built" is the number of instances in which the merge-and-shrink abstraction process completed within the given time and memory; "#D in-best" is the number of domains where coverage is maximal among all algorithms run (not only among those shown here); "#(D > BJ)" is the number of domains where coverage is strictly larger than that of BJOLP; "#(D > LM)" is the same for LM-cut; "#(D > PDB)" is the number of domains where coverage is strictly larger than that of any PDB in the experiment.

parameters: a limit P on the size of any single PDB in the collection, and a limit C on the size of the collection as a whole; the number S of samples used to estimate improvement; and finally a limit I on the minimum improvement required to add a new pattern to the collection. We used $P \in \{1000k, 2000k, 5000k, 10000k\}$, $C = 10 \cdot P$, and $(S, I) \in \{(100, 1), (100, 10), (1000, 10), (1000, 100)\}$.

In total, this gave us 25 different PDB heuristics.

All heuristics compared are implemented within the same framework, Fast Downward [Helmert 2006a], and all results we report use the same A* implementation. We ran experiments on 44 benchmark domains, from all seven editions of the International Planning Competition so far, IPC'98 – IPC'11. For the eight domains that were used in both IPC'08 and IPC'11, we use the benchmark instances from the latter. The experiments were performed on dual-CPU AMD Opteron 2384 machines, running eight experiments simultaneously in order to fully utilize the available (eight-core) machines. Each planner instance ran on a single core with a time limit of 30 minutes and a memory limit of 2 GB.

Figure 9 gives coverage data, that is, the number of benchmark instances solved within the given time and memory (the size of each domain's benchmark set is included

		Solved by Any						Portfolios					
D	$\|D\|$	BL	PDB	MS	BL+ PDB	BL+ MS	BL+ PDB+ MS	BL	PDB	MS	BL+ PDB	BL+ MS	BL+ PDB+ MS
airport	50	30	27	23	32	30	32	30	27	22	32	30	32
barman	20	4	4	4	4	4	4	4	4	4	4	4	4
blocks	35	28	23	22	28	28	28	28	23	21	28	28	28
depot	22	7	7	7	7	7	7	7	7	7	7	7	7
driverlog	20	14	14	13	15	15	15	14	12	12	14	14	14
elevators	20	18	16	14	18	18	18	16	16	11	16	16	16
floortile	20	7	3	8	8	8	8	6	3	7	6	7	7
freecell	80	60	21	19	60	60	60	55	20	16	54	54	53
grid	5	2	3	3	3	3	3	2	3	3	3	3	3
gripper	20	7	7	20	20	20	20	7	7	20	7	20	20
logistics00	28	20	20	20	22	22	22	20	20	20	20	20	20
logistics98	35	6	5	5	6	6	6	6	5	5	6	6	6
miconic	150	142	61	81	142	142	142	142	61	67	141	141	141
mprime	35	23	23	23	24	24	24	22	23	23	23	23	23
mystery	30	17	15	15	17	17	17	15	15	15	15	15	15
nomystery	20	20	18	20	20	20	20	20	16	20	19	20	20
openstacks	20	14	14	14	14	14	14	14	14	14	14	14	14
openstack-str	30	7	7	7	7	7	7	7	7	7	7	7	7
parcprinter	20	13	11	13	13	13	13	13	10	13	13	13	13
parking	20	3	5	5	5	5	5	2	5	5	5	5	5
pathways	30	5	4	4	5	5	5	5	4	4	5	5	5
pegsol	20	17	18	19	19	19	19	17	17	19	17	19	19
pipes-notank	50	17	17	17	18	18	18	17	16	15	17	17	17
pipes-tank	50	13	17	16	17	16	17	12	17	16	17	16	17
psr-small	50	49	50	50	50	50	50	49	50	50	50	50	50
rovers	40	7	7	8	8	8	8	7	7	8	7	8	8
satellite	36	7	6	7	7	7	7	7	6	7	7	7	7
scanalyzer	20	12	10	10	14	14	14	12	10	10	14	14	14
sokoban	20	20	20	20	20	20	20	20	20	20	20	20	20
tidybot	20	14	14	13	14	14	14	14	14	13	14	14	14
tpp	30	6	6	8	7	8	8	6	6	7	6	7	7
transport	20	6	6	6	6	6	6	6	6	6	6	6	6
trucks	30	10	9	7	11	10	11	9	9	6	9	9	9
visitall	20	11	12	13	13	13	13	10	12	12	12	12	12
woodworking	20	12	7	12	12	14	14	12	6	9	11	11	11
zenotravel	20	13	11	12	13	13	13	12	11	12	12	12	12
total	1156	661	518	558	699	698	702	645	509	526	658	674	676
w/o miconic, freecell	926	459	436	458	497	496	500	448	428	443	463	479	482
#D in-best		18	11	18	34	33	36	23	17	22	27	31	33

Fig. 10. Coverage data in IPC benchmarks for a selection of algorithm combinations. "BL" stands for BJOLP and LM-cut. In "Solved by Any", "PDB" ("MS") stands for all PDB (merge-and-shrink) variants in the experiment; in "Portfolios", "PDB" ("MS") stands for the pair of PDB (merge-and-shrink) variants whose portfolio performs best in the total, among all such pairs. Every portfolio runs the component algorithms in sequence, giving each component an equal share of the 30 minutes runtime.

below in Figure 10). For space reasons, we show data for only 2 of the 25 PDB variants, and for 16 of the 64 merge-and-shrink variants; 3 additional merge-and-shrink variants (outside the family described previously) are included for test reasons. The two PDBs shown are strategy (a) with $P = 100000k$ ("PDB (a)") and strategy (d) with $P = 10000k, C = 100000k, S = 100, I = 10$ ("PDB (d)"). Out of all possible sequential portfolios combining two PDB variants (dividing runtime evenly), this pair perform best in terms of total coverage; "PDB (d)" also is the best variant individually. For merge-and-shrink, we show data only for $T = 1$ because this setting dominates all $T > 1$ almost universally. We further omit data for CGL with $N = 100k$ or $N = 200k$, and for CGL with $N = \infty$ and G or H because these are almost universally dominated either by the corresponding variant using RL, or by the corresponding variant using CGL with $N = 10k$.

Consider first only the merge-and-shrink variants in Figure 9. With $N = \infty$, using B yields a perfect heuristic but, as earlier hinted, is seldom feasible. CGL works slightly

better than RL (274 for CGL vs. 252 for RL in the total). Comparing to the third column from the right, CGL-B-$N\infty$-$T1$ with label projection turned off, we see that reducing labels is important for obtaining smaller bisimulations (not using it, we solve only 247 instances). Note also that H with $N = \infty$ does not aggregate any states at all, so builds the full state space; the small total coverage (208 for RL-H-$N\infty$-$T1$) shows the advantage of bisimulation over a blind approach.

State aggregation method G typically builds smaller abstractions than B, and thus incurs less of a runtime overhead. This is most pronounced in the absence of an abstraction size bound ($N = \infty$) where, for RL, B completes only 252 abstractions while G completes 1061. This advantage often translates into better performance of G rather than B in terms of total coverage (H is much worse than both). Interestingly, the same is not the case when considering instead the count of domains where a technique is competitive: as the last four rows of Figure 9 show, in this regard B often is better. These differences are a result of the different accuracy vs. runtime trade-offs made by the two methods—B tends to favor heuristic accuracy while G tends to favor small runtimes—whose relative performance depends on the domain. We will get back to this in some more detail.

For state aggregation strategy H, the states aggregated are exactly the same whether or not label projection is used. Thus, in this setting, we can test the impact of label projection as a technique to reduce computational effort independently from the shrinking strategy. The second column from the right shows data for CGL-H-$N10k$-$T1$ with label projection turned off. Comparing this to the data for CGL-H-$N10k$-$T1$ with label projection turned on (5th column from the right), the better coverage of the latter (388 vs. 361 in the total) shows that label projection is important even just as an implementation technique. Finally, as a sanity test, we ran a merge-and-shrink variant selecting the state pairs to aggregate randomly (rightmost column in Figure 9). The result clearly confirms that these choices need to be taken carefully.[14]

Comparing merge-and-shrink to its competitors in Figure 9, clearly all the competitors (except "PDB (a)") do better in terms of total coverage. For BJOLP and LM-cut, the difference is huge but is, to a considerable extent, due to only two domains. They excel in Miconic-STRIPS and FreeCell, which contain more instances than the other domains. Ignoring Miconic-STRIPS and FreeCell, the difference becomes much smaller. More generally, the relative performance of different planners typically is mainly determined by the domain.

Counting solved instances suffers, as a performance measure, not only from the choice of domains that happen to be included in the IPC, but also from the choice of instances and their number (as vividly demonstrated here by Miconic-STRIPS). A more stable measure is to only count the domains in which a planner performs well. In that measure, LM-cut is in the lead by far, but the best merge-and-shrink variant is equally good as BJOLP and better than the PDBs. For each of the competitors, there are several domains (10 for BJOLP, 6 for LM-cut, 10 for PDBs) where the best merge-and-shrink variant is strictly better.[15]

[14]The random aggregation typically leads to failure during construction of the abstraction already. From a superficial check, it appears the main reason is that random state aggregation yields very densely connected graphs (abstract transition systems), that is, many arcs whose handling consumes a lot of runtime and memory.

[15]Regarding the comparison to PDB (d), note also that this strategy generates heuristic functions based on several patterns, that is, based on additive ensembles of heuristics. By contrast, the merge-and-shrink variants shown generate heuristics based on single abstractions. While merge-and-shrink can, in principle, capture additivity within a single heuristic (cf. Section 7.1), most likely this kind of technique could be used to further improve the performance of merge-and-shrink in practice.

The approaches shown have strengths in different domains, and can be fruitfully combined. Indeed, as mentioned, a portfolio doing so won the optimal planning track of IPC'11. Figure 10 gives coverage data for some relevant combinations. We show data for "Solved by Any" to gauge the extent of complementary strength within a group of planners (ranging over all PDB/merge-and-shrink configurations run in the experiment). We show sequential portfolios (dividing runtime evenly across all component planners) as a simple means to gauge the extent to which this complementarity can be turned into an advantage when respecting the runtime bound. For merge-and-shrink, all portfolio data shown is for the two configurations RL-G-$N100k$-$T1$ and RL-B-$N200k$-$T1$, the merge-and-shrink pair for which the portfolio performed best in the total; similarly for PDBs (where these two configurations are those shown in Figure 9). The counts "#D in-best" of domains with best performance are within each group, that is, relative to the other "Solved by Any" and "Portfolios" configurations.

Planner combination helps significantly even within the merge-and-shrink family, raising the total from 472 for the best individual variant to 558 for the instances solved by any variant, and to 526 for the best portfolio of two variants. In the present data, merge-and-shrink has more potential for combination than PDBs, overtaking them in both solved-by-any (total 558 vs. 518, #D in-best 18 vs. 11) and in sequential portfolios of variant pairs (total 526 vs. 509, #D in-best 22 vs. 17). This must be qualified against the sets of variants run here, and in particular against the fact that we run 64 merge-and-shrink variants vs. 25 PDBs. That said, the picture hardly changes when fixing $T = 1$ and thus reducing our merge-and-shrink set to 24 variants: the best portfolio remains the same; the change for solved-by-any is minor, and PDBs still are clearly dominated (total now 551 vs. 518, #D in-best now 17 vs. 11).

BJOLP and LM-cut (together) are still in the lead, but merge-and-shrink is getting very close (#D in-best identical for solved-by-any, and 22 vs. 23 for portfolios). This is especially true when ignoring Miconic-STRIPS and FreeCell, both for solved-by-any (total 458 vs 459, #D in-best 18 vs 16) and for our simple portfolios here (total 443 vs. 448, #D in-best 22 vs. 21). Adding either of merge-and-shrink or PDBs to BJOLP and LM-cut helps significantly, showing that each of merge-and-shrink and PDBs can contribute to the state of the art in optimal planning. The performance is about equal regarding solved-by-any (total 698 vs. 699, #D in-best 34 vs. 33), and is stronger for merge-and-shrink regarding portfolios (total 674 vs. 658, #D in-best 31 vs. 27). Adding both merge-and-shrink and PDBs on top of BJOLP and LM-cut yields another small improvement in each case.

Coverage is a function of the tradeoff between the accuracy of a heuristic function, and the effort needed for computing it. Due to the multitude of domains and algorithms tested, it is beyond the scope of our discussion to give detailed data. We provide a summary for a few of the algorithms, using Richter and Helmert's [2009] *expansion score* (E) and *total-runtime score* (T). Both range between 0 and 100, for each individual instance. $E = 100$ if ≤ 100 expansions were made, $E = 0$ if $\geq 1{,}000{,}000$ expansions were made. In between, E interpolates logarithmically, so that an additive difference of 7.53 in scores corresponds to a factor 2. Similarly, $T = 100$ for runtimes $\leq$1 second, $T = 0$ for time-outs, and doubling the runtime decreases the score by about 9.25. The advantage of these scores is that they are absolute, that is, there is no need to restrict the set of instances considered to those solved by all planners.[16] Figure 11 compares the two merge-and-shrink variants constituting the best merge-and-shrink

[16]Experimenting with summaries limited to commonly solved tasks, we found that they often misrepresented the results. For example, on instances solved by both, RL-G-$N\infty$-$T1$ beats LM-cut even in some domains where LM-cut actually scales better—because RL-G-$N\infty$-$T1$'s cheap heuristic solves the small tasks very quickly, and times out on the larger ones.

X = RL-G-$N100k$-$T1$			X = BJOLP			X = LM-cut			X = PDB (d)		
Y = RL-B-$N200k$-$T1$			Y = CGL-B-$N10k$-$T1$			Y = CGL-B-$N10k$-$T1$			Y = CGL-B-$N10k$-$T1$		
	D	E	T	D	E	T	D	E	T	D	E	T
gripper	87.3	64.4	psr	14.8	2.8	psr	12.8	5.7	miconic	15	14.5
nomystery	43.5	20.3	pegsol	7.6	15.1	openst	5.1	2.2	nomystery	11.7	5.1
parcprint	36.5	-2.6	scanaly	5.1	15.2	nomystery	3	23.2	psr	6.2	7.8
log00	23	0.4	parcprint	4.9	3.5	gripper	1.3	4.8	zenotrav	5.8	5.7
miconic	22.7	6.4	zenotrav	4.1	6	visitall	0.6	-0.9	openst	5.6	0.4
floortile	18.4	12.4	visitall	3.4	-0.4	freecell	0.2	3.6	satellite	5.3	1.3
driverlog	17.8	-17.3	mystery	3.3	-2	barman	0	2.9	driverlog	3.5	1.6
grid	13	-9.7	driverlog	3.3	-5.8	tpp	-0.5	0	pathways	3.1	-2.3
transport	11.5	-3.2	sokoban	2.9	-2.6	pipestank	-0.5	2.1	tpp	2.5	1
zenotrav	9.9	-3.7	openst	1.9	-0.4	openst-str	-0.5	7.6	gripper	1.9	-0.2
satellite	8.8	1.6	gripper	1.2	2.6	parking	-1.2	5	floortile	1	3.6
psr	8.4	-11.3	parking	1.2	3.7	satellite	-1.8	-1.6	openst-str	0.2	-5.4
openst	6.8	-2.1	satellite	1	-1.8	log00	-2.3	2.9	pipesnota	0.1	-15.6
trucks	6.3	-1.8	pathways	1	-2.7	rovers	-2.7	-1.1	barman	0	-0.4
pathways	6.3	-2.4	trucks	0.9	-1.2	pathways	-3.7	-4.1	rovers	-0.1	0.5
sokoban	5	-12.3	tpp	0.7	0.1	pegsol	-3.8	10.8	mprime	-0.9	-14.8
pegsol	3.9	-5.1	mprime	0.7	-7	pipesnota	-4.7	-11.7	transport	-1.5	-2.2
elevators	3.5	1.6	openst-str	0.2	15.8	depot	-7	2.1	parking	-1.5	-9.4
tpp	3.4	0.8	floortile	0.2	1.6	driverlog	-7.6	-1.1	log98	-1.6	-2.1
rovers	2.9	1.4	barman	0	2	log98	-7.9	-4.5	scanaly	-2.6	-4.4
log98	2.9	0.3	pipestank	-1.2	-2.1	floortile	-8.5	-10.4	depot	-3.2	-5.7
openst-str	0.2	-8.5	rovers	-1.7	-1.6	zenotrav	-10	-0.1	freecell	-3.5	-0.9
pipestank	0.1	-18.2	log00	-2.2	-4.4	trucks	-10.1	-7.2	trucks	-3.5	-4
barman	0	-4.8	blocks	-2.7	-7	grid	-10.9	3.3	pegsol	-4.6	59.9
scanaly	-1.5	-9.4	wood	-2.9	-6.5	transport	-13.2	2.6	pipestank	-5	-0.5
depot	-2	-14.5	transport	-3	1.4	mystery	-13.4	-6	blocks	-5.1	-6.7
pipesnota	-3.2	-22	depot	-3.8	-2.2	mprime	-15.3	-7.4	visitall	-6.4	-14.1
visitall	-3.3	-17.7	pipesnota	-4.8	-17	blocks	-16.4	-13.5	parcprint	-6.5	6.7
wood	-4.6	-30.7	log98	-4.8	-5	sokoban	-19.5	-14.7	log00	-8.7	-0.6
airport	-4.8	-31.5	elevators	-8.4	-8.4	airport	-19.7	-18.8	sokoban	-9.5	-33.1
parking	-5.2	-16.6	grid	-11.1	-7.4	parcprint	-25.1	-16.6	mystery	-9.8	-17.8
freecell	-5.4	-15	nomystery	-12.1	-2	wood	-28.2	-19.6	tidybot	-10.2	-44.9
mystery	-9.3	-33.7	airport	-14.4	-22.9	tidybot	-29.5	-26.3	elevators	-12.5	-27.2
blocks	-9.5	-23.6	tidybot	-15.2	-44.4	scanaly	-31.8	-8.7	wood	-16.6	7.1
mprime	-14.1	-47.1	freecell	-38.3	-40.7	elevators	-32.8	-18.5	airport	-20	-11.3
tidybot	-15.8	-34.6	miconic	-58	-53.9	miconic	-57.9	-47.2	grid	-26	-16

Fig. 11. Difference $Y - X$ between per-domain averaged expansions (E) and total-runtime (T) scores (as per Richter and Helmert, see text), for selected pairs of heuristics X, Y. Columns ordered by decreasing $E(Y) - E(X)$.

portfolio (RL-G-$N100k$-$T1$ and RL-B-$N200k$-$T1$) against each other, and compares the individually best merge-and-shrink variant (CGL-B-$N10k$-$T1$) against each of the competing approaches (where for PDBs we selected the configuration PDB (d) from Figure 9).

Considering first the leftmost part of the table, X = RL-G-$N100k$-$T1$ vs Y = RL-B-$N200k$-$T1$, we clearly observe the aforementioned differences in the accuracy vs. runtime tradeoffs of G vs B. $E(Y) - E(X)$ is positive in the majority of the domains, and reaches very large values at its top end, showing that RL-B-$N200k$-$T1$ has a clear advantage in terms of heuristic accuracy. Despite this, $T(Y) - T(X)$ is negative in the majority of the domains, that is, RL-G-$N100k$-$T1$ is in the advantage because the overhead for creating the abstraction is smaller (consider, e.g., ParcPrinter and Driverlog).

In the comparison between X = BJOLP and Y = CGL-B-$N10k$-$T1$, none of the two planners has the edge in expansions: $E(Y) - E(X)$ is positive in a few more domains which plays in favor of CGL-B-$N10k$-$T1$, but the extent of the difference grows much larger on the domains where $E(Y) - E(X)$ is negative (extreme end: Miconic-STRIPS). In most domains, the planner better in expansions also is better in runtime (the most notable exception to this is Driverlog).

For LM-cut, the picture is very different. $E(Y) - E(X)$ is negative in 30 of the 36 domains considered here, and the difference is larger than 7.53 (i.e., about a factor 2) in 18 of the domains. On the other hand, LM-cut creates a large overhead. It does not require a costly pre-process (like computing an abstraction), but takes a lot of time to compute in each state. This often counters the effect of the better heuristic accuracy (consider, e.g., Grid and Transport); the number of domains where $T(Y) - T(X)$ is negative is only 21, in only 10 of which the difference is larger than 9.25 (i.e., about a factor 2).

Consider finally the rightmost part of Figure 9, comparing PDB (d) to CGL-B-$N10k$-$T1$. A basic observation about $E(Y) - E(X)$ is that its absolute values are much smaller than in the comparisons to BJOLP and LM-cut; the same is true when comparing PDB (d) to RL-G-$N100k$-$T1$ and RL-B-$N200k$-$T1$ (except for Gripper where RL-B-$N200k$-$T1$ has a huge advantage). This is natural since PDBs are much more closely related to merge-and-shrink than BJOLP and LM-cut. PDB (d) has a slight advantage in terms of expansions. In some cases, this advantage is bought at prohibitively high costs in runtime (consider in particular PegSolitaire and Woodworking). Conversely, prohibitively high runtime costs also sometimes occur in CGL-B-$N10k$-$T1$ (e.g., in TidyBot the advantage of PDB (d) is much larger in T than in E). All in all, the two techniques have complementary strengths, and both suffer from occasional efficiency issues in the computation of abstractions.

9. CONCLUSION

Merge-and-shrink abstractions are a new tool to generate lower bound estimates for reachability in compactly described discrete transition systems. We have identified the class of transition systems—factored ones—which allow these abstractions to be built in a practical, incremental way, without loss of information. This also enabled us to demonstrate the power of the approach, in particular with respect to pattern databases, which are, in theory, strictly dominated by merge-and-shrink abstractions. The picture is not as clear in practice, where pattern databases have advantages because they can be implemented very efficiently; but merge-and-shrink contributes to the state of the art in planning.

Since we observed that a major strength of merge-and-shrink lies in the combination of heuristics resulting from different abstraction strategies, a major question for future research is whether this can be done in a more targeted manner. Can we design meta-strategies that automatically choose a suitable combination of parameter values? A related question is to what extent merge-and-shrink can profit from using it in additive ensembles rather than relying on a single abstraction.

Our present data shows that a change in the merging strategy can have a huge effect on performance. The influence of the merging strategy on the merge-and-shrink process, and in particular of its interaction with the shrinking strategy, are not yet well understood.

An interesting generalization of greedy bisimilarity is to allow the algorithm to choose an arbitrary subset of transitions relative to which bisimilarity will be established. This allows fine-grained abstraction design, controlling the size of the transition subset to trade off accuracy against computational effort. We have already obtained first results along these lines [Katz et al. 2012].

Finally, merge-and-shrink can be applied to any domain in which PDBs have been applied. Given the success of PDBs, combined with the strict domination of merge-and-shrink over PDBs in theory and the empirical advantages in part of our benchmarks, we consider this an exciting line of research that we hope will inspire other researchers as well.

APPENDIX

A. PROOFS

We restate all formal claims, now with detailed proofs. We proceed in order of presentation of the main text.

A.1. Basic Concepts

PROPOSITION 2.4. Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system with state variables $\mathcal{V}$. Then Θ is factored if and only if the following two conditions hold.

(i) There exists a family of "goal domains" $(\mathcal{D}_v^{\mathrm{g}})_{v\in\mathcal{V}}$ where $\mathcal{D}_v^{\mathrm{g}} \subseteq \mathcal{D}_v$ for all $v \in \mathcal{V}$, such that for all $s \in S$, we have $s \in S_\star$ iff $s(v) \in \mathcal{D}_v^{\mathrm{g}}$ for all $v \in \mathcal{V}$.
(ii) For each label $l \in L$, there exists a family of "precondition/effect domains" $(\mathcal{D}_v^{\mathrm{pe}}(l))_{v\in\mathcal{V}}$ where $\mathcal{D}_v^{\mathrm{pe}}(l) \subseteq \mathcal{D}_v \times \mathcal{D}_v$ for all $v \in \mathcal{V}$, such that for all $s, s' \in S$, we have $(s, l, s') \in T$ iff $(s(v), s'(v)) \in \mathcal{D}_v^{\mathrm{pe}}(l)$ for all $v \in \mathcal{V}$.

PROOF. We consider solution states and transitions separately, proving that each of the characterizations is equivalent to the respective one given in Definition 2.3. We denote the conditions (i), (ii) of that definition with (i'), (ii').

Trivially, (i) implies (i'): recombining any $s_\star, t_\star$ clearly results in a state with the same property. To see that (i') implies (i), define $\mathcal{D}_v^{\mathrm{g}} := \{s_\star(v) \mid s_\star \in S_\star\}$. Clearly, if $s \in S_\star$, then $s(v) \in \mathcal{D}_v^{\mathrm{g}}$ for all v. Conversely, consider s with $s(v) \in \mathcal{D}_v^{\mathrm{g}}$ for all v. By construction, for each $v \in \mathcal{V}$ there exists a state $s_\star^v \in S_\star$ so that $s_\star^v(v) = s(v)$. Let $v_1, \ldots, v_n$ be some ordering of $\mathcal{V}$. By (i'), $s_\star^{1,2} := \rho_{\{v_1\}}(s_\star^{v_1}, s_\star^{v_2}) \in S_\star$. By construction, $s_\star^{1,2}$ agrees with s on both v_1 and v_2. Now iterate the argument by setting $s_\star^{1,\ldots,i+1} := \rho_{\{v_1,\ldots,v_i\}}(s_\star^{1,\ldots,i}, s_\star^{v_{i+1}})$. We get $s = s_\star^{1,\ldots,n} \in S_\star$.

It is easy to see that (ii) implies (ii'): if we recombine s with s' and t with t' on the same variable set, then the relevant property—$(\rho_V(s,t)(v), \rho_V(s',t')(v)) \in \mathcal{D}_v^{\mathrm{pe}}(l)$—remains intact for each variable v. To prove that (ii') implies (ii), we define a suitable family of precondition/effect domains. We set $\mathcal{D}_v^{\mathrm{pe}}(l) := \{(s(v), l, s'(v)) \mid (s, l, s') \in T\}$. Clearly, if $(s, l, s') \in T$, then $(s(v), s'(v)) \in \mathcal{D}_v^{\mathrm{pe}}(l)$ for all v. Conversely, consider s, s' with $(s(v), s'(v)) \in \mathcal{D}_v^{\mathrm{pe}}(l)$ for all v. By construction, for each $v \in \mathcal{V}$ there exists a transition $(s^v, l, s'^v) \in T$ so that $s^v(v) = s(v)$ and $s'^v(v) = s'(v)$. Let $v_1, \ldots, v_n$ be some ordering of $\mathcal{V}$. By (ii'), if we define $s^{1,2} := \rho_{\{v_1\}}(s^{v_1}, s^{v_2})$ and $s'^{1,2} := \rho_{\{v_1\}}(s'^{v_1}, s'^{v_2})$ then $(s^{1,2}, l, s'^{1,2}) \in T$. By construction, $s^{1,2}$ agrees with s on both v_1 and v_2, and $s'^{1,2}$ agrees with s' on both v_1 and v_2. Now iterate the argument by setting $s^{1,\ldots,i+1} := \rho_{\{v_1,\ldots,v_i\}}(s^{1,\ldots,i}, s^{v_{i+1}})$ and $s'^{1,\ldots,i+1} := \rho_{\{v_1,\ldots,v_i\}}(s'^{1,\ldots,i}, s'^{v_{i+1}})$. We get $s = s^{1,\ldots,n}$, $s' = s'^{1,\ldots,n}$, and $(s^{1,\ldots,n}, l, s'^{1,\ldots,n}) \in T$. □

PROPOSITION 2.7. *Let $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$ be a planning task. Then, $\Theta(\Pi)$ is factored with state variables $\mathcal{V}$.*

PROOF. To see this, just note that planning tasks satisfy the characterization in Proposition 2.4. Suitable goal domains are defined by $\mathcal{D}_v^{\mathrm{g}} := \{s_\star(v)\}$ if $s_\star(v)$ is defined and $\mathcal{D}_v^{\mathrm{g}} := \mathcal{D}_v$ otherwise. Further, if o is a label in $\Theta(\Pi)$ (and hence an operator in $\mathcal{O}$), then suitable precondition/effect domains are given by

$$\mathcal{D}_v^{\mathrm{pe}}(o) := \begin{cases} \{(pre_o(v), eff_o(v))\} & \text{if } v \in V_{pre_o} \cap V_{eff_o} \\ \{(pre_o(v), pre_o(v))\} & \text{if } v \in V_{pre_o} \setminus V_{eff_o} \\ \{(d, eff(v)) \mid d \in \mathcal{D}_v\} & \text{if } v \in V_{eff_o} \setminus V_{pre_o} \\ \{(d, d) \mid d \in \mathcal{D}_v\} & \text{otherwise.} \end{cases}.$$

□

A.2. An Algorithmic View of Abstractions

THEOREM 4.5. *Let $\Theta = (S, L, T, s_0, S_\star)$ be a factored transition system, and let α_1 and α_2 be orthogonal abstractions of Θ. Then, $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2} = \Theta^{\alpha_1 \otimes \alpha_2}$.*

PROOF. Let $\Theta^{\alpha_1} = (S^1, L^1, T^1, s_0^1, S_\star^1)$, $\Theta^{\alpha_2} = (S^2, L^2, T^2, s_0^2, S_\star^2)$, $\alpha_{12} = \alpha_1 \otimes \alpha_2$, $\Theta^{\alpha_{12}} = (S^{12}, L^{12}, T^{12}, s_0^{12}, S_\star^{12})$, and $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2} = (S^\otimes, L^\otimes, T^\otimes, s_0^\otimes, S_\star^\otimes)$.

To prove $\Theta^{\alpha_{12}} = \Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$, we must show that $S^{12} = S^\otimes$, $L^{12} = L^\otimes$ with identical label costs, $T^{12} = T^\otimes$, $s_0^{12} = s_0^\otimes$, and $S_\star^{12} = S_\star^\otimes$. We show these five properties in sequence.

In the proofs of three of the properties, we will need to consider the recombination $\rho_V(s_1, s_2)$ of two states $s_1, s_2 \in S$. The set V will always be the set of state variables on which α_1 depends. Since V is always the same, we omit it from the notation for the rest of the proof. Note that, by the orthogonality of α_1 and α_2, $\rho(s_1, s_2)$ agrees with s_i on all state variables on which α_i depends, for $i = 1, 2$. This implies that $\alpha_1(\rho(s_1, s_2)) = \alpha_1(s_1)$ and $\alpha_2(\rho(s_1, s_2)) = \alpha_2(s_2)$. We will refer to this equality as "the orthogonality property" throughout the proof.

(i) $S^{12} = S^\otimes$: Consider $(s^1, s^2) \in S^{12}$. By definition of $\Theta^{\alpha_{12}}$, there exists $s \in S$ with $\alpha_{12}(s) = (s^1, s^2)$, which by definition of α_{12} means that $\alpha_i(s) = s^i$ and hence $s^i \in S^i$ for $i = 1, 2$. Therefore, $(s^1, s^2) \in S^1 \times S^2 = S^\otimes$. This shows that $S^{12} \subseteq S^\otimes$.

Conversely, consider $(s^1, s^2) \in S^\otimes$. Because $S^\otimes = S^1 \times S^2$, this means that $s^i \in S^i$ for $i = 1, 2$, and hence there exist states $s_i \in S$ with $\alpha_i(s_i) = s^i$ for $i = 1, 2$. Let $s = \rho(s_1, s_2)$. By the orthogonality property, $\alpha_i(s) = \alpha_i(s_i)$ for $i = 1, 2$, and hence $\alpha_{12}(s) = (\alpha_1(s), \alpha_2(s)) = (\alpha_1(s_1), \alpha_2(s_2)) = (s^1, s^2)$. Thus, there exists a state s which α_{12} maps to (s^1, s^2), and hence $(s^1, s^2) \in S^{12}$. We conclude that $S^\otimes \subseteq S^{12}$.

(ii) $L^{12} = L^\otimes$ with identical label costs: holds directly by definition since abstraction leaves the labels untouched, that is, $L^{12} = L^1 = L^2 = L = L^1 \cap L^2 = L^\otimes$ and the label costs are all the same.

(iii) $T^{12} = T^\otimes$: Consider $((s^1, s^2), l, (t^1, t^2)) \in T^{12}$. By definition of $\Theta^{\alpha_{12}}$, there exists a transition $(s, l, t) \in T$ with $\alpha_{12}(s) = (s^1, s^2)$ and $\alpha_{12}(t) = (t^1, t^2)$. By definition of α_{12}, we get $\alpha_i(s) = s^i$ and $\alpha_i(t) = t^i$ for $i = 1, 2$, and hence (s, l, t) induces the transition $(s^i, l, t^i) \in T^i$ for $i = 1, 2$. By definition of the synchronized product, we obtain $((s^1, s^2), l, (t^1, t^2)) \in T^\otimes$. This shows that $T^{12} \subseteq T^\otimes$.

Conversely, consider $((s^1, s^2), l, (t^1, t^2)) \in T^\otimes$. By definition of the synchronized product, we obtain $(s^i, l, t^i) \in T^i$ for $i = 1, 2$. By definition of Θ^{α_i}, there exist transitions $(s_i, l, t_i) \in T$ with $\alpha_i(s_i) = s^i$ and $\alpha_i(t_i) = t^i$ for $i = 1, 2$. Let $s = \rho(s_1, s_2)$ and $t = \rho(t_1, t_2)$. Because Θ is factored, we have $(s, l, t) \in T$. By the orthogonality property, $\alpha_i(s) = \alpha_i(s_i)$ and $\alpha_i(t) = \alpha_i(t_i)$, and thus the transition induced by (s, l, t) in $\Theta^{\alpha_{12}}$ is $(\alpha_{12}(s), l, \alpha_{12}(t)) = ((\alpha_1(s), \alpha_2(s)), l, (\alpha_1(t), \alpha_2(t))) = ((\alpha_1(s_1), \alpha_2(s_2)), l, (\alpha_1(t_1), \alpha_2(t_2))) = ((s^1, s^2), l, (t^1, t^2))$, and thus $((s^1, s^2), l, (t^1, t^2)) \in T^{12}$. We conclude that $T^\otimes \subseteq T^{12}$.

(iv) $s_0^{12} = s_0^\otimes$: We have $s_0^{12} = \alpha^{12}(s_0) = (\alpha^1(s_0), \alpha^2(s_0)) = (s_0^1, s_0^2) = s_0^\otimes$.

(v) $S_\star^{12} = S_\star^\otimes$: Consider $(s_\star^1, s_\star^2) \in S_\star^{12}$. By definition of $\Theta^{\alpha_{12}}$, there exists a solution state $s_\star \in S_\star$ with $\alpha_{12}(s_\star) = (s_\star^1, s_\star^2)$. By definition of α_{12}, we get $\alpha_i(s_\star) = s_\star^i$ for $i = 1, 2$. Thus, by definition of Θ^{α_i}, $s_\star^i \in S_\star^i$ for $i = 1, 2$. By definition of the synchronized product, we obtain $(s_\star^1, s_\star^2) \in S_\star^\otimes$. This shows that $S_\star^{12} \subseteq S_\star^\otimes$.

Conversely, consider $(s_\star^1, s_\star^2) \in S_\star^\otimes$. By definition of the synchronized product, we obtain $s_\star^i \in S_\star^i$ for $i = 1, 2$. By definition of Θ^{α_i}, there exist solution states $s_{\star i} \in S_\star$ with $\alpha_i(s_{\star i}) = s_\star^i$ for $i = 1, 2$. Let $s_\star = \rho(s_{\star 1}, s_{\star 2})$. Because Θ is factored, we have $s_\star \in S_\star$. By the orthogonality property, $\alpha_i(s_\star) = \alpha_i(s_{\star i})$, and thus the state induced by $s_\star$ in $\Theta^{\alpha_{12}}$ is $\alpha_{12}(s_\star) = (\alpha_1(s_\star), \alpha_2(s_\star)) = (\alpha_1(s_{\star 1}), \alpha_2(s_{\star 2})) = (s_\star^1, s_\star^2)$. Since $s_\star \in S_\star$, by definition of $\Theta^{\alpha_{12}}$ this implies that $(s_\star^1, s_\star^2) \in S_\star^{12}$. Thus, $S_\star^\otimes \subseteq S_\star^{12}$. □

A.3. Label Reduction

PROPOSITION 5.4. *Let Θ be a transition system, and let α be an abstraction of Θ. If τ is a conservative label reduction for Θ, then τ is a conservative label reduction for Θ^α.*

PROOF. We need to show that for all labels $l_1, l_2 \in L$ of Θ^α with $\tau(l_1) = \tau(l_2)$, every transition in Θ^α with label l_1 has a parallel transition with label l_2.

Let, l_1, l_2 be labels with $\tau(l_1) = \tau(l_2)$, and let (s', l_1, t') be a transition in Θ^α. By definition of abstract transition systems, there exist $s \in \alpha^{-1}(s')$ and $t \in \alpha^{-1}(t')$ so that (s, l_1, t) is a transition in Θ. Because τ is conservative for Θ, this implies that (s, l_2, t) is a transition in Θ, and hence by definition of abstract transition systems, $(\alpha(s), l_2, \alpha(t)) = (s', l_2, t')$ is a transition in Θ^α. □

PROPOSITION 5.5. *Let L be a set of labels, let Θ^1 and Θ^2 be transition systems using L, and let τ be a label reduction on L. If τ is conservative for both Θ^1 and Θ^2, then τ is conservative for $\Theta^1 \otimes \Theta^2$.*

PROOF. Let l_1 and l_2 be equivalent labels. Then $((s_1, s_2), l_1, (s_1', s_2'))$ is a transition in $\Theta^1 \otimes \Theta^2$ iff (by definition) (s_1, l_1, s_1') and (s_2, l_1, s_2') are transitions in Θ^1 and Θ^2 iff (by prerequisite) (s_1, l_2, s_1') and (s_2, l_2, s_2') are transitions in Θ^1 and Θ^2 iff (by definition) $((s_1, s_2), l_2, (s_1', s_2'))$ is a transition in $\Theta^1 \otimes \Theta^2$. □

LEMMA 5.6. *Let L be a set of labels, let Θ^1 and Θ^2 be transition systems using L, and let τ be a label reduction on L. If τ is conservative for Θ^2, then $\Theta^1|_\tau \otimes \Theta^2|_\tau = (\Theta^1 \otimes \Theta^2)|_\tau$.*

PROOF. Let $\Theta^1 = (S^1, L, T^1, s_0^1, S_\star^1)$, $\Theta^2 = (S^2, L, T^2, s_0^2, S_\star^2)$, and $\Theta^1 \otimes \Theta^2 = \Theta^{12} = (S^{12}, L, T^{12}, s_0^{12}, S_\star^{12})$. Further, let $\Theta^1|_\tau \otimes \Theta^2|_\tau = (S^A, L^\tau, T^A, s_0^A, S_\star^A)$, and $(\Theta^1 \otimes \Theta^2)|_\tau = (S^B, L^\tau, T^B, s_0^B, S_\star^B)$. It is obvious that $S^A = S^B$, $s_0^A = s_0^B$, $S_\star^A = S_\star^B$, and the label costs are identical. We need to show that $T^A = T^B$.

The direction $T^B \subseteq T^A$ is obvious: by reducing labels prior to the synchronized product, we can only ever introduce more combined transitions because we can only increase the sets of transitions on both sides that have a common label. More formally, assume that $((s_1, s_2), r, (s_1', s_2')) \in T^B$. Then, by definition, there exists $l \in L$ so that $r = \tau(l)$ and $((s_1, s_2), l, (s_1', s_2')) \in T^{12}$. By definition of the synchronized product, we get $(s_1, l, s_1') \in T^1$ and $(s_2, l, s_2') \in T^2$, which immediately leads us to $(s_1, \tau(l), s_1') \in T^1|_\tau$ and $(s_2, \tau(l), s_2') \in T^2|_\tau$. From there, we get $((s_1, s_2), \tau(l), (s_1', s_2')) \in T^A$ and hence $((s_1, s_2), r, (s_1', s_2')) \in T^A$ as desired. Note that in this argument we do not need to make any restrictions whatsoever on the structure of τ—as hinted, $T^B \subseteq T^A$ no matter how labels are reduced.

It remains to show $T^A \subseteq T^B$, that is, we need to show that, when reducing only labels that are equivalent in Θ^2, then no new combined transitions are introduced. Assume that $((s_1, s_2), r, (s_1', s_2')) \in T^A$. We need to show $((s_1, s_2), r, (s_1', s_2')) \in T^B$. Trying to simply reverse this argumentation, we first get that $(s_1, r, s_1') \in T^1|_\tau$ and $(s_2, r, s_2') \in T^2|_\tau$. This does not immediately show the presence of the combined transition in T^{12} because, prior to application of τ, the labels of these two transitions may be different. However, clearly, we do know that there exist $l^1, l^2 \in L$ so that (1) $\tau(l^1) = \tau(l^2) = r$, (2) $(s_1, l^1, s_1') \in T^1$, and (3) $(s_2, l^2, s_2') \in T^2$.

From (1) and because τ is conservative for Θ^2, we know that l^1 and l^2 are equivalent in Θ^2, and hence (3) implies $(s_2, l^1, s_2') \in T^2$. From this and (2), we get $((s_1, s_2), l^1, (s_1', s_2')) \in T^{12}$ and thus $((s_1, s_2), \tau(l^1), (s_1', s_2')) = ((s_1, s_2), r, (s_1', s_2')) \in T^B$. □

THEOREM 5.11. *Let Θ be a factored transition system with variables $\mathcal{V}$, and let $V \subseteq \mathcal{V}$. Let Γ be a conservative label reduction family for Θ, let $\mathcal{T}^\alpha$ be a merge-and-shrink tree over V for Θ. Then, $\Theta^\tau_\alpha = \Theta^\alpha|_{\tau^\emptyset}$ for any choice of pivot $v^* \in V$.*

PROOF. We prove by induction over the construction of $\mathcal{T}^\alpha$ that, for any intermediate merge-and-shrink abstraction β over $V' \subseteq V$:

$$\text{(*) } \Theta^\tau_\beta = \Theta^\beta \text{ if } v^* \notin V', \text{ and } \Theta^\tau_\beta = \Theta^\beta|_{\tau^{\overline{V'}}} \text{ if } v^* \in V'.$$

Obviously, the claim follows from the case where $\beta = \alpha$.

Clearly, (*) holds for rule (A) of Definition 3.6, that is, if $\mathcal{T}^\alpha = \mathcal{T}_v$ for a variable $v \in \mathcal{V}$. Since abstraction and label reduction do not interfere, (*) is invariant over rule (S). It remains to show that (*) is invariant over rule (M), where the root of $\mathcal{T}^\alpha$ has two children $\mathcal{T}^{\alpha_1}$ and $\mathcal{T}^{\alpha_2}$ over V_1 and V_2 such that $V_1 \cap V_2 = \emptyset$. By induction hypothesis, (*) holds for each of α_1 and α_2. We set $\beta = \alpha_1 \otimes \alpha_2$.

If $v^* \notin V_1 \cup V_2$, then $\Theta^\tau_\beta = \Theta^\tau_{\alpha_1} \otimes \Theta^\tau_{\alpha_2}$. By induction hypothesis, we have that $\Theta^\tau_{\alpha_1} = \Theta^{\alpha_1}$ and $\Theta^\tau_{\alpha_2} = \Theta^{\alpha_2}$. But then, (*) follows for β simply because (with the same argument as given for Corollary 4.6) α_1 and α_2 are orthogonal, and hence Theorem 4.5 applies.

Now, say, without loss of generality, that $v^* \in V_1$. We have $\Theta^\tau_\beta = (\Theta^\tau_{\alpha_1} \otimes \Theta^\tau_{\alpha_2}|_{\tau^{\overline{V_1}}})|_{\tau^{\overline{V_1 \cup V_2}}}$. By induction hypothesis we have that $\Theta^\tau_{\alpha_1} = \Theta^{\alpha_1}|_{\tau^{\overline{V_1}}}$, and that $\Theta^\tau_{\alpha_2} = \Theta^{\alpha_2}$. Thus we need to prove that $(\Theta^{\alpha_1}|_{\tau^{\overline{V_1}}} \otimes \Theta^{\alpha_2}|_{\tau^{\overline{V_1}}})|_{\tau^{\overline{V_1 \cup V_2}}} = \Theta^{\alpha_1 \otimes \alpha_2}|_{\tau^{\overline{V_1 \cup V_2}}}$. By prerequisite, $\tau^{\overline{V_1}}$ is conservative for $\Theta^{\pi_{\overline{V_1}}}$. Now, $V_2 \subseteq \overline{V_1}$ (due to $V_1 \cap V_2 = \emptyset$). Thus, $\tau^{\overline{V_1}}$ is conservative also for $\Theta^{\pi_{V_2}}$: π_{V_2} is an abstraction of $\pi_{\overline{V_1}}$ thus this follows with Proposition 5.4. Because Θ^{α_2} is an abstraction of $\Theta^{\pi_{V_2}}$, we can again apply Proposition 5.4 and get that $\tau^{\overline{V_1}}$ is conservative for Θ^{α_2}. We can thus apply Lemma 5.6 to $\Theta^{\alpha_1}|_{\tau^{\overline{V_1}}}$ and $\Theta^{\alpha_2}|_{\tau^{\overline{V_1}}}$, getting that $\Theta^{\alpha_1}|_{\tau^{\overline{V_1}}} \otimes \Theta^{\alpha_2}|_{\tau^{\overline{V_1}}} = (\Theta^{\alpha_1} \otimes \Theta^{\alpha_2})|_{\tau^{\overline{V_1}}}$. We can apply Theorem 4.5, and transform this into $\Theta^{\alpha_1 \otimes \alpha_2}|_{\tau^{\overline{V_1}}}$. Thus, it remains to show that $(\Theta^{\alpha_1 \otimes \alpha_2}|_{\tau^{\overline{V_1}}})|_{\tau^{\overline{V_1 \cup V_2}}} = \Theta^{\alpha_1 \otimes \alpha_2}|_{\tau^{\overline{V_1 \cup V_2}}}$. This is obvious because, by prerequisite, $\tau^{\overline{V_1 \cup V_2}} \circ \tau^{\overline{V_1}} = \tau^{\overline{V_1 \cup V_2}}$. □

PROPOSITION 5.13. *Let $\Theta = (S, L, T, s_0, S_\star)$ be a factored transition system with variables $\mathcal{V}$. Let $v \in \mathcal{V}$ be a variable, and let $l_1, l_2 \in L$. Then, l_1 and l_2 are equivalent in Θ^{π_v} if and only if $c(l_1) = c(l_2)$ and $\mathcal{D}^{\text{pe}}_v(l_1) = \mathcal{D}^{\text{pe}}_v(l_2)$.*

PROOF. For the "only if" direction, let l_1 and l_2 be equivalent in Θ^{π_v}. Then, $c(l_1) = c(l_2)$ follows by definition of equivalent labels. We prove $\mathcal{D}^{\text{pe}}_v(l_1) = \mathcal{D}^{\text{pe}}_v(l_2)$ by contradiction: assume $\mathcal{D}^{\text{pe}}_v(l_1) \neq \mathcal{D}^{\text{pe}}_v(l_2)$. Assume without loss of generality that $(d, d') \in \mathcal{D}^{\text{pe}}_v(l_1) \setminus \mathcal{D}^{\text{pe}}_v(l_2)$. By Proposition 2.4(ii), there exists a transition $(s_1, l_1, s_1') \in T$ where $s_1(v) = d$ and $s_1'(v) = d'$. Thus, Θ^{π_v} contains the transition (d, l_1, d'). On the other hand, by Proposition 2.4(ii) there does not exist a transition $(s_1, l_2, s_1') \in T$ where $s_1(v) = d$ and $s_1'(v) = d'$. Thus, Θ^{π_v} does not contain the transition (d, l_2, d'). Hence, l_1 and l_2 are not equivalent in Θ^{π_v}, as we needed to show.

For the "if" direction, say that $\mathcal{D}^{\text{pe}}_v(l_1) = \mathcal{D}^{\text{pe}}_v(l_2)$. Assume that (d, l_1, d') is a transition in Θ^{π_v}. We show that (d, l_2, d') also is a transition in Θ^{π_v}. This suffices because the same argument can be applied conversely, showing together with $c(l_1) = c(l_2)$ that l_1 and l_2 are equivalent in Θ^{π_v}.

By definition of abstract transition systems, there exist $s_1, s_1' \in S$ so that $s_1(v) = d$, $s_1'(v) = d'$, and $(s_1, l_1, s_1') \in T$. By Proposition 2.4(ii), $(s_1(u), s_1'(u)) \in \mathcal{D}^{\text{pe}}_u(l_1)$ for all $u \in \mathcal{V}$. Let W be the set of variables w where $\mathcal{D}^{\text{pe}}_w(l_1) \neq \mathcal{D}^{\text{pe}}_w(l_2)$. For each $w \in W$, let $(d(w), d'(w)) \in \mathcal{D}^{\text{pe}}_w(l_2)$; note here that $\mathcal{D}^{\text{pe}}_w(l_2)$ is not empty due to our assumption that pathologically useless labels have been removed from the transition system.

Define the state s_2 to be identical to s_1, except that $s_2(w) := d(w)$ for each $w \in W$. Define the state s_2' to be identical to s_1', except that $s_2'(w) := d'(w)$ for each $w \in W$. By construction, $(s_2(u), s_2'(u)) \in \mathcal{D}_u^{\text{pe}}(l_2)$ for all $u \in \mathcal{V}$. Hence, by Proposition 2.4(ii), $(s_2, l_2, s_2') \in T$. With this, by definition of abstract transition systems, $(s_2(v), l_2, s_2'(v))$ is a transition in Θ^{π_v}. By construction, $v \notin W$ and thus $s_2(v) = s_1(v) = d$ and $s_2'(v) = s_1'(v) = d'$. So (d, l_2, d') is a transition in Θ^{π_v}, which is what we needed to show. □

A.4. Bisimilarity

PROPOSITION 6.3. *Let Θ be a transition system, let α be an abstraction of Θ, and let β be an abstraction of Θ^α. If α is a bisimulation for Θ and β is a bisimulation for Θ^α, then $\beta \circ \alpha$ is a bisimulation for Θ.*

PROOF. Let $\Theta = (S, L, T, s_0, S_\star)$. Say that $s, s', t \in S$ so that (1) $(s, l, s') \in T$ and (2) $s \sim^{\beta\circ\alpha} t$. We need to prove that there exists $t' \in S$ so that (A) $(t, l, t') \in T$ and (B) $s' \sim^{\beta\circ\alpha} t'$.

By definition of $\sim^{\beta\circ\alpha}$, we have that $[s]_{\sim^\alpha} \sim^\beta [t]_{\sim^\alpha}$. Since β is a bisimulation for Θ^α, which is isomorphic to $\Theta/\!\sim^\alpha$, we know that there exists $[w]_{\sim^\alpha} \in S/\!\sim^\alpha$ so that (3) $([t]_{\sim^\alpha}, l, [w]_{\sim^\alpha}) \in T/\!\sim^\alpha$ and (4) $[s']_{\sim^\alpha} \sim^\beta [w]_{\sim^\alpha}$. Now, due to the definition of quotient systems, (3) implies that there exist $u, v \in S$ so that (5) $u \in [t]_{\sim^\alpha}$, (6) $v \in [w]_{\sim^\alpha}$, and (7) $(u, l, v) \in T$. Since α is a bisimulation for Θ, (7) together with (5) and (6) implies that there exists $t' \in S$ so that (A) $(t, l, t') \in T$ (obtaining one half of the result) and (8) $t' \sim^\alpha v$. Finally, (8) together with (6) implies that $t' \sim^\alpha w$ and hence (9) $t' \sim^{\beta\circ\alpha} w$, (4) implies that (10) $s' \sim^{\beta\circ\alpha} w$, and (9) and (10) together imply that $s' \sim^{\beta\circ\alpha} t'$ and hence we get (B) as well. □

PROPOSITION 6.4. *Let Θ^1 and Θ^2 be transition systems, and let α_1 and α_2 be abstractions for Θ^1 and Θ^2, respectively. If α_1 is a bisimulation for Θ^1, and α_2 is a bisimulation for Θ^2, then $\overline{\alpha_1} \otimes \overline{\alpha_2}$ is a bisimulation for $\Theta^1 \otimes \Theta^2$, where $\overline{\alpha_1}(s_1, s_2) := \alpha_1(s_1)$ and $\overline{\alpha_2}(s_1, s_2) := \alpha_2(s_2)$.*

PROOF. Let $\Theta^{12} = \Theta^1 \otimes \Theta^2$ where $\Theta^{12} = (S^{12}, L, T^{12}, s_0^{12}, S_\star^{12})$. Let $s = (s_1, s_2), s' = (s_1', s_2'), t = (t_1, t_2) \in S^{12}$ and $l \in L$, such that $s \sim^{\overline{\alpha_1}\otimes\overline{\alpha_2}} t$ and $(s, l, s') \in T^{12}$. To show that $\overline{\alpha_1} \otimes \overline{\alpha_2}$ is a bisimulation, we need to show that there exists $t' \in S^{12}$ such that $(t, l, t') \in T^{12}$ and $s' \sim^{\overline{\alpha_1}\otimes\overline{\alpha_2}} t'$.

Since $(s, l, s') \in T^{12}$, by definition of the synchronized product, we have that (1) $(s_1, l, s_1') \in T^1$ and (2) $(s_2, l, s_2') \in T^2$. Since $s \sim^{\overline{\alpha_1}\otimes\overline{\alpha_2}} t$, by definition of $\sim^{\overline{\alpha_1}\otimes\overline{\alpha_2}}$ we have that (3) $s_1 \sim^{\alpha_1} t_1$ and (4) $s_2 \sim^{\alpha_2} t_2$. This is because $(\overline{\alpha_1} \otimes \overline{\alpha_2})(s) = (\overline{\alpha_1}(s), \overline{\alpha_2}(s)) = (\alpha_1(s_1), \alpha_2(s_2))$ and similarly for t, so if $(\overline{\alpha_1} \otimes \overline{\alpha_2})(s) = (\overline{\alpha_1} \otimes \overline{\alpha_2})(t)$, then we have $\alpha_1(s_1) = \alpha_1(t_1)$ and $\alpha_2(s_2) = \alpha_2(t_2)$.

Due to (1) and (3) and because α_1 is a bisimulation for Θ^1, there exists t_1' such that (5) $(t_1, l, t_1') \in T^1$ and (6) $s_1' \sim^{\alpha_1} t_1'$. Due to (2) and (4) and because α_2 is a bisimulation for Θ^2, there exists t_2' such that (7) $(t_2, l, t_2') \in T^2$ and (8) $s_2' \sim^{\alpha_2} t_2'$.

Define $t' := (t_1', t_2')$. Then, by (5) and (7), we have $(t, l, t') \in T^{12}$. By (6) and (8), we have that $\alpha_1(s_1') = \alpha_1(t_1')$ and $\alpha_2(s_2') = \alpha_2(t_2')$, thus $(\alpha_1(s_1'), \alpha_2(s_2')) = (\alpha_1(t_1'), \alpha_2(t_2'))$, thus $(\overline{\alpha_1} \otimes \overline{\alpha_2})(s') = (\overline{\alpha_1} \otimes \overline{\alpha_2})(t')$, thus $s' \sim^{\overline{\alpha_1}\otimes\overline{\alpha_2}} t'$ as desired. □

COROLLARY 6.5. *Let Θ be a factored transition system with variables $\mathcal{V}$, and let $V \subseteq \mathcal{V}$. Let $\mathcal{T}^\alpha$ be a merge-and-shrink tree over V for Θ constructed so that in each application of Definition 3.6 rule (S) α_1 is a goal-respecting bisimulation for Θ^{α_2}. Then, α is a goal-respecting bisimulation for Θ^{π_V}.*

PROOF. This property clearly holds for atomic T^α. To see that the property is invariant over rule (S), note that by prerequisite α_1 is a goal-respecting bisimulation for Θ^{α_2}, and by induction hypothesis $\alpha_2 = \alpha_{2V'} \circ \pi_{V'}$ where $\alpha_{2V'}$ is a goal-respecting bisimulation for $\Theta^{\pi_{V'}}$. We can apply Proposition 6.3 to $\Theta^{\pi_{V'}}$, $\alpha_{2V'}$ and α_1, obtaining that $\alpha_1 \circ \alpha_{2V'}$ is a goal-respecting bisimulation for $\Theta^{\pi_{V'}}$. This is the same as saying that $\alpha_1 \circ \alpha_{2V'} \circ \pi_{V'} = \alpha_1 \circ \alpha_2$ is a goal-respecting bisimulation for $\Theta^{\pi_{V'}}$, so we are done. Considering property (M), suppose that α_1 is a goal-respecting bisimulation for $\Theta^{\pi_{V_1}}$ and α_2 is a goal-respecting bisimulation for $\Theta^{\pi_{V_2}}$. That is, $\alpha_1 = \alpha_{1V_1} \circ \pi_{V_1}$ and $\alpha_2 = \alpha_{2V_2} \circ \pi_{V_2}$, where α_{1V_1} is a goal-respecting bisimulation for $\Theta^{\pi_{V_1}}$ and α_{2V_2} is a goal-respecting bisimulation for $\Theta^{\pi_{V_2}}$. By Proposition 6.4, $\overline{\alpha_{1V_1}} \otimes \overline{\alpha_{2V_2}}$ is a goal-respecting bisimulation for $\Theta^{\pi_{V_1}} \otimes \Theta^{\pi_{V_2}}$, where $\overline{\alpha_{1V_1}}$ extends α_{1V_1} to $\Theta^{\pi_{V_1}} \otimes \Theta^{\pi_{V_2}}$ by ignoring the part of the state pertaining to $\Theta^{\pi_{V_2}}$, and similarly for $\overline{\alpha_{2V_2}}$. Since $\Theta^{\pi_{V_1}}$ and $\Theta^{\pi_{V_2}}$ are orthogonal, by Theorem 4.5 their synchronized product is equal to $\Theta^{\pi_{V_1 \cup V_2}}$. Obviously, since $V_1 \cap V_2 = \emptyset$, $\overline{\alpha_{1V_1}} = \alpha_{1V_1} \circ \pi_{V_1} = \alpha_1$ and $\overline{\alpha_{2V_2}} = \alpha_{2V_2} \circ \pi_{V_2} = \alpha_2$. Thus, $\alpha_1 \otimes \alpha_2$ is a goal-respecting bisimulation for $\Theta^{\pi_{V_1 \cup V_2}}$, as we needed to show. □

PROPOSITION 6.8. *There exist families of transition systems Θ with associated label reductions τ so that the coarsest goal-respecting bisimulation for $\Theta|_\tau$ is exponentially smaller than the coarsest goal-respecting bisimulation for Θ.*

PROOF. Consider the family of factored transition systems $\Theta(n) = (S(n), L(n), T(n), s_0(n), S_\star(n))$ with Boolean variables $\mathcal{V}(n) = \{v_1, \ldots, v_n\}$, with transitions labeled $l_1, \ldots l_n$ where $c(l_i) = 1$ and l_i applies if $v_i = 0$ and sets v_i to 1, where $s_0(n)$ assigns each v_i to 0, and where $S_\star(n)$ contains the single state assigning each v_i to 1. For any two different states s, t in $S(n)$, the labels of the outgoing transitions are different because the subset of variables with value 0 is different. Thus, s and t are not bisimilar, and the coarsest possible bisimulation is the identity function. The size of this abstraction is $|S(n)| = 2^n$.

Now, let $\tau(n)$ be a label reduction mapping every label to the same unique symbol; this is permissible since the label costs are uniform. Consider an abstraction $\alpha(n)$ that maps two states s, t to the same symbol if and only if $|\{v_i \mid s(v_i) = 1\}| = |\{v_i \mid t(v_i) = 1\}|$. The size of $\alpha(n)$ is $n + 1$. Clearly, $\alpha(n)$ is goal-respecting for $\Theta(n)|_{\tau(n)}$. It remains to show that $\alpha(n)$ is a bisimulation for $\Theta(n)|_{\tau(n)}$. To see this, say that s, t are states with $\alpha(n)(s) = \alpha(n)(t)$, and say that $(s, \tau(n)(l_j), s')$ is a transition in $\Theta(n)|_{\tau(n)}$. The transition in s sets variable v_j to 1 (which implies $s(v_j) = 0$). Because $|\{v_i \mid s(v_i) = 1\}| = |\{v_i \mid t(v_i) = 1\}|$, there exists a variable v_k with $t(v_k) = 0$. Thus, we have the transition $(t, \tau(n)(l_k), t')$, which satisfies $\tau(n)(l_j) = \tau(n)(l_k)$ and $\alpha(n)(s') = \alpha(n)(t')$. □

THEOREM 6.10. *Let Θ be a factored transition system with variables $\mathcal{V}$, and let $V \subseteq \mathcal{V}$. Let Γ be a conservative label reduction family for Θ, and let $\mathcal{T}^\alpha$ be a merge-and-shrink tree over V for Θ. If $\mathcal{T}^\alpha$ is constructed by label-reduced goal-respecting bisimulations, then α is a goal-respecting bisimulation for $\Theta^{\pi_V}|_{\tau^\emptyset}$.*

PROOF. Let v^* be the pivot variable in the construction of $\mathcal{T}^\alpha$. We prove by induction over the construction of $\mathcal{T}^\alpha$ that, for any intermediate merge-and-shrink abstraction β over V':

(*) β is a goal-respecting bisimulation for $\Theta^{\pi_{V'}}$ if $v^* \notin V'$, and β is a goal-respecting bisimulation for $\Theta^{\pi_{V'}}|_{\tau^{\overline{V'}}}$ if $v^* \in V$.

Obviously, the claim follows from the case where $\beta = \alpha$.

If $\mathcal{T}^\beta = \mathcal{T}_v$ for a variable $v \in \mathcal{V}$, that is, if $\mathcal{T}^\beta$ is the direct outcome of an application of rule (A) of Definition 3.6, then $\beta = \mathrm{id}_{\Theta^{\pi_v}} \circ \pi_v$, where $\mathrm{id}_{\Theta^{\pi_v}}$ is the identity function on Θ^{π_v}. But then, (*) holds trivially because the identity function induces, of course, a

goal-respecting bisimulation no matter if the labels in the transition system have been reduced.

If the root of $\mathcal{T}^\beta$ has a single child $\mathcal{T}^{\alpha_2}$, and $\beta = \alpha_1 \circ \alpha_2$, that is, if $\mathcal{T}^\beta$ is the direct outcome of an application of rule (S) of Definition 3.6, then (*) follows similarly as in the proof to Corollary 6.5. For $v^* \notin V'$, the argument is exactly the same. For $v^* \in V$, the argument is the same except that we also apply a label reduction. In detail, by induction hypothesis, we know that $\alpha_2 = \alpha_{2V'} \circ \pi_{V'}$ where $\alpha_{2V'}$ is a goal-respecting bisimulation for $\Theta^{\pi_{V'}}|_{\tau^{\overline{V'}}}$. By prerequisite, α_1 is a goal-respecting bisimulation for $\Theta^{\alpha_2}|_{\tau^{\overline{V'}}}$. Now, $\Theta^{\alpha_2} = \Theta^{\alpha_{2V'} \circ \pi_{V'}} = (\Theta^{\pi_{V'}})^{\alpha_{2V'}}$. Thus α_1 is a goal-respecting bisimulation for $[(\Theta^{\pi_{V'}})^{\alpha_{2V'}}]|_{\tau^{\overline{V'}}}$. Clearly, we can commute label reduction and abstraction, getting that α_1 is a goal-respecting bisimulation for $(\Theta^{\pi_{V'}}|_{\tau^{\overline{V'}}})^{\alpha_{2V'}}$. Thus, we can now apply Proposition 6.3 to $\Theta^{\pi_{V'}}|_{\tau^{\overline{V'}}}$, $\alpha_{2V'}$, and α_1, getting that $\alpha_1 \circ \alpha_{2V'}$ is a goal-respecting bisimulation for $\Theta^{\pi_{V'}}|_{\tau^{\overline{V'}}}$. But then, $\alpha_1 \circ \alpha_{2V'} \circ \pi_{V'} = \alpha_1 \circ \alpha_2$ is a goal-respecting bisimulation for $\Theta^{\pi_{V'}}|_{\tau^{\overline{V'}}}$, so we are done.

Say finally that β is the direct outcome of an application of rule (M) of Definition 3.6. Then the root of $\mathcal{T}^\beta$ has two children $\mathcal{T}^{\alpha_1}$ and $\mathcal{T}^{\alpha_2}$ over V_1 and V_2 such that $V_1 \cap V_2 = \emptyset$. By induction hypothesis, (*) holds for each of α_1 and α_2. We define $\beta = \alpha_1 \otimes \alpha_2$.

If $v^* \notin V_1 \cup V_2$ then, by induction hypothesis, we have that α_1 is a goal-respecting bisimulation for $\Theta^{\pi_{V_1}}$, and α_2 is a goal-respecting bisimulation for $\Theta^{\pi_{V_2}}$. Exactly as for Corollary 6.5, we can apply Proposition 6.4 and Theorem 4.5, and are done.

Say now without loss of generality that $v^* \in V_1$. We need to prove that $\alpha_1 \otimes \alpha_2$ is a goal-respecting bisimulation for $\Theta^{\pi_{V_1 \cup V_2}}|_{\tau^{\overline{V_1 \cup V_2}}}$. Since further label reduction can only make bisimilarity easier to achieve, obviously it suffices to prove that $\alpha_1 \otimes \alpha_2$ is a goal-respecting bisimulation for $\Theta^{\pi_{V_1 \cup V_2}}|_{\tau^{\overline{V_1}}}$. Since $V_1 \cap V_2 = \emptyset$, by Theorem 4.5, this is the same as saying that $\alpha_1 \otimes \alpha_2$ is a goal-respecting bisimulation for $(\Theta^{\pi_{V_1}} \otimes \Theta^{\pi_{V_2}})|_{\tau^{\overline{V_1}}}$. Now, similarly as argued in the proof of Theorem 5.11, since by prerequisite we have that $\tau^{\overline{V_1}}$ is conservative for $\Theta^{\pi_{\overline{V_1}}}$, with Proposition 5.4 we get that $\tau^{\overline{V_1}}$ is conservative for $\Theta^{\pi_{V_2}}$. Thus, we can apply Lemma 5.6, getting that $(\Theta^{\pi_{V_1}} \otimes \Theta^{\pi_{V_2}})|_{\tau^{\overline{V_1}}} = \Theta^{\pi_{V_1}}|_{\tau^{\overline{V_1}}} \otimes \Theta^{\pi_{V_2}}|_{\tau^{\overline{V_1}}}$. Hence, it suffices to prove that *(A) $\alpha_1 \otimes \alpha_2$ is a goal-respecting bisimulation for* $\Theta^{\pi_{V_1}}|_{\tau^{\overline{V_1}}} \otimes \Theta^{\pi_{V_2}}|_{\tau^{\overline{V_1}}}$. Now, by induction hypothesis, we have that *(B) α_1 is a goal-respecting bisimulation for* $\Theta^{\pi_{V_1}}|_{\tau^{\overline{V_1}}}$. We also have that α_2 is a goal-respecting bisimulation for $\Theta^{\pi_{V_2}}$. Since further label reduction can only make bisimilarity easier to achieve, we have that *(C) α_2 is a goal-respecting bisimulation for* $\Theta^{\pi_{V_2}}|_{\tau^{\overline{V_1}}}$.

The rest of the proof now consists of appropriately applying Proposition 6.4. (B) means that $\alpha_1 = \alpha_{1V_1} \circ \pi_{V_1}$ where α_{1V_1} is a goal-respecting bisimulation for $\Theta^{\pi_{V_1}}|_{\tau^{\overline{V_1}}}$. (C) means that $\alpha_2 = \alpha_{2V_2} \circ \pi_{V_2}$ where α_{2V_2} is a goal-respecting bisimulation for $\Theta^{\pi_{V_2}}|_{\tau^{\overline{V_1}}}$. By Proposition 6.4, $\overline{\alpha_{1V_1}} \otimes \overline{\alpha_{2V_2}}$ is a goal-respecting bisimulation for $\Theta^{\pi_{V_1}}|_{\tau^{\overline{V_1}}} \otimes \Theta^{\pi_{V_2}}|_{\tau^{\overline{V_1}}}$, where $\overline{\alpha_{1V_1}}$ extends α_{1V_1} to $\Theta^{\pi_{V_1}}|_{\tau^{\overline{V_1}}} \otimes \Theta^{\pi_{V_2}}|_{\tau^{\overline{V_1}}}$ by ignoring the part of the state pertaining to $\Theta^{\pi_{V_2}}|_{\tau^{\overline{V_1}}}$, and similarly for $\overline{\alpha_{2V_2}}$. Since $V_1 \cap V_2 = \emptyset$, $\overline{\alpha_{1V_1}} = \alpha_{1V_1} \circ \pi_{V_1} = \alpha_1$ and $\overline{\alpha_{2V_2}} = \alpha_{2V_2} \circ \pi_{V_2} = \alpha_2$. Thus, $\alpha_1 \otimes \alpha_2$ is a goal-respecting bisimulation for $\Theta^{\pi_{V_1}}|_{\tau^{\overline{V_1}}} \otimes \Theta^{\pi_{V_2}}|_{\tau^{\overline{V_1}}}$, which is the property (A) that we needed to show. □

A.5. Expressive Power

PROPOSITION 7.1. *Let $\Theta = (S, L, T, s_0, S_\star)$ be a factored transition system, let α_1 and α_2 be merge-and-shrink abstractions of Θ over disjoint variable sets V_1 and V_2, and let c_1, c_2 be a cost partitioning for Θ. Then, one can in time bounded by a polynomial in $|\alpha_1|$ and $|\alpha_2|$ construct a merge-and-shrink abstraction α_{12} with cost function $c_{12} \leq c$ so that $h^{\alpha_{12}c_{12}} \geq h^{\alpha_1 c_1} + h^{\alpha_2 c_2}$ and $|\alpha_{12}| = |\{h^{\alpha_{12}c_{12}}(s) \mid s \in S\}|$.*

PROOF. We first construct an abstraction $\overline{\alpha_{12}}$ of Θ by merging α_1 and α_2, that is, $\overline{\alpha_{12}} := \alpha_1 \otimes \alpha_2$. Because the variable sets underlying α_1 and α_2 are disjoint, $\overline{\alpha_{12}}$ is a

merge-and-shrink abstraction according to Definition 3.6. We associate $\overline{\alpha_{12}}$ with the modified cost function $c_{12} := c_1 + c_2$. Since c_1, c_2 is a cost partitioning, we have that $c_{12} \leq c$, that is, as desired c_{12} is bounded from above by the original cost function associated with Θ.

We next show that $h^{\overline{\alpha_{12}}c_{12}} \geq h^{\alpha_1 c_1} + h^{\alpha_2 c_2}$. Say, for arbitrary $s \in S$, that $\vec{p_{12}}$ is an optimal solution path for $\overline{\alpha_{12}}(s)$ in $\Theta^{\overline{\alpha_{12}}}$, under the cost function c_{12}. Note that each transition on this path has the form $((s_1, s_2), l, (s_1', s_2'))$, where s_i, s_i' are states in Θ^{α_i}. Define $\vec{p_i}$ to be the subsequence of $\vec{p_{12}}$ consisting of all transitions where $s_i \neq s_i'$. That is, $\vec{p_i}$ captures the part of the solution path pertaining to α_i. By definition of synchronized transition systems, each transition in $\vec{p_i}$ has a correspondence in Θ^{α_i}, so $\vec{p_i}$ is a solution path for $\alpha_i(s)$ in that system. Denote by $c_i(\vec{p_i})$ the cost of that solution path under the cost function associated with α_i; similar for $c_{12}(\vec{p_{12}})$. Clearly, $h^{\alpha_i c_i}(s) \leq c_i(\vec{p_i})$, and (by construction) $c_{12}(\vec{p_{12}}) \geq c_1(\vec{p_1}) + c_2(\vec{p_2})$. Putting this together, we get $h^{\overline{\alpha_{12}}c_{12}}(s) = c_{12}(\vec{p_{12}}) \geq c_1(\vec{p_1}) + c_2(\vec{p_2}) \geq h^{\alpha_1 c_1}(s) + h^{\alpha_2 c_2}(s)$ as desired.

Now, we construct α_{12} by shrinking $\overline{\alpha_{12}}$ so that states $s, t \in S^{\overline{\alpha_{12}}}$ are aggregated iff their solution cost within that abstraction is the same. That is, we reduce $\overline{\alpha_{12}}$ to the minimal abstraction required to represent the same heuristic function. Obviously, α_{12} satisfies the requirements of the claim. □

THEOREM 7.2. *Let $\Theta = (S, L, T, s_0, S_\star)$ be a factored transition system, let $\alpha_1, \ldots, \alpha_k$ be merge-and-shrink abstractions of Θ over pairwise disjoint variable sets $V_1, \ldots, V_k$, and let $c_1, \ldots, c_k$ be a cost partitioning for Θ. Then, for each $2 \leq j \leq k$, there exists a merge-and-shrink abstraction $\alpha_{1\ldots j}$ with cost function $c_{1\ldots j} \leq c$ so that $h^{\alpha_{1\ldots j}c_{1\ldots j}} \geq \sum_{i=1}^{j} h^{\alpha_i c_i}$, $|\alpha_{1\ldots j}| = |\{h^{\alpha_{1\ldots j}c_{1\ldots j}}(s) \mid s \in S\}|$ where $h^{\alpha_{1\ldots j}c_{1\ldots j}}(s) \leq \sum_{i=1}^{j} \max_{s \in S} h^{\alpha_i c_i}(s)$ for all $s \in S$, and $\alpha_{1\ldots j}$ can be constructed in time bounded by a polynomial in $|\alpha_1|, \ldots, |\alpha_j|, |\alpha_{12}|, \ldots, |\alpha_{1\ldots(j-1)}|$.*

PROOF. For the case $j = 2$, the construction is the same as the one underlying Proposition 7.1, except for how we obtain α_{12} from $\overline{\alpha_{12}}$. We do so by shrinking $\overline{\alpha_{12}}$ in two steps. The first step is the same as before, aggregating states $s, t \in S^{\overline{\alpha_{12}}}$ iff their solution cost within that abstraction is the same. In the second step, we remove any separate classes of states whose solution cost exceeds the sum of the maximum costs in α_1 and α_2, that is, we aggregate all states s, t whose solution cost is at least $\max_{s \in S} h^{\alpha_1 c_1}(s) + \max_{s \in S} h^{\alpha_2 c_2}(s)$. Afterwards, α_{12} obviously satisfies the claimed properties regarding c_{12}, $h^{\alpha_{12}c_{12}}$, and $|\alpha_{12}|$.

If $j > 2$, then we first construct $\overline{\alpha_{123}}$ by merging α_{12} with α_3, and we associate $\overline{\alpha_{123}}$ with the cost function $c_{123} := c_{12} + c_3 = \sum_{i=1}^{3} c_i$. We construct α_{123} from $\overline{\alpha_{123}}$ by first aggregating all states with identical solution cost, then aggregating all states whose solution cost is at least $\sum_{i=1}^{3} \max_{s \in S} h^{\alpha_i c_i}(s)$. The same arguments as before show that $h^{\alpha_{123}c_{123}}$ satisfies the claimed properties regarding c_{123}, $h^{\alpha_{123}c_{123}}$, and $|\alpha_{123}|$. We iterate this construction up to j.

Regarding runtime, the crucial point is that constructing each $\overline{\alpha_{1\ldots j}}$ and its heuristic takes time polynomial in $|\alpha_{1\ldots(j-1)}|$ and $|\alpha_j|$, and does not depend in any other way on the size of the synchronized product of the original abstractions $\alpha_1, \ldots, \alpha_{j-1}$. Since constructing $\alpha_{1\ldots j}$ is of course polynomial in $|\overline{\alpha_{1\ldots j}}|$, the claim follows. □

THEOREM 7.4. *There exist families of transition systems where (a) there exists an abstraction strategy for which merge-and-shrink yields a perfect heuristic in polynomial time, whereas (b) there exists no ensemble of polynomial-size projections $\pi_{V_1}, \ldots, \pi_{V_k}$ with cost partitioning $c_1, \ldots, c_k$, which can guarantee that $\sum_{i=1}^{k} h^{\pi_{V_i} c_i}(s_0) \geq (\frac{1}{2} + \varepsilon) h^*(s_0)$, for any $\varepsilon > 0$.*

PROOF. We slightly extend the construction from Example 6.6. Consider the family of factored transition systems $\Theta(n) = (S(n), L(n), T(n), s_0(n), S_\star(n))$ defined as follows. There are the Boolean variables $\mathcal{V}(n) = \{v_0, v_1, \ldots, v_n\}$. The transitions are labeled $l_0, l_1, \ldots l_n$, where the label costs are uniformly 1. Transition l_0 applies if $v_0 = 0$, and sets v_0 to 1. For $j > 0$, transition l_j applies if $v_0 = 1$ and $v_j = 0$; it sets v_j to 1 and it sets v_0 to 0. The initial state s_0 assigns every v_j to 0, and $S_\star$ contains the single state assigning every v_j to 1. In other words, we extend Example 6.6 by introducing a variable v_0 that must be set to 1 in between the moves on any of the variables $v_1, \ldots, v_n$.

Say that, when constructing the merge-and-shrink heuristic α, in Step (ii) we always aggregate (partial) states s and t iff $s(v_0) = t(v_0)$ and $|\{v_j \mid j > 0, s(v_j) = 1\}| = |\{v_j \mid j > 0, t(v_j) = 1\}|$, that is, iff they agree on the count of variables with index $j > 0$ already set to 1. It is easy to see that this yields a perfect heuristic and that abstraction size is linear in n. This shows part (a) of the claim.

Let $\pi_{V_1}, \ldots, \pi_{V_k}$ be any ensemble of projections, with cost partitioning $c_1, \ldots, c_k$. Consider the initial state s_0. The solution cost is $2n + 1$, owing to n moves for variables $j > 0$, and $n + 1$ moves for v_0. We show that $\sum_{i=1}^k h^{\pi_{V_i} c_i}(s_0)$ incurs the claimed error due to accounting for only a logarithmic number of v_0 moves. Consider any individual projection onto variable subset V_i. If $v_0 \in V_i$, then an optimal solution in $\Theta^{\pi_{V_i}}$ includes $|V_i| + 1$ moves of v_0. If $v_0 \notin V_i$, no such moves are included. In both cases, the cost pertaining to v_0 in $h^{\pi_{V_i} c_i}(s_0)$ is bounded from above by $(|V_i|+1) \cdot c_i(l_0)$. Now, the size of $\Theta^{\pi_{V_i}}$ is $2^{|V_i|}$. By prerequisite, this is polynomial in the input size n, so $|V_i| \in O(\log n)$. Thus, the cost pertaining to v_0 in $h^{\pi_{V_i} c_i}(s_0)$ is bounded by $(A_i + B_i \log n) \cdot c_i(l_0)$, for some constants A_i and B_i. Summing this up over the ensemble $\pi_{V_1}, \ldots, \pi_{V_k}$, we get the upper bound $\sum_{i=1}^k (A_i + B_i \log n) \cdot c_i(l_0) \leq (A + B \log n) \cdot \sum_{i=1}^k c_i(l_0)$ for some constants A and B. Because $c_1, \ldots, c_k$ is a cost partitioning, $(A + B \log n) \cdot \sum_{i=1}^k c_i(l_0) \leq (A + B \log n) \cdot c(l_0) = A + B \log n$. Clearly, the contribution of variables $v_1, \ldots, v_n$ to $\sum_{i=1}^k h^{\pi_{V_i} c_i}(s_0)$ is at most n, so altogether we have $\sum_{i=1}^k h^{\pi_{V_i} c_i}(s_0) \leq n + A + B \log n$. The ratio between this and $h^*(s_0) = 2n+1$ converges to $\frac{1}{2}$ which proves part (b) of the claim. □

We now provide the proof of Theorem 7.8. In order to do so, we first need to define the critical-path heuristics h^m. These are defined for planning tasks only (not for factored transition systems in general), so we stick to that formalism here. We assume a planning task $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$ with state space $\Theta(\Pi) = (S, L, T, s_0, S_\star)$.

To define h^m, we first need the concept of *regression*. Given a partial variable assignment s_G, the set of regressions of s_G, denoted $R(s_G)$, contains all pairs (o, s'_G) where $o = (pre_o, eff_o)$ is an operator and $s'_G = (s_G \setminus eff_o) \cup pre_o$ is a partial variable assignment, such that eff_o and s_G are consistent (i.e., do not contain conflicting assignments to any variable) and $eff_o \cap s_G \neq \emptyset$ (else, there is no point in trying to achieve s_G using o). A regression $(o, s'_G) \in R(s_G)$ has the property that if $\vec{p}$ is a path leading to s'_G, then $\vec{p} \circ \langle o \rangle$ is a path leading to s_G. Thus, the Bellman equation for the optimal cost function h^* can be formulated as:

$$h^*(s, s_G) = \begin{cases} 0 & \text{if } s_G \subseteq s \\ \min_{(o, s'_G) \in R(s_G)} h^*(s, s'_G) + cost_o & \text{otherwise.} \end{cases} \quad (1)$$

Note here that, deviating from Definition 2.9, we define the heuristic as a function of two variable assignments, the first one being the state s we evaluate (previously the only argument), the second one being a subgoal to achieve. This is merely a notational convenience allowing to formulate the recursion in Eq. (1).

The h^m heuristic function is obtained by recursively approximating the cost of achieving an assignment to more than m variables by the highest cost of achieving the assignment restricted to any subset of size m or less. Formally, it is defined as the pointwise

greatest solution to this equation:

$$h^m(s, s_G) = \begin{cases} 0 & \text{if } s_G \subseteq s \\ \min_{(o,s'_G)\in R(s_G)} h^m(s, s'_G) + cost_o & \text{if } |s_G| \leq m \\ \max_{s'_G \subset s_G, |s'_G|\leq m} h^m(s, s'_G) & \text{otherwise.} \end{cases} \quad (2)$$

Given a vocabulary V of propositional variables, let $\varphi_W = \bigvee_{\{p,q\}\in W}(p \wedge q)$ be a propositional formula where W is allowed to be any subset of the set of all distinct pairs of propositional variables in V. We construct a family of planning tasks corresponding to φ_W, designed so that the state space encodes all variable assignments, and so that those states containing a pair in W have solution cost 1, whereas those not containing such a pair have solution cost > 1.

Given a set $W \subseteq V \times V$ as previously mentioned, the planning task Π_W is defined as follows. There is one Boolean state variable p for each $p \in V$, and another Boolean variable L. The initial state assigns each $\mathtt{p} = 0$ and $\mathtt{L} = 0$. The goal is $\mathtt{L} = 1$. We have the following operators.

—For every $p \in V$, an operator `true(p)` with cost 0, no precondition, and effects $\mathtt{p} = 1$ as well as $\mathtt{q} = 0$ for all q such that $\{p, q\} \in W$.
—For every pair $\{p, q\} \in W$, an operator `set(p,q)` with cost 1, no precondition, and effects $\mathtt{p} = 1$, $\mathtt{q} = 1$.
—For every pair $\{p, q\} \in W$, an operator `goal(p,q)` with cost 1, preconditions $\mathtt{p} = 1$, $\mathtt{q} = 1$, and effect $\mathtt{L} = 1$.

This construction allows to reach at 0 cost any state that does not contain a pair in W. An optimal plan is `set(p,q)` followed by `goal(p,q)`, where $\{p, q\} \in W$.

LEMMA A.1. *Let $V = \{p_1, \ldots, p_n\}$, $W \subseteq V \times V$, and $\varphi_W = \bigvee_{\{p,q\}\in W}(p \wedge q)$. Let $S_{\overline{W}}$ be the set of assignments to V that do not satisfy φ_W. Let Π_W be the planning task defined above, and let $\Theta(\Pi_W) = (S, L, T, s_0, S_\star)$. Let α be any abstraction with $h^\alpha(s_0) = h^*(s_0) = 2$. Then, $\{s \mid s \in S, h^\alpha(s) = 2\} = \{a \cup \{L = 0\} \mid a \in S_{\overline{W}}\}$.*

PROOF. Let s be a state with $s = a \cup \{\mathtt{L} = 0\}$ for $a \in S_{\overline{W}}$. Clearly, $h^\alpha(s) \leq 2$. Assume $h^\alpha(s) < 2$. By construction, we can reach s from s_0 by a sequence of 0-cost `true(p)` operators. From the consistency of h^α we get $h^\alpha(s_0) \leq 0 + h^\alpha(s) < 2$, in contradiction. For the other direction, say that s is a state with $h^\alpha(s) = 2$, and assume to the contrary that either $s(\mathtt{L}) = 1$, or s does contain a pair in W. If $s(\mathtt{L}) = 1$, then s is a goal state, and hence $h^*(s) = 0$, a contradiction because h^α is admissible. Finally, consider the case where $s(\mathtt{L}) = 0$ and s does contain a pair $\{p, q\}$ in W. But then, we can apply `goal(p,q)` and thus $h^*(s) \leq 1$, again a contradiction to the admissibility of h^α. This proves the claim. □

LEMMA A.2. *Let $V = \{p_1, \ldots, p_n\}$, $W \subseteq V \times V$, and $\varphi_W = \bigvee_{\{p,q\}\in W}(p \wedge q)$. Let Π_W be the planning task defined previously. Then, h^2 is a perfect heuristic for $\Theta(\Pi_W)$.*

PROOF. Say $\Pi_W = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$. We show that *(*) any subgoal s_G that can be generated by regression on $s_\star$ satisfies $|s_G| \leq 2$.* This proves that $h^2(s, s_\star) = h^*(s, s_\star)$: We can simplify Eq. (1) by considering only subgoals of size at most 2 in the second line. The same can be done in Eq. (2), and consequently the third line in Eq. (2) can be dropped (it will never be considered). The simplified equations are identical, proving the claim.

It remains to prove (*). This obviously holds for $s_\star$ itself. The operators over which $s_\star$ can be regressed have the form `goal(p,q)` where $\{p, q\} \in W$—these are the only ones affecting L—generating a subgoal of form (1) $s_G = \{\mathtt{p} = 1, \mathtt{q} = 1\}$ where $\{p, q\} \in W$. Subgoals of this form cannot be regressed over any operator `true(p')`: if $p' \notin \{p, q\}$ then the intersection of the effect with s_G is empty; and if $p' \in \{p, q\}$, say without loss

of generality $p' = p$, then `true(p')` has the effect `q` $= 0$ which is not consistent with the assignment `q` $= 1$ in s_G. Thus, subgoal (1) can be regressed only over operators `set(p',q')` where $\{p', q'\} \cap \{p, q\} \neq \emptyset$. If $\{p', q'\} = \{p, q\}$, then the subgoal generated is (2) $s_G = \emptyset$ which cannot be regressed any further. Otherwise, say without loss of generality that $p' = p$ and $q' \neq q$. Then, the subgoal generated is (3) $s_G = \{\texttt{q} = 1\}$. This can be regressed either over an operator `set(p',q')` where $q \in \{p', q'\}$, yielding the empty subgoal (2); or over the operator `true(q)`, generating again the empty subgoal. These are all subgoals that may be generated. They all satisfy (*), concluding the argument. □

THEOREM 7.8. *There exist families of factored transition systems where h^2 is a perfect heuristic, whereas unless* $\mathbf{P} = \mathbf{NP}$ *there exists no polynomial-time abstraction strategy so that merge-and-shrink yields abstractions α guaranteeing that $h^\alpha(s_0) = h^*(s_0)$.*

PROOF. The family of factored transition systems we consider are the state spaces $\Theta(\Pi_W) = (S, L, T, s_0, S_\star)$ of the planning tasks Π_W defined previously, for arbitrary $V = \{p_1, \ldots, p_n\}$, $W \subseteq V \times V$, and $\varphi_W = \bigvee_{\{p,q\} \in W}(p \wedge q)$.

Assume that we have a merge-and-shrink abstraction strategy for the planning tasks Π_W, yielding abstractions α guaranteeing that $h^\alpha(s_0) = h^*(s_0)$, and that this strategy runs in polynomial time. This implies it can only construct abstractions where $|\mathcal{T}^\alpha|^{\max}$ is bounded by a polynomial. Thus, by Theorem 7.6, we can use the representation of α, that is, the merge-and-shrink tree and the abstract state space (or table of heuristic values obtained from it), to count, in polynomial time, the number of states with $h^\alpha(s) = 2$, which by Lemma A.1 is the number of assignments over V that are not models of φ_W. Subtracting this from 2^n, we obtain the number of models of φ_W. As discussed in Section 7.5, counting the satisfying assignments for φ_W is **#P**-hard and thus this implies that $\mathbf{P} = \mathbf{NP}$. The claim follows directly from this and Lemma A.2. □

ACKNOWLEDGMENTS

We thank Gabriele Röger for her help with the figures of transition systems and Moritz Gronbach for his contribution to the merge-and-shrink implementation. We also thank the anonymous reviewers for their useful comments.

REFERENCES

Christer Bäckström and Bernhard Nebel. 1995. Complexity results for SAS^+ planning. *Computat. Intell.* 11, 4, 625–655.

R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. 1993. Algebraic decision diagrams and their applications. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'93)*. Michael R. Lightner and Jochen A. G. Jess, Eds., 188–191.

Marcel Ball and Robert C. Holte. 2008. The compression power of symbolic pattern databases, In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*. Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, Eds., AAAI Press, 2–11.

Mark Boddy, Johnathan Gohde, Tom Haigh, and Steven Harp. 2005. Course of action generation for cyber security using classical planning. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS'05)*. Susanne Biundo, Karen Myers, and Kanna Rajan, Eds., AAAI Press, 12–21.

Tom Bylander. 1994. The computational complexity of propositional STRIPS planning. *Artif. Intell.* 69, 1–2, 165–204.

Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. 2000. *Model Checking*. The MIT Press.

Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages*. 238–252.

Nadia Creignou and Miki Hermann. 1996. Complexity of generalized satisfiability counting problems. *Inf. Computat.* 125, 1, 1–12.

Joseph C. Culberson and Jonathan Schaeffer. 1998. Pattern databases. *Computat. Intell.* 14, 3, 318–334.

Adnan Darwiche and Pierre Marquis. 2002. A knowledge compilation map. *J. Artif. Intell. Res.* 17, 229–264.

Rina Dechter and Judea Pearl. 1985. Generalized best-first search strategies and the optimality of A*. *J. ACM* 32, 3, 505–536.

Minh B. Do, Wheeler Ruml, and Rong Zhou. 2008. Planning for modular printers: Beyond productivity, In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*. Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, Eds., AAAI Press 68–75.

Carmel Domshlak, Malte Helmert, Erez Karpas, Emil Keyder, Silvia Richter, Gabriele Röger, Jendrik Seipp, and Matthias Westphal. 2011. BJOLP: The big joint optimal landmarks planner. In *IPC 2011 Planner Abstracts*, 91–95.

Klaus Dräger, Bernd Finkbeiner, and Andreas Podelski. 2006. Directed model checking with distance-preserving abstractions. In *Proceedings of the 13th International SPIN Workshop (SPIN'06)*. Lecture Notes in Computer Science, Antti Valmari, Ed., vol. 3925, Springer-Verlag, 19–34.

Klaus Dräger, Bernd Finkbeiner, and Andreas Podelski. 2009. Directed model checking with distance-preserving abstractions. *Int. J. Softw. Tools Techn. Trans.* 11, 1, 27–37.

Stefan Edelkamp. 2001. Planning with pattern databases. In *Pre-proceedings of the 6th European Conference on Planning (ECP'01)*. Amedeo Cesta and Daniel Borrajo, Eds., 13–24.

Stefan Edelkamp. 2002. Symbolic pattern databases in heuristic search planning. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02)*. Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, Eds., AAAI Press, 274–283.

Stefan Edelkamp. 2003. Promela planning. In *Proceedings of the 10th International SPIN Workshop (SPIN'03)*. Lecture Notes in Computer Science, Thomas Ball and Sriram K. Rajamani, Eds., vol. 2648, Springer-Verlag, 197–212.

Ariel Felner, Richard Korf, and Sarit Hanan. 2004. Additive pattern database heuristics. *J. Artif. Intell. Res.* 22, 279–318.

Richard E. Fikes and Nils J. Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.* 2, 189–208.

Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* 4, 2, 100–107.

Patrik Haslum, Blai Bonet, and Héctor Geffner. 2005. New admissible heuristics for domain-independent planning. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05)*. AAAI Press, 1163–1168.

Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI'07)*. AAAI Press, 1007–1012.

Patrik Haslum and Héctor Geffner. 2000. Admissible heuristics for optimal planning. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'00)*. Steve Chien, Subbarao Kambhampati, and Craig A. Knoblock, Eds., AAAI Press, 140–149.

Malte Helmert. 2004. A planning heuristic based on causal graph analysis. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*. Shlomo Zilberstein, Jana Koehler, and Sven Koenig, Eds., AAAI Press, 161–170.

Malte Helmert. 2006a. The Fast Downward planning system. *J. Artif. Intell. Res.* 26, 191–246.

Malte Helmert. 2006b. Solving planning tasks in theory and practice. Ph.D. Dissertation, Albert-Ludwigs-Universität Freiburg.

Malte Helmert and Carmel Domshlak. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*. Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, Eds., AAAI Press, 162–169.

Malte Helmert, Patrik Haslum, and Jörg Hoffmann. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*. Mark Boddy, Maria Fox, and Sylvie Thiébaux, Eds., AAAI Press, 176–183.

Malte Helmert and Hauke Lasinger. 2010. The Scanalyzer domain: Greenhouse logistics as a planning problem, In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*. Ronen brafman, Héctor Geffner, Jörg Hoffmann, and Henry Kautz, Eds., AAAI Press, 234–237.

Robert Holte, Ariel Felner, Jack Newton, Ram Meshulam, and David Furcy. 2006. Maximizing over multiple pattern databases speeds up heuristic search. *Artif. Intell.* 170, 16–17, 1123–1136.

Robert C. Holte. 2010. Common misconceptions concerning heuristic search. In *Proceedings of the 3rd Annual Symposium on Combinatorial Search (SoCS'10)*. Ariel Felner and Nathan Sturtevant, Eds., AAAI Press, 46–51.

Gerard J. Holzmann. 2004. *The SPIN Model Checker – Primer and Reference Manual*. Addison-Wesley.

Robert E. Horton. 1945. Erosional development of streams and their drainage basins; Hydrophysical approach to quantitative morphology. *Bull. Geol. Soc. Amer.* 56, 275–370.

Michael Katz and Carmel Domshlak. 2008. Optimal additive composition of abstraction-based admissible heuristics, In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*. Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, Eds., AAAI Press 174–181.

Michael Katz, Jörg Hoffmann, and Malte Helmert. 2012. How to relax a bisimulation? In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press, 101–109.

Alexander Koller and Jörg Hoffmann. 2010. Waking up a sleeping rabbit: On natural-language sentence generation with FF. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*. Ronen Brafman, Héctor Geffner, Jörg Hoffmann, and Henry Kautz, Eds., AAAI Press 238–241.

Alexander Koller and Ronald Petrick. 2011. Experiences with planning for natural language generation. *Computat. Intell.* 27, 1, 23–40.

Richard E. Korf. 1997. Finding optimal solutions to Rubik's cube using pattern databases. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'97)*. AAAI Press, 700–705.

Richard E. Korf and Weixiong Zhang. 2000. Divide-and-conquer frontier search applied to optimal sequence alignment. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00)*. Henry Kautz and Bruce Porter, Eds., AAAI Press, 910–916.

Gianfranco Lamperti and Marina Zanella. 2003. *Diagnosis of Active Systems*. Kluwer Academic Publishers.

Jorge Lucangeli Obes, Carlos Sarraute, and Gerardo Richarte. 2010. Attack planning in the real world. In *Proceedings of the AAAI Workshop on Intelligent Security (SecArt)*. 10–17.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003, Yale Center for Computational Vision and Control.

Robin Milner. 1980. A Calculus of Communicating Systems. Lecture Notes in Computer Science, vol. 92, Springer-Verlag.

Robin Milner. 1990. Operational and algebraic semantics of concurrent processes. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, Jan van Leeuwen, Ed., Elsevier and MIT Press, 1201–1242.

Bernhard Nebel. 2000. On the compilability and expressive power of propositional planning formalisms. *J. Artif. Intell. Res.* 12, 271–315.

Allen Newell and Herbert A. Simon. 1963. GPS: A program that simulates human thought. In *Computers and Thought*, E. A. Feigenbaum and J. Feldman, Eds., Oldenbourg, 279–293.

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

Edwin P. D. Pednault. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*. Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, Eds., Morgan Kaufmann, 324–332.

Silvia Richter and Malte Helmert. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*. AAAI Press, 273–280.

Wheeler Ruml, Minh Binh Do, Rong Zhou, and Markus P. J. Fromherz. 2011. On-line planning and scheduling: An application to controlling modular printers. *J. Artif. Intell. Res.* 40, 415–468.

Krishnan K. Sabnani, Aleta M. Lapone, and M. Ümit Uyar. 1989. An algorithmic procedure for checking safety properties of protocols. *IEEE Trans. Commun.* 37, 9, 940–948.

Antti Valmari. 1989. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets (APN'89)* Lecture Notes in Computer Science, Grzegorz Rozenberg, Ed., vol. 483, Springer-Verlag, 491–515.

Received July 2012; revised June 2013; accepted December 2013